NTNU
Norwegian University of
Science and Technology

System Identification of a High-Speed USV, with Genetic Programming

TTK4551 - Engineering Cybernetics,
Specialization Project

Gisle Finstad Halvorsen

Department of Engineering Cybernetics
Norwegian University of Science and Technology
June 2, 2019

Preface

This report presents the results from my specialization project during the fall 2018, with 7.5 credits. The project was carried out for the Norwegian Defence Research Establishment (FFI). This project will be continued further during the spring 2019, in my Master's thesis work.

I would like to thank my supervisors, Else-Line Ruud and Martin Syre Wiig, that has been of great help to me throughout this semester.

Abstract

This report considers a data-driven approach to system identification of the Unmanned surface vessel Odin, operating in the displacement, semi-displacement and planing regions. The technique used in this report is Genetic Programming (GP) with Symbolic Regression (SR). This approach was tested on different systems, to get a better insight into the algorithm. GP was also used to find a model dynamic model of Odin, but with poor performance.

Contents

List of Figures

Acronyms

GPS	Global Positioning system
IMU	Inertial Measurment Unit
ANN	Artificial Neural Network
USV	Unmanned Surface vehicle
GP	Genetic Programming
SR	Symbolic regression
DOF	Degree Of Freedom
SOG	Speed Over Ground
ROT	Rate Of Turn
MSD	Mass Spring Damper
NED	North East Down
OLS	Orthogonal Least Squares
MSE	Mean Square Error

Chapter 1

Introduction

1.1 Motivation

In the past two decades, a lot of research has been put into developing unmanned surface vehicles (USV's) for various applications. This is in part due to to reduced cost and effectiveness of commercially available global positioning systems (GPS's), internal measuring units (IMU's) and reliable wireless communication systems.

The main reason to develop USV's is to reduce the operational cost, improved personnel safety and increasing the range of operation. USV's are most suited in the dull, dangerous, dirty and harsh environments.

The applications of a USV is numerous. From scientific research of the ocean from a biological point of view or environmental mission to sample or monitor. Ocean resource exploration of oil and gas or offshore maintenance of platforms or pipelines. To military uses of reconnaissance and mine countermeasures, and a lot more.

1.2 Problem formulation

To effectively control a USV a sufficiently precise mathematical model of the system is useful. The goal of this project is to find a model that describes the dynamics of the high-speed USV Odin, in all tree phases: displacement, semi-displacement and planing phase.

The modeling approach used in this report is the data-driven model-free Genetic programming (GP) with Symbolic regression (SR). This was chosen because it results in differential equations of the systems kinetics. This in turn makes the design of controller easier, and stability properties possible to find.

1.3 Literature review

When describing the dynamics of a marine vessel the vast majority uses the model described in Fossen (2011). This model is used when modelling a craft in the displacement phase. Models for high speed vessels are described in Faltinsen (2005). The previous approaches are derived from physical principles.

A more data driven approach is described in Eriksen and Breivik (2017), where the identification of model parameters using linear regression are described. Another way to find the parameters of the model using a artificial neural network (ANN) is described in Rajesh and Bhattacharyya (2008).

Model free approach to finding a model using genetic programming (GP) and symbolic regression (SR) is described in Moreno-Salinas et al. (2015), using techniques described in Madar et al. (2002). A more in depth analysis on Evolutionary computing, which GP is a subgroup of, is described in Eiben and Smith (2012).

1.4 Odin

Odin's propulsion system consists of two Hamilton jets which are located in the aft of the ship. The jets generate thrust when water is forced in a rearward direction. The jets are equipped with an astern deflector, also called reversing bucket, and a steering nozzle.

The steering nozzle is controllable and changes the horizontal direction of the water stream, with a range of $[-27°, 27°]$. This creates a moment in yaw, and forces in surge and sway.

The position of the astern deflector is in the range $[-1, 1]$, where 1 is fully open, and -1 is fully closed. When the deflector is at 0, the stream is divided into three components, one in the aft direction, and two in the fore direction, negating each other and resulting in zero force. When the deflector is fully closed, the stream is in the fore direction, resulting in a negative force in surge. The water jet is illustrated in Figure 1.2.

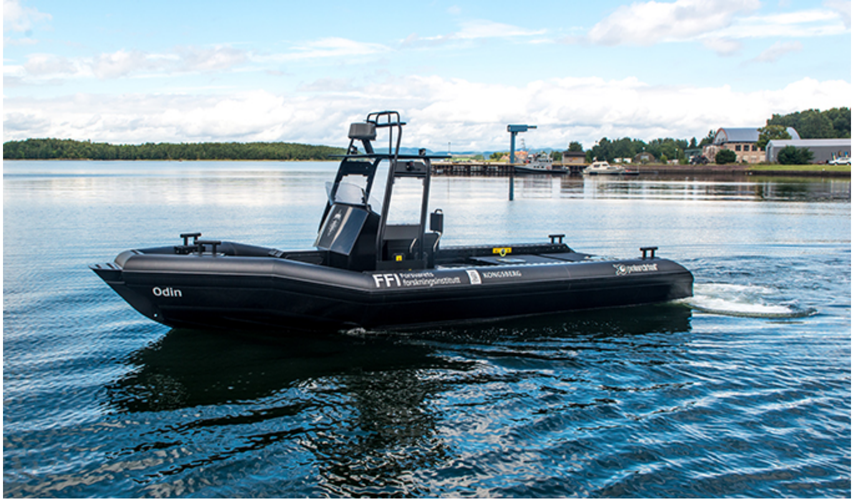


Figure 1.1: Odin

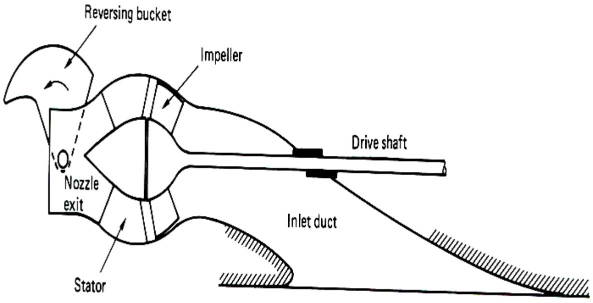


Figure 1.2: Water jet

When Odin is in standard operation, the 2 water-jets have the same angle. This makes the system act like a standard ship model with a rudder and a propeller. This is how the vessel is viewed throughout the report.

1.5 Outline

The report is organized as follows:

- Chapter 2 describes the theory that the work in this report builds on.
- Chapter 3 gives an analysis of the genetic programming with symbolic regression algorithm tested on the less complex, mass-spring-damper system, and the 3-DOF model from Fossen (2011).
- Chapter 4 presents model formulations and prepossessing steps, before the genetic programming algorithm is tested on real data gathered by Odin.
- Chapter 5 contains discussions about the results
- Chapter 6 concludes the report

Chapter 2

Theory

When a vessel is modeled for control purposes, the 3-DOF (degree of freedom) model from Fossen (2011) is often used. This model describes the kinetics and kinematics of the vessel and is shown in eq. (2.1 and 2.2).

$$\dot{\eta} = R(\psi)\nu \tag{2.1}$$

$$M\dot{\nu} + C_{RB}(\nu) + C_A(\nu_r)\nu + D(\nu_r)\nu_r = \tau \tag{2.2}$$

where $\eta = [N\ E\ \psi]^T$ is the vessels pose, $\nu = [u\ v\ r]^T$ is the vessels velocity, and ν_r is the relative velocity between the vessel and the water. τ is the sum of forces acting on the vessel.

The limitation of the model is that it is only valid for a surface vessel in the displacement phase. There are three phases, the displacement, semi-displacement and planing phase. To find which phases a vessel is operating in, the Froude number is common approximation to use Fossen (2011). The Froude number is calculated as shown in eq.(2.3).

$$Fn = \frac{U_r}{\sqrt{Lg}} \tag{2.3}$$

where U_r is the speed of the vessel relative to the water, L is the overall submerged length of the vessel, and g is the acceleration of gravity. For a Froude number below **0.4**, the vessel is in the displacement phase. From **0.4 to 1.1** the vessel is in the semi-displacement phase, and above **1.1** the vessel is in the planing phase. As Odin will be operating in all three phase, the model above will not give sufficiently accurate representation of the dynamics.

2.1 High-speed modelling

There are different ways to find the desired model of the high-speed USV. One of these would be the model suggested in Eriksen and Breivik (2017). It suggests to use a 2-DOF model instead of the 3-DOF. The reason behind this is that the USV is under-actuated. This means that there are fewer actuators than the degree of freedom, which is also true for Odin. The new states are therefore speed over ground (SOG) U and rate of turn (ROT) r. The input to the kinetic equation $\tau = [\tau_m \ \tau_\delta]$, where τ_m is the motor throttle and τ_δ is the rudder input. The proposed model is shown below:

$$\dot{\eta} = \begin{bmatrix} \cos\chi & 0 \\ \sin\chi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} U \\ r \end{bmatrix} \tag{2.4}$$

$$\dot{\chi} = r + \dot{\beta} \tag{2.5}$$

$$M(x)\dot{x} + \sigma(x) = \tau \tag{2.6}$$

where $x = [U \ r]^T$, the inertia matrix $M(x) = diag(m_U(x), m_r(x))$, and the damping term $\sigma(x) = [\sigma_U \ \sigma_r]^T$. The parameters of the matrices $M(x)$ and $\sigma(x)$ needs to be identified. This is done by linear regression on data gathered by a grid search in steps in rudder and throttle, with the vessel reaching steady state in between.

2.2 Model-free approach

A model free approach is where no *a priori* knowledge is used to find the dynamics. Here there are no matrices or pre-defined parameters to identify (also called nonparametric identification). The theory behind this is to find the model composition and parameters that best fit the data.

2.2.1 Genetic programming with symbolic regression

Genetic programming is inspired by nature. The principle, like evolution, is to have the fittest individuals reproduce to make better offspring. This is done multiple times, generations, to eventually reach the best possible individual. In this context the individuals are models, and the 'fittest' models are the ones that best fit the empirical data.

Genetic programming with symbolic regression results in the individuals being symbolic expressions. These are easily represented by a tree structure as in Figure 2.1.

In the first iteration the models are stochastically chosen to create high diversity. The models are tested with some fitness function, usually MSE, to determine which models best fit the empirical data. A subset of the best models is used to make the new generation.

The new generation of individuals could be chosen from the last generation in three different ways: direct reproduction, mutation or crossover. In direct reproduction the individual is transferred directly into the next generation. In mutation a random change is imposed on the individual before it is transferred to the next generation. The crossover type of making new individuals, is to take two individuals from the last generation, and randomly choose a crossover point. Then combining the two individuals to form two new for the next generation. In each iteration there will be some of each of these types of operations preformed, with the probabilities of mutation, crossover and direct reproduction being p_m, p_c and $1 - p_m - p_c$ respectably.

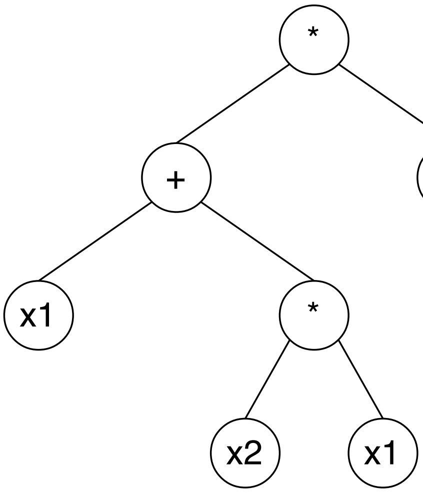


Figure 2.1: Structure of a tree

One of the parameters of this algorithm is what operations between the variables are chosen. The simplest operators $\{+, -, *\}$, are the easiest to implement. If division is included, the singularity of dividing by zero might occur, and that will cause the algorithm to fail. Another operator that have a similar problem is the square root, where rooting by a negative number results in a complex number. There are different ways to work around this, but for the scope of this report, only the simplest operators will be used.

This might pose some limitations on the final result, as other models of surface vessels have the absolute operator, $|.|$, in them. In Eriksen and Breivik (2017) the *tanh* operator is used in the model.

Bloat and Orthogonal Least Squares

Bloat is a phenomenon that occurs when training a model with GP, and is often referred to as *survival of the fattest*. Bloat is when the average tree size grows during the run of the training, and results in a far to complex model. OLS is used in GP to avoid bloat, and hence reduce the complexity of the resulting model, make it less prone to overfitting, and easier to interpret. The main idea is to remove the terms that contribute less to the accuracy of the model. The concept is shown in Figure (2.2), where the function F1 was eliminated. This is done in every fitness evaluation.

The way the algorithm decides to cut away a branch or not is based on the error reduction ratio *err*. This is a measure of what contribution each branch has on the accuracy of the model. When the *err* is below a certain threshold, the branch is cut. How to calculate the *err* is described in (Madar et al. (2002)).

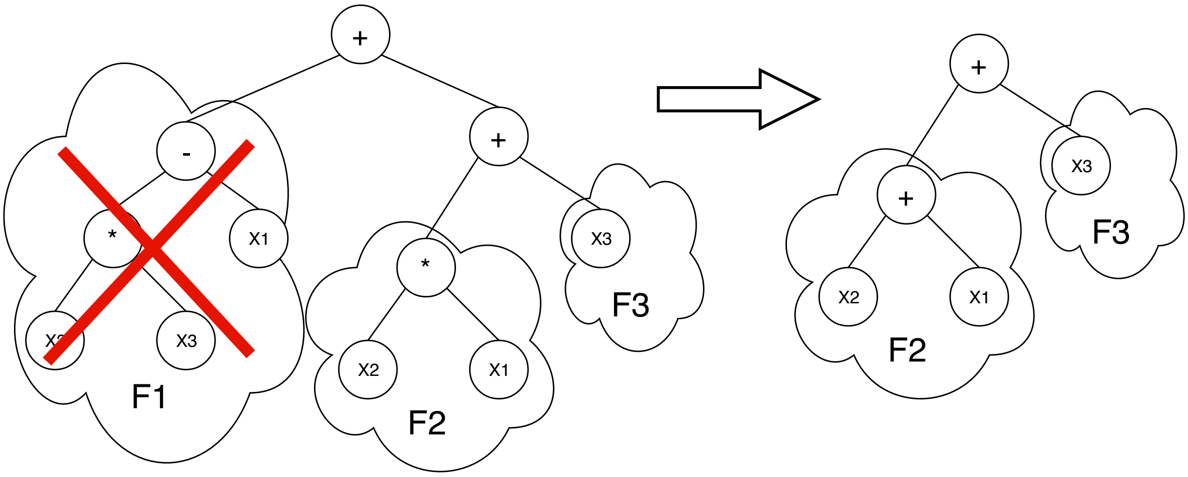


Figure 2.2: OLS

2.3 Maneuvers that describe a system

Different maneuvers executed by a surface vessel to generate data is described in the literature. A series of step responses and Kempf's zig-zag are described in more detail.

Series of step responses

In Eriksen and Breivik (2017) they conducted a series of step responses, in a grid from 0 to 100% on the throttle and the rudder. Then let the vessel reach steady state, and do a full turn after reaching steady state to account for external disturbances. Assuming that the vessel response in yaw is symmetric, such that only positive yaw angle experiments are necessary. This was done to to identify damping and inertial terms, for both steady-state and transient information.

This data is meant to describe the dynamics of a high-speed USV in all three phases, and might me more applicable for Odin than the following manoeuvre.

Kempf's zig-zag maneuver

Another way of creating data for vessel system identification is described in Kempf (1944), and used in (Moreno-Salinas et al. (2015)) and (Rajesh and Bhattacharyya (2008)). Where the chosen inputs are a square wave on the rudder, and a constant desired surge speed. This results in a zig-zag movement of the ship. This simple but effective manoeuvre should be enough to define the most important characteristics of the ships dynamics to be used for control purposes, (Fossen (2002)). This is close to the actual data gather on Odin fall 2018, as described in sec. 4.2.1. This maneuver is ment for vessels that only operate in the displacement phase, and therefore might not be the best choice of maneuvers for describing a vessel all three phases.

Chapter 3

Genetic programming on simple systems

3.1 Algorithm

The Genetic Programming done in this report is built on the Toolbox Abonyi (2014). It applies OLS (orthogonal least squares), see Section 2.2.1, as proposed by the same author in (Madar et al. (2002)).

3.1.1 Hyperparameters

Hyperparameters are an important part of any machine learning algorithm. Different values were tested, but the algorithm converged to approximately the same model regardless of the values. The parameter that had the most impact on the result were *max tree-depth*, which resulted in a complex model for high values, and *population size*, which with a smaller number resulted in more iterations before it converged. As suggested in Section 6.4 in the book Eiben and Smith (2012), the probability of mutation is much lower than the probability of crossover.

The hyperparameters used for most of the following training in Chapter 3 and 4 is listed in the table 3.1.

Table 3.1: Table of the hyperparameters used in the GP algorithm

Hyperparameter	**Value**	
Probability of crossover	0.9	
Probability of mutation	0.1	
Probability of direct reproduction	0	
Population size	2000	
OLS threshold	0.05	
Max tree-depth	6	
Operators	+ - *	
Iterations	25-200	depending on the complexity

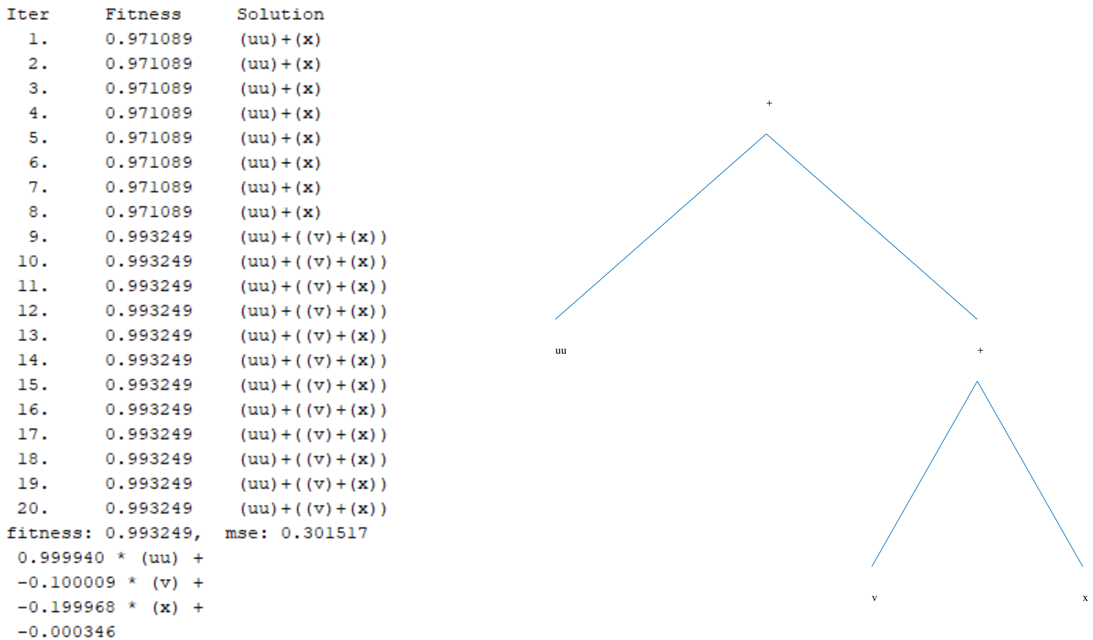
3.2 Mass-Spring-Damper

To better understand how the Genetic programming algorithm works, it strengths and weaknesses, it was tested on a simpler system than the data from Odin. Simulations of a Mass Spring Damper (MSD) system, implemented in Matlab-simulink, was run to generate training data.

The equation for the mass spring damper:

$$m\ddot{x} + d\dot{x} + kx = u \tag{3.1}$$

Without any noise or time delays introduced in the system, the GP algorithm seems robust enough to find the correct model, regardless of the input force imposed on the system. With the exception of a constant input. The algorithm converged after about 2 to 12 iterations.



```
Iter     Fitness     Solution
  1.     0.971089    (uu)+(x)
  2.     0.971089    (uu)+(x)
  3.     0.971089    (uu)+(x)
  4.     0.971089    (uu)+(x)
  5.     0.971089    (uu)+(x)
  6.     0.971089    (uu)+(x)
  7.     0.971089    (uu)+(x)
  8.     0.971089    (uu)+(x)
  9.     0.993249    (uu)+((v)+(x))
 10.     0.993249    (uu)+((v)+(x))
 11.     0.993249    (uu)+((v)+(x))
 12.     0.993249    (uu)+((v)+(x))
 13.     0.993249    (uu)+((v)+(x))
 14.     0.993249    (uu)+((v)+(x))
 15.     0.993249    (uu)+((v)+(x))
 16.     0.993249    (uu)+((v)+(x))
 17.     0.993249    (uu)+((v)+(x))
 18.     0.993249    (uu)+((v)+(x))
 19.     0.993249    (uu)+((v)+(x))
 20.     0.993249    (uu)+((v)+(x))
fitness: 0.993249,  mse: 0.301517
 0.999940 * (uu) +
 -0.100009 * (v) +
 -0.199968 * (x) +
 -0.000346
```

(a) Solution over each iteration, where uu = u

(b) The resulting tree structure

Figure 3.1

3.2.1 Introducing noise in the system

When only a small amount of white noise was introduced in the output of the system, the algorithm converges close to the real model of the system. The algorithm fails to find the true model, only when the noise was on the same scale as the signal.

To check that the resulting model is not overfitted to the data, the test was also run with a training set containing 20% of the data. The algorithm preformed equally well.

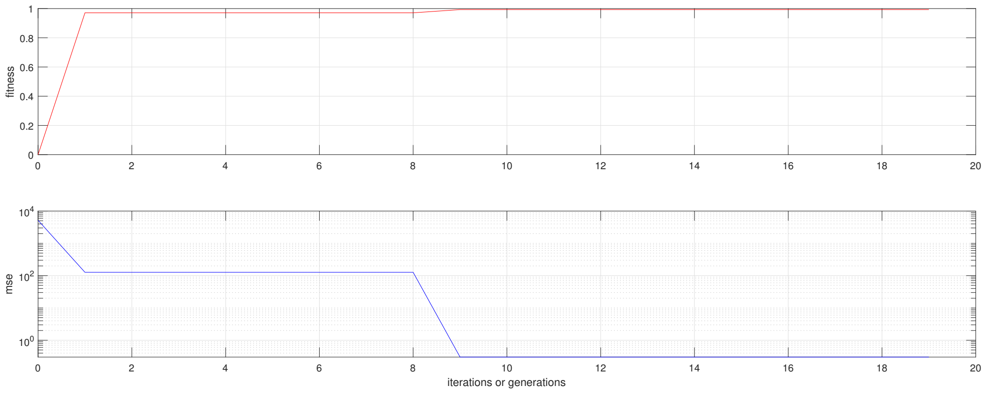


Figure 3.2: Fitness and mean square error over iterations for the model with best fit in a generation. Fitness of 1 is a perfect fit

An example of a resulting system with $m = 10, k = 2, d = 1$.

$$\begin{aligned} GP_{prediction}: &\quad \ddot{x} = -0.199968x + 0.999940u - 0.100009v - 0.000346 \\ Model: &\quad \ddot{x} = \frac{1}{m}(-dv - kx + u), \quad \dot{x} = v \end{aligned}$$

Figures (A.2 - A.5) show the result from the GP algorithms resulting models, with the input shown in Figure (A.1) in the appendix.

3.2.2 Time delay

To test the robustness of the GP algorithm regarding delays, a simple delay was implemented in the system. The delay is from the input to the system. The algorithm gets the original input and not the delayed signal. Figure 3.3 illustrates how the delay was implemented.

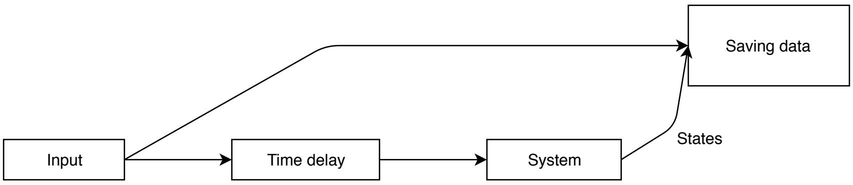


Figure 3.3: Implemented delay in the system

The GP algorithm has no inherited tools to deal with time delays, as it only has the ability to use the simple opperators, $[+, -, *]$. In Figure 3.4 the result from running the GP algorithm with a time delay of 0.5s is shown, and shows the problem clearly. The algorithm finds about the same model as before, but does not fit the data well.

It is assumed that the real system, Odin, as most systems has some time delays and a method of dealing with it has to be found in order to identify the model correctly.

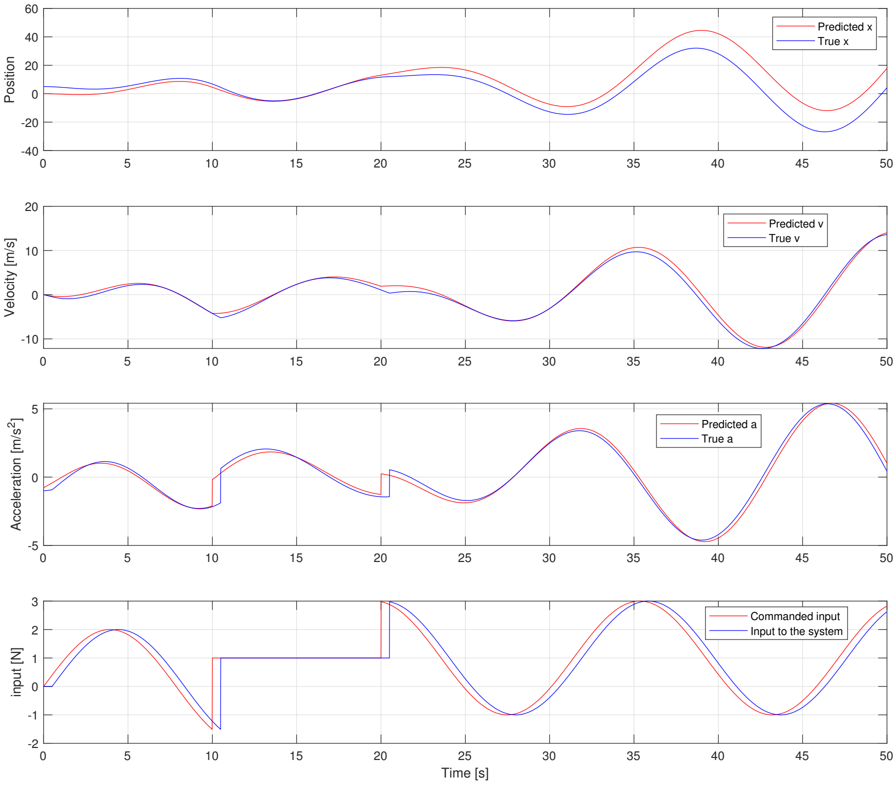


Figure 3.4: Result from the GP algorithm. x and v is found by implementing a simple Forward Euler method on the acceleration and on the resulting velocity.

3.2.3 A method for dealing with time delays

For the scope of this report, a simple brute force method that look at the input to the system n time steps ago, was implemented. This could be done in the training or in the resulting model. The drawback from including this after the training would be that the algorithm has tried to fit a model to the delayed systems data. Then when including the brute force method afterwards would result in a poorer system performance. Including a time delay compensation in the training, would feed the compensated input to the algorithm, and keep the remaining states the same. The two different approaches are shown in Figure 3.5.

Delaying the input $u(t)$ with n steps:

$$u_t = \begin{cases} u_0 & \text{if } t \leq n \\ u_{t-n} & \text{else} \end{cases}$$

while the other states remain the same. The time delay in this system is known, $n = delay/samplingtime$.

This works perfectly when the time delay is constant and known. But when the time delay is unknown, n needs to be estimated. Including it in the training would be the most correct way of doing it, but it would mean that the algorithm has to train for every possible n. This would be time consuming for large systems, but for the

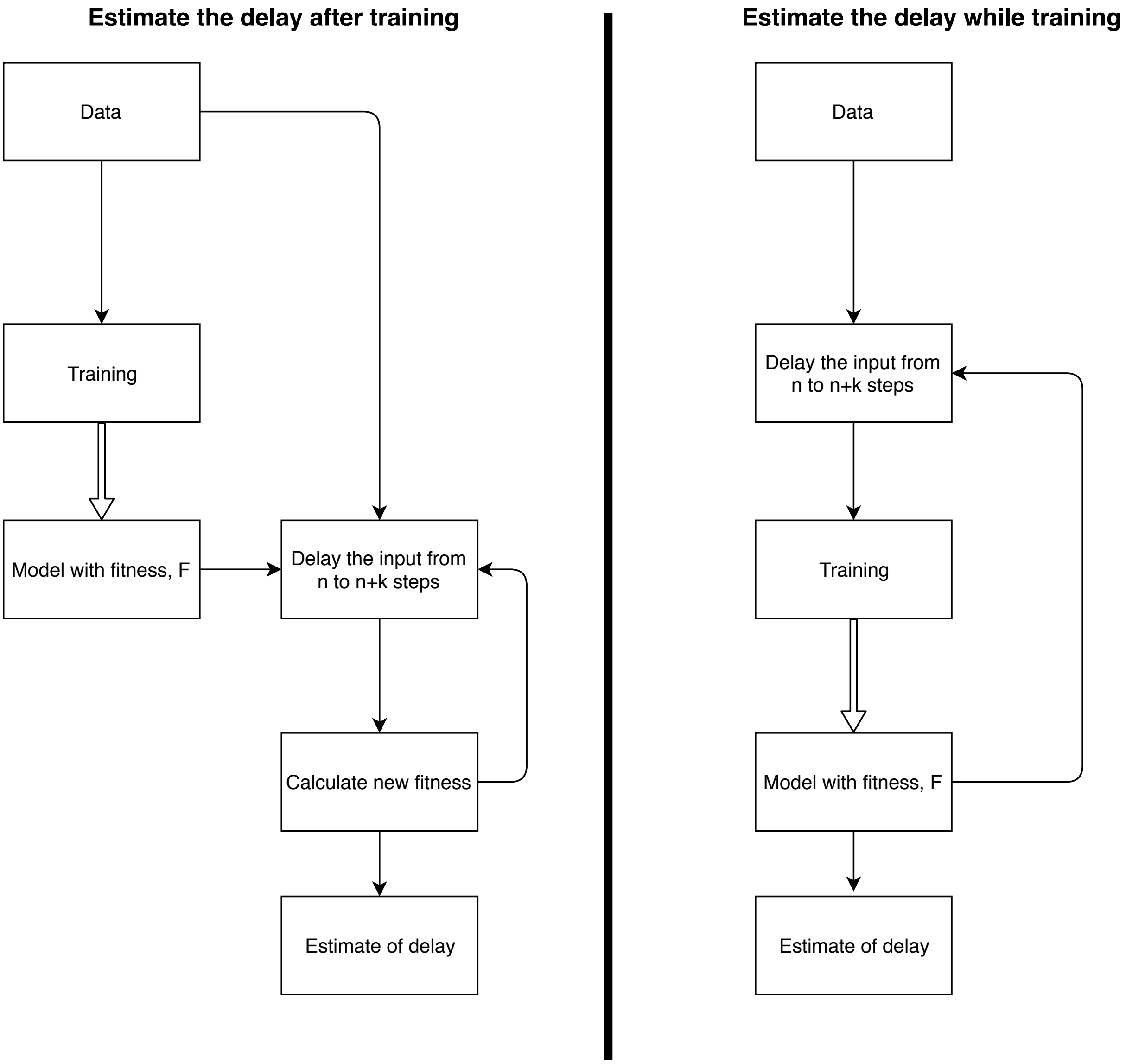

Estimate the delay after training
Data
Training
Model with fitness, F
Delay the input from n to n+k steps
Calculate new fitness
Estimate of delay
Estimate the delay while training
Data
Delay the input from n to n+k steps
Training
Model with fitness, F
Estimate of delay

Figure 3.5: Showing the different approaches to time delay estimation

MSD system it would be possible.

Looping over every n, and fitting the system to that set of inputs u, the time delay implemented in the system becomes apparent from the resulting plot in Figure 3.6

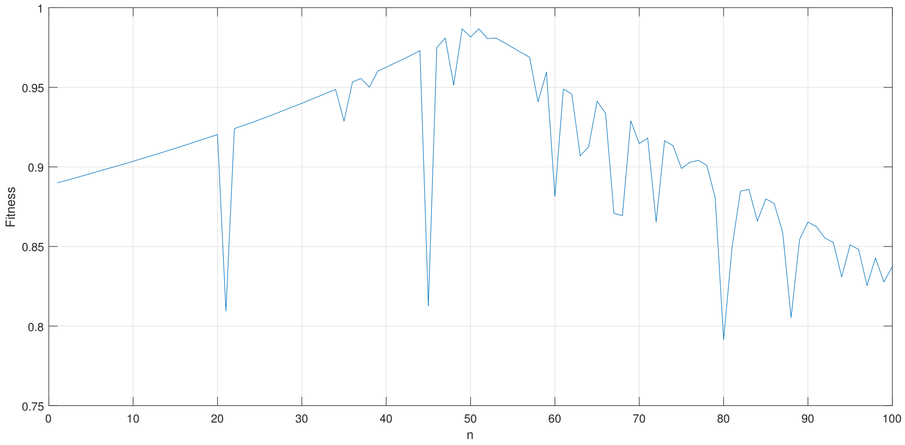


Figure 3.6: A plot of the resulting models fitness for each n. The time delay implemented was 0.5s, with a sampling time of 1e-2.

Some noise was implemented for this test, and that is probably why the algorithm does not find a good model of the system for every n. And that's why the algorithm does not reach a fitness of 1, when n = 50.

3.3 Data from the 3-DOF simulator of the ship

To test if the algorithm works on a more complex system, the algorithm was tested on data generated by a 3-DOF model simulation of Odin.

The 3 DOF model:

$$\begin{aligned} &Kinematics: \\ &\begin{bmatrix} \dot{x}_b^n \\ \dot{y}_b^n \\ \dot{\psi}_b^n \end{bmatrix} = \begin{bmatrix} \cos(\psi_b^n) & -\sin(\psi_b^n) & 0 \\ \sin(\psi_b^n) & \cos(\psi_b^n) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_b \\ v_b \\ r_b \end{bmatrix} \end{aligned} \tag{3.2}$$

$$\begin{aligned} &Dynamics: \\ &M\dot{v}_{b/n}^b + C(v_{b/n}^b)v_{b/n}^b + D(v_{b/n}^b)v_{b/n}^b = \tau \end{aligned} \tag{3.3}$$

Where n is the NED (North-East-Down) coordinate frame, and b is the body frame. $\mathbf{M} = \mathbf{M}^T > 0$ is the mass and inertial matrix. $\mathbf{C}$ contains Coriolis and centripetal terms, and $\mathbf{D}$ is the hydrodynamic damping.

In this model the gravitational term is neglected, as only surge, sway and yaw is considered. By assuming port-starboard symmetry of the vessel, and that the origin of b lies along the centerline of the vessel, the following structure for **M** and **D** can be found:

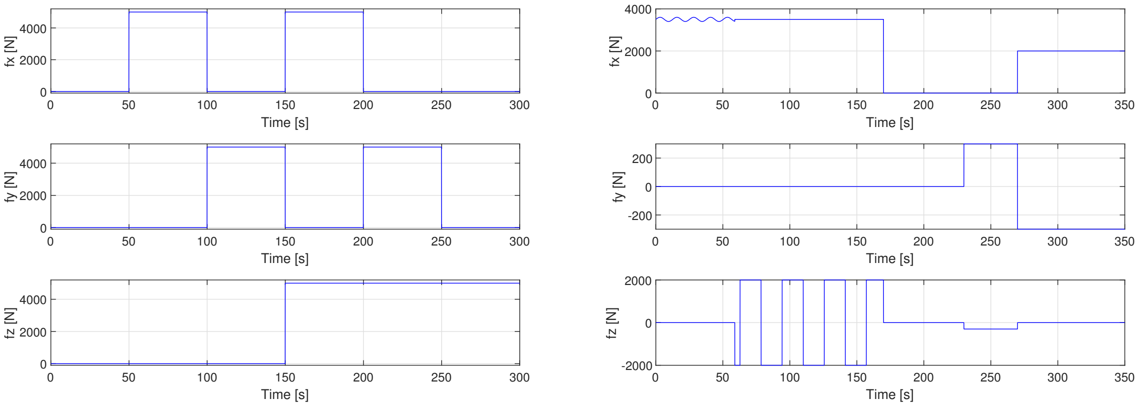
$$M = \begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & m_{23} \\ 0 & m_{23} & m_{33} \end{bmatrix}, \quad D = \begin{bmatrix} X_u - X_{|u|u}|u_b| & 0 & 0 \\ 0 & -Y_u - Y_{|u|u}|v_b| & -Y_r \\ 0 & -N_r & -N_r - N_{|r|r}|r| \end{bmatrix}$$

The Coriolis and centripetal matrix **C** is then derived from **M**:

$$C = \begin{bmatrix} 0 & 0 & -m_{22}v_b - m_{23}r_b \\ 0 & 0 & m_{11}u_b \\ m_{22}v_b + m_{23}r_b & -m_{11}u_b & 0 \end{bmatrix}$$

A few data sets with different inputs to the model and the resulting outputs was created. It's necessary to train and test on different data to test how the models produced by the algorithm performs on unseen data, how well it generalizes.

Two of these inputs are shown in Figure 3.7. A model was trained on each of these, and tested on the other. The algorithm finds models that fit the data that they were trained on with a high accuracy. But when checking how the model generalizes to the other set of inputs, model 2 does not fit the validation data well, while model 1 does, as seen in Figures 3.8 and 3.9.



(a) Input 1 for data set 1

(b) Input 2 for data set 2

Figure 3.7: The two sets of inputs to the system that were used when creating data set 1 and 2, are shown in this figure.

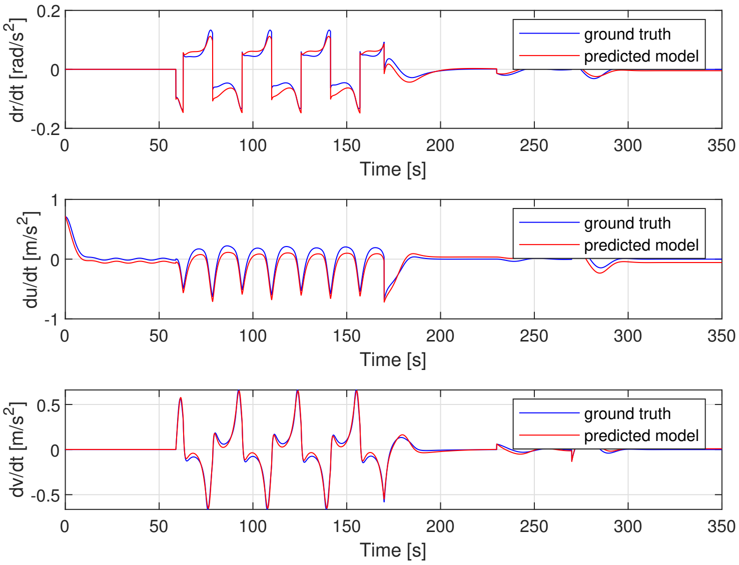


Figure 3.8: Model 1, trained on data set 1, validated on data set 2

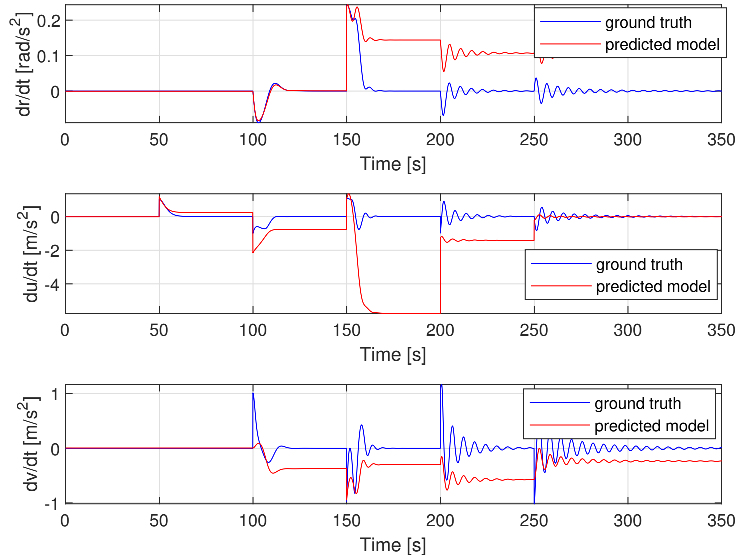


Figure 3.9: Model 2, trained on data set 2, validated on data set 1

To check the what models the algorithm should find, the actual model can be found from eq. (3.2 and 3.3) on component form by solving for $v^b_{b/n}$:

$$\dot{u} = F_u(u_b, v_b, r_b) + \tau_u \tag{3.4}$$

$$\dot{v} = X_v(u_b)r_b + Y_v(u_b)v_b \tag{3.5}$$

$$\dot{r} = F_r(u_b, v_b, r_b) + \tau_r \tag{3.6}$$

Where

$$F_u(u_b, v_b, r_b) = \frac{1}{m_{11}}(m_{22}v_b + m_{23}r_b)r_b - \frac{d_{11}|u_r|}{m_{11}}u_b \tag{3.7}$$

$$F_r(u_b, v_b, r_b) = \frac{m_{23}d_{22}|v_r| + m_{22}(d_{32} + (m_{22} - m_{11})u_b)}{m_{22}m_{33} - m_{23}^2}v_b + \tag{3.8}$$

$$\frac{m_{23}(d_{23} - m_{11}u_b) - m_{22}(d_{33}|r| + m_{23}u_b)}{m_{22}m_{33} - m_{23}^2}r_b \tag{3.9}$$

$$X_v(u_b) = \frac{m_{23}^2 - m_{11}m_{33}}{m_{22}m_{33} - m_{23}^2}u_b + \frac{d_{33}|r|m_{23} - d_{23}m_{33}}{m_{22}m_{33} - m_{23}^2} \tag{3.10}$$

$$Y_v(u_b) = \frac{(m_{22} - m_{11})m_{23}}{m_{22}m_{33} - m_{23}^2}u_b - \frac{d_{22}|v_r|m_{33} - d_{32}m_{23}}{m_{22}m_{33} - m_{23}^2} \tag{3.11}$$

The mass and damper matrix, **M** and **D**, from the simulator are the following:

$$M = \begin{bmatrix} 4935 & 0 & 0 \\ 0 & 4935 & 0 \\ 0 & 0 & 20928 \end{bmatrix}, \quad D = \begin{bmatrix} 50 + 243|u_r| & 0 & 0 \\ 0 & 200 + 2000|u_r| & 0 \\ 0 & 1281 & 1281 + 2975|r| \end{bmatrix} \tag{3.12}$$

Inserting the constants in eq. (3.12) in eq. (3.7- 3.11), results in the following equations:

$$F_u(u_b, v_b, r_b) = v_b r_b - 0.01 - 0.05|u_b|u_b \tag{3.13}$$

$$F_r(u_b, v_b, r_b) = -0.26v_b - v_b r_b - 0.6|r_b|r_b \tag{3.14}$$

$$X_v(u_b) = -4.24u_b \tag{3.15}$$

$$Y_v = -0.17 - 1.72|u_b| \tag{3.16}$$

The resulting equations that the algorithm seeks to find is therefore:

$$\begin{aligned} \dot{u} &= v_b r_b - 0.01 - 0.05|u_b|u_b + \tau_u \\ \dot{v} &= -4.24u_b r_b - 0.17 - 1.72|u_b|v_b \\ \dot{r} &= -0.26v_b - v_b r_b - 0.6|r_b|r_b + \tau_r \end{aligned} \tag{3.17}$$

Model 1:

$$\begin{aligned} \dot{u} &= 0.967422 * v * r + 0.000197 * f_x - 0.129406 * u - 0.021278 * u * u + 0.027803 \\ \dot{v} &= -0.993242 * u * r + 0.000203 * f_y - 0.165217 * v * v * v - 0.251991 * v + 0.001364 \\ \dot{r} &= -0.057697 * v - 0.139841 * r + 0.000051 * f_z - 0.085564 * r * r * r + 0.000170 \end{aligned} \tag{3.18}$$

Model 2:

$$\dot{u} = -0.000913 * f_x * r * r - 0.000240 * (f_y - f_x) - 0.213298 * u - 0.019426$$
$$\dot{v} = -0.332013 * u * r + 0.309584 * r - 0.000116 * f_z + 0.004254$$
$$\dot{r} = 0.000047 * f_z - 0.060488 * v - 0.114454 * r - 0.000348 \tag{3.19}$$

One of the restrictions of the toolbox Abonyi (2014), that could be solved by using another library like *GPlearn* for python, is that it only uses the simple operators $[+, -, *]$. From eq. (3.17) it is seen that the model contains the absolute value operator which the models from the algorithm can not.

In model 1 there are more terms than in the original model. This is probably to compensate for the signed squared terms. Apart from that, this model contains lot of the components of the of the actual model.

From these results it becomes apparent that the maneuvers of the USV, that the data is comprised of, is one of the most important aspects to consider when using the GP algorithm to find a model. This is to be expected, as the performance of most machine learning algorithms is highly dependent on the quality of the data set. More interesting is to check what maneuvers that describe the system well enough to be able to find an accurate model. The maneuvers described in Section 2.3, was tested on the simulator, and is shown in Figures 3.10 and 3.11.

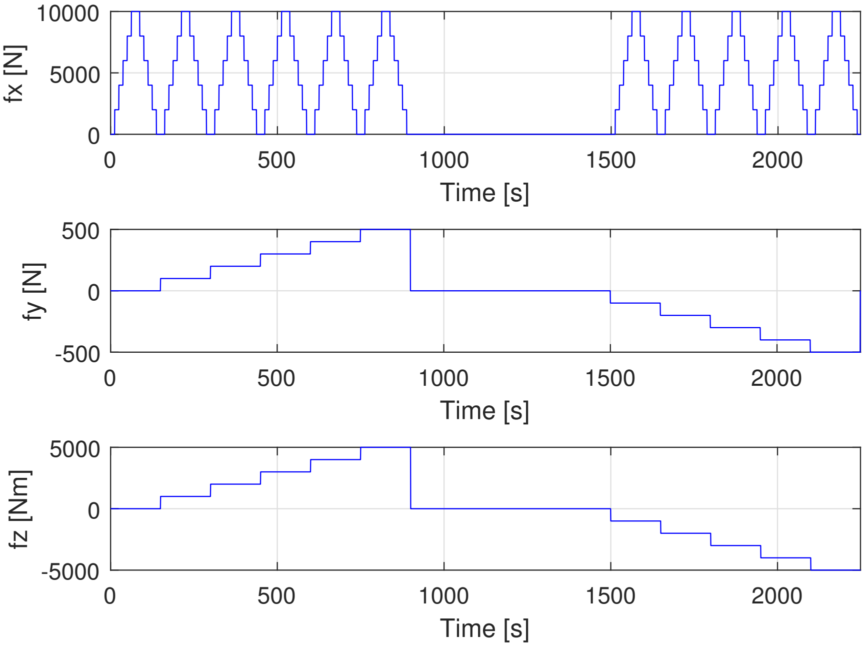


Figure 3.10: Input force and moment to make a data set resembling Eriksen and Breivik (2017)

When testing the performance of the resulting system models on another data set, shown in Figures 3.12 and 3.13, the zig-zag maneuver fails to capture any of the dynamics sufficiently. The model trained with a series of step responses as input does a little better, but not as well as model 1, described in eq. 3.18.

These data sets were not created exactly as described in the original reports, and this might be a reason why the results are this poor. The input to the the system was force, and not rudder and SOG command.

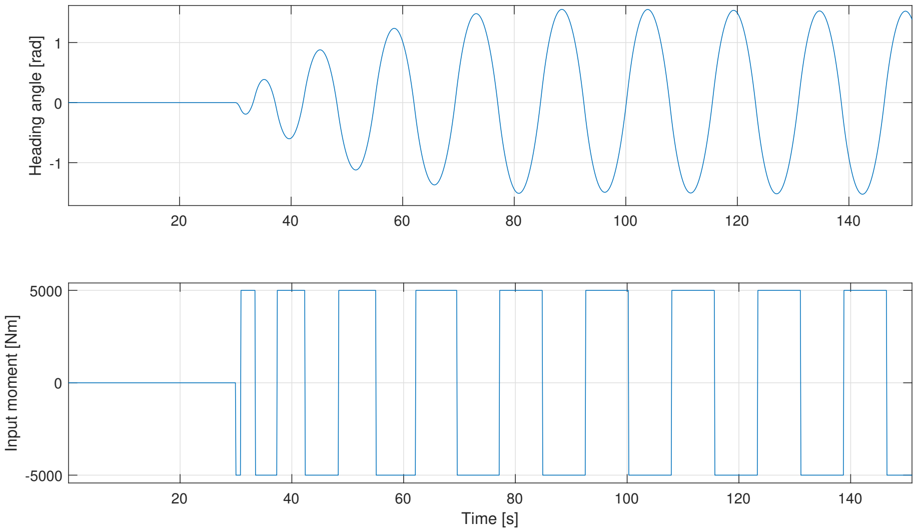


Figure 3.11: Heading angle and input moment in the zig-zag maneuver

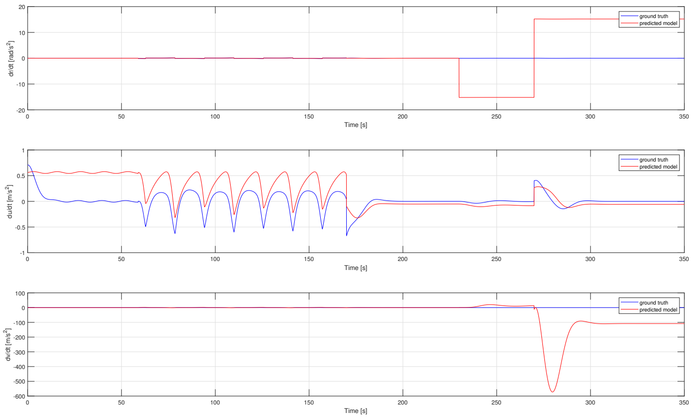


Figure 3.12: Result of the model trained on the zig-zag data seen in Figure 3.11, testes on the data shown in Figure 3.7b

3.3.1 Delay

The same simple scheme for identifying the time delay as in Section 3.2.2 was tested to see if it also works for a more complex system.

As the data comprised of an input series of steps gave the best model, model 1, this is the data that will be used to test with delay. The time delay is implemented as previously described, in Section 3.2.2. As this system has 3 inputs, the search space is 3 dimensional. The results are shown in the 3d surface plot in Figure 3.14. This was computationally expensive, so the plot contains a grid search only in the f_y, f_z-space. The result shows no clear trend as in Figure 3.6 for the MSD-system. It might not have a clear trend because the algorithm finds different models each time and is not actually able to find the exact model, as in the MSD-system.

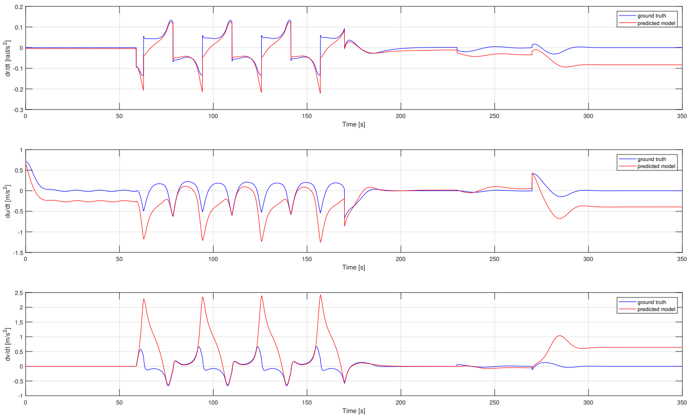


Figure 3.13: Result of the model trained on the grid data seen in Figure 3.10, testes on the data shown in Figure 3.7b

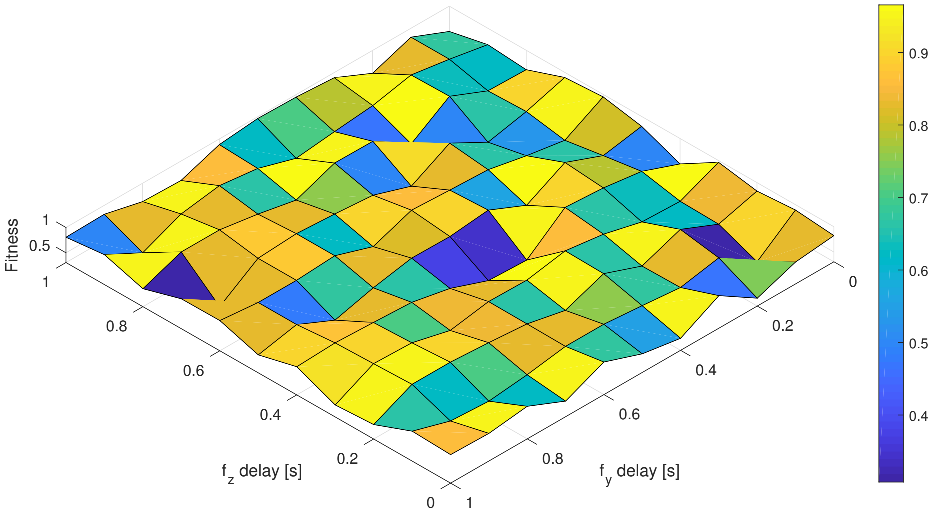


Figure 3.14: Plot showing a grid search with different delay compensations in f_y, f_z-space. Delay = 0.5s

Chapter 4

System identification on real data

4.1 Model

There are different approaches to finding the best model of a ships dynamics when choosing the model-free Genetic programming approach. One of the choices is to model the ship with 2 DOF as in Eriksen and Breivik (2017) or with the 3 DOF normally used. Another choice is whether to use one model for each region the ship is in. The ideal model would be able to model the the dynamics of the ship in all 3 states.

2 DOF model:

$$Kinetics:$$

$$\dot{U} = f_U(U(t), r(t), \delta_r(t), \delta_t(t)) \tag{4.1}$$

$$\dot{r}_\chi = f_{r_\chi}(U(t), r(t), \delta_r(t), \delta_t(t)) \tag{4.2}$$

$$Kinematics:$$

$$\dot{\eta} = \begin{bmatrix} cos(\chi) & 0 \\ sin(\chi) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} U \\ r \end{bmatrix} \tag{4.3}$$

$$\dot{\chi} = r_\chi \tag{4.4}$$

3 DOF model:

$$Kinetics:$$

$$\dot{u} = f_u(u(t), v(t), r(t), \delta_r(t), \delta_t(t)) \tag{4.5}$$

$$\dot{v} = f_v(u(t), v(t), r(t), \delta_r(t), \delta_t(t)) \tag{4.6}$$

$$\dot{r} = f_r(u(t), v(t), r(t), \delta_r(t), \delta_t(t)) \tag{4.7}$$

$$Kinematics:$$

$$\dot{\eta} = R(\psi)\nu \tag{4.8}$$

Where $U(t)$ is the SOG, $r(t)$ is the yaw rate, $\delta_r(t)$ is the rudder angle and $\delta_t(t)$ is throttle from the water jets.

When designing the models in this way the actuator dynamics are implicitly modeled, and no control allocation is needed.

4.2 Data

4.2.1 Zig-zag

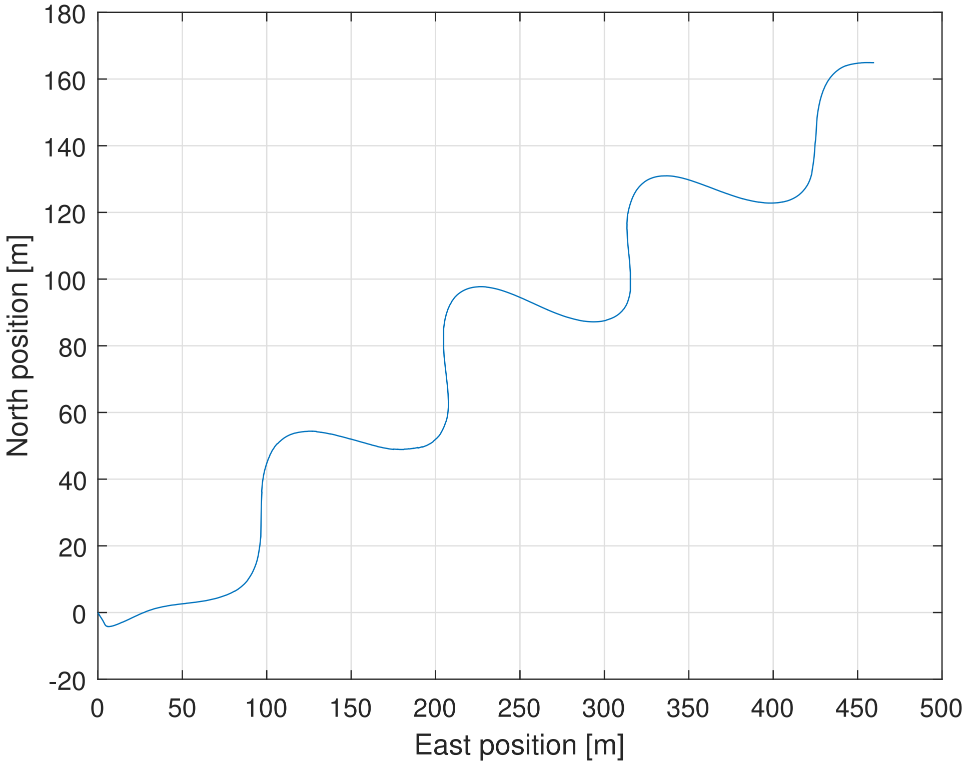


Figure 4.1: Zig-zag

The data used for training is captured by the on-board IMU and GPS. There exists more data than the zig-zag data described below, but this was not captured with the intention of model identification. Only the zig-zag data was used for training, as the other data gave poor to no result.

The data that will be used for training was recorded fall 2018 outside of Horten, Norway. There are 4 zig-zag maneuvers with different throttle inputs. The basis for this choice of maneuvers is described in Section 2.3. From this simple set of maneuvers, the data collected should be sufficient to describe some of the systems dynamics, but only in the displacement phase.

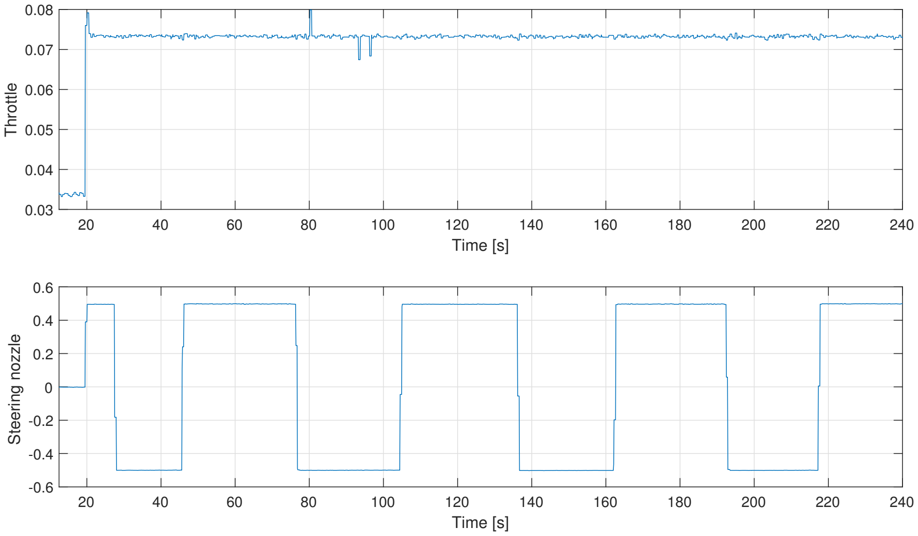


Figure 4.2: Plot showing the jet nozzle angle and the throttle for one of the zig-zag maneuvers. The jet nozzle is in the range [-1,1] where 1 and -1 is the max and min. The throttle is derived from the jets RPM and is in the range [0,1]

A drawback, later discovered, is that the throttle should not be constant but rather a function of commanded SOG. This is to cover more of the actuators impact on the dynamics.

4.2.2 Smothing

The data from the onboard IMU is a bit noisy. To find $\dot{u}$, $\dot{v}$, $\dot{U}$ and $\dot{r}$ with Euler approximation eq. (4.9), a lot of smoothing was needed.

$$\dot{x} = \frac{x_{k+1} - x_k}{t_{k+1} - tk} \tag{4.9}$$

The smoothing is done on the raw data of u, v and r, before the derivatives was found. The different states was smoothed with different step lengths as the but with the same *MATLAB* function *smoothdata* with the *Savitzky-Golay filter* method.

One of the drawback of smoothing the data is that it can create some errors and delays in the data, compared to the real values. This can be viewed in Figure 4.4 for $\dot{u}$, for about 230s. This is something that is not addressed in this report, but is something that need to be dealt with.

Another subject that becomes apparent when looking at Figure 4.4, is that when the noise is removed from $\dot{r}$, there is no signal left that the algorithm really can train on. And if smoothing with a smaller window is used, the signal contains too much noise. A maneuver that results in larger values would be preferred.

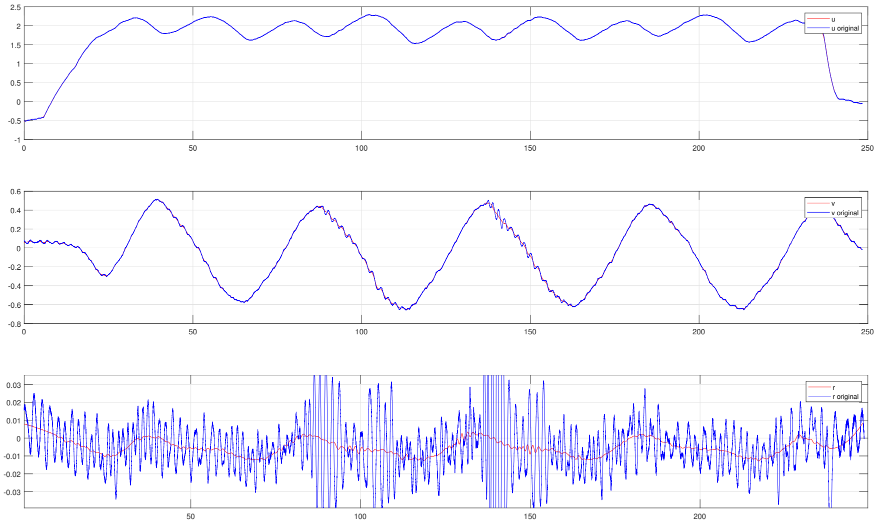


Figure 4.3: Difference with and without smoothing

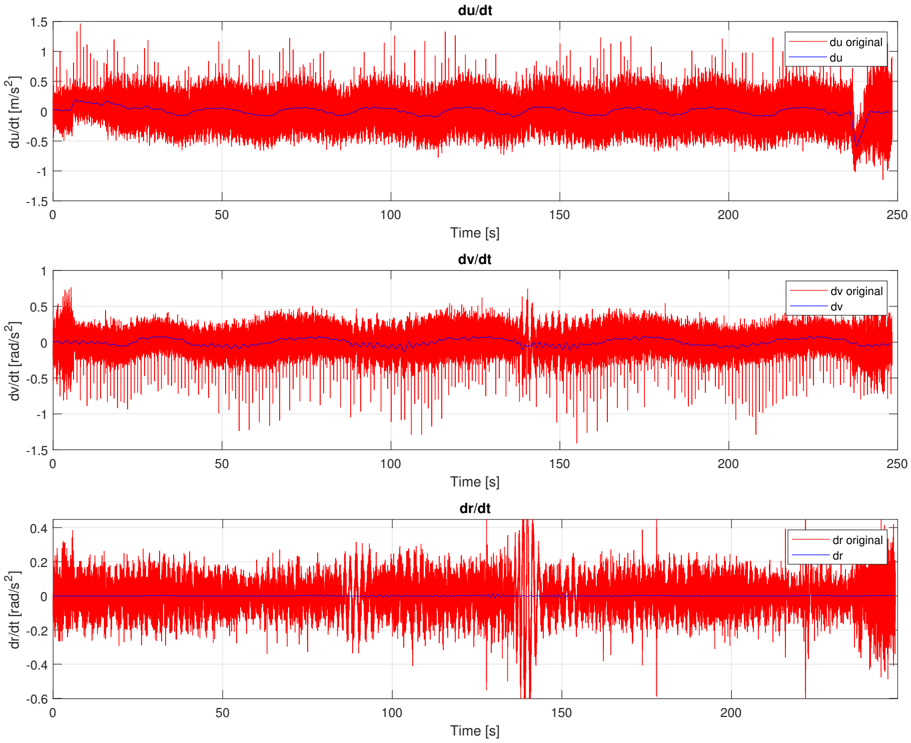


Figure 4.4: The result a smooth u, v and r has on the derivatives

4.2.3 Interpolation

The data from the water jets and the IMU is recorded at different frequencies. To make a data set that the GP algorithm understands the sampling times must be the same. This is solved by using an interpolations method in

MATLAB, *interp1* with the *spline* method.

4.2.4 Scaling

The recorded data is of completely different scales. The shaft RPM ranges from 400 to 2000, while most of the other variables ranges most of the time from -10 to 10. this is problematic for most machine learning algorithm, and contributes to increase the convergence time, and generally deceases the performance. The solution to this was to apply zero mean, and unit variance to the variables individually.

4.2.5 Cross-validation

To test how well the found model generalizes to unseen data, the data is split up in different sets, as done in Section 3.3. This is an important step in the process, as the algorithm is prone to overfitting, especially is the data trained on does not contain all aspects of the dynamics. How well a found model preforms was tested solely on unseen data.

4.3 Result

Data sets were created from the zig-zag maneuvers, and trained with the Genetic programming algorithm. Two of the best preforming model are listed below.

Model 3, trained on zigzag 2:

$$\begin{aligned}
\dot{u} &= -0.036082 * u * u + 1.322479 * \delta_t + 0.460200 * v * (\delta_t - v) + 0.087091 \\
\dot{v} &= -0.000400 * \delta_r . * u - 0.000695 \\
\dot{r} &= 0.125470 * u * \delta_t * (\delta_t + v) * \delta_t - 0.000004 * \delta_r - 0.000093
\end{aligned} \tag{4.10}$$

Model 4, trained on zigzag 4:

$$\begin{aligned}
\dot{u} &= -1.645115 * (r + \delta_t) * u - 0.458437 * v * v + 4.312088 * \delta_t - 0.011866 \\
\dot{v} &= 0.001850 * (v * \delta_r * v) - 0.000609 * \delta_r * u + 0.003096 \\
\dot{r} &= -0.000008 * \delta_r - 0.002161 * v * v + 0.001262 * u * \delta_t * u - 0.000419
\end{aligned} \tag{4.11}$$

These model should have some of the same components as the actual model eq. 3.17, shown in Section 3.3, as this is a model of Odin. The difference in the results may be a result of the actuator dynamics being modeled.

The results from testing the models on each others data set to validate, is shown in figures 4.6 and 4.7. This shows that the models fits quite well to the data from each other. But this is highly biased, as both these data sets are similar. This becomes clear when a model 1 is tested on data from a drive around the marina in Horten, shown in Figure 4.8. From this figure it's apparent that the model does not represent the systems dynamics.

Another result that becomes apparent from the Figure 4.6, is that the model are too simple to fit the real values. For $\dot{v}$ in model 3, this is easy to see from the equation, and is reflected the result in the figure.

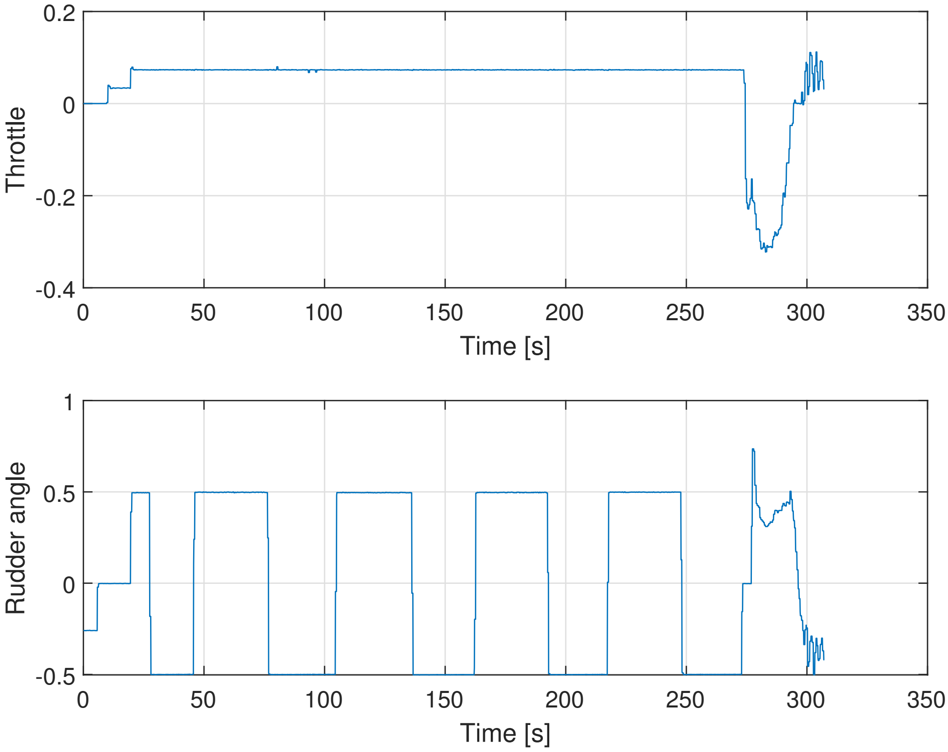


Figure 4.5: Rudder and motor throttle for zigzag 1

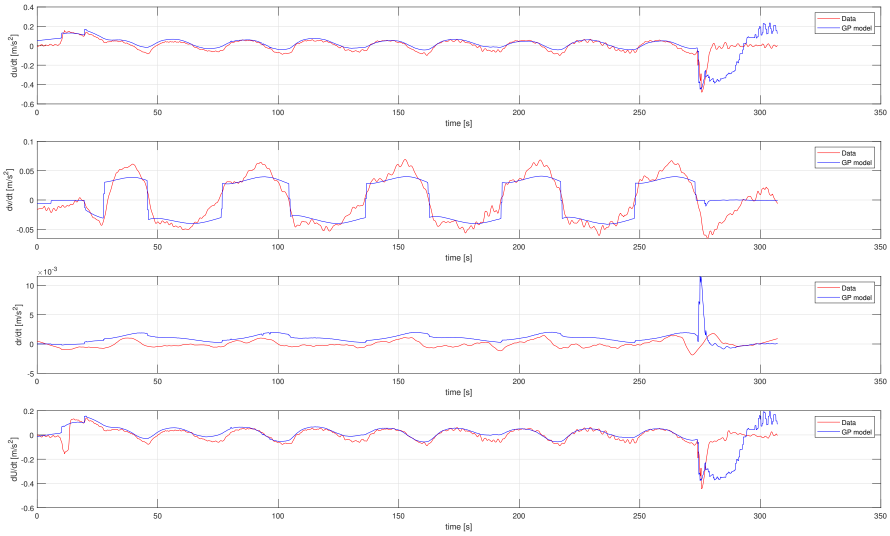


Figure 4.6: Result when the data from zigzag 1 was tested on model 3. $\dot{U}$ can be disregarded.

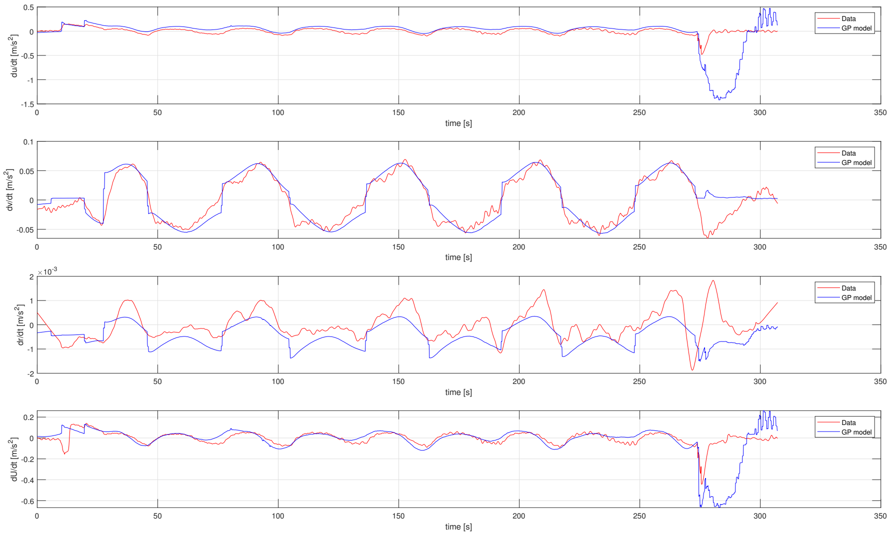


Figure 4.7: Result when the data from zigzag 1 was tested on model 4. $\dot{U}$ can be disregarded.

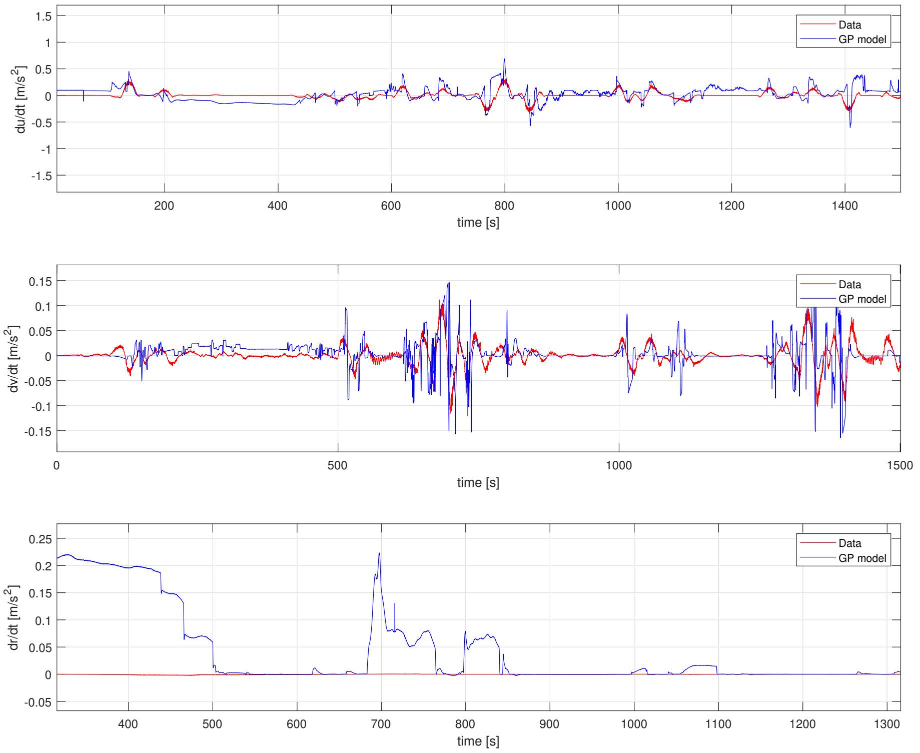


Figure 4.8: Model 3 tested on data from a drive around the marina in Horten.

Chapter 5

Discussion

There may be multiple reasons that the resulting models, trained on the real data, is not sufficiently accurate. The genetic programming algorithm might not be powerful enough to find a good representation of the dynamics. There is a time delay in the system, but how much it contributes to the result is unclear. The chosen toolbox may not be the best one for this task, or perhaps there are better ways of choosing the hyperparameters of the algorithm. The data used for training could be the main contribution to the poor resulting model.

Genetic programming

The main reason that GP with SR was chosen as the way to identify the model, was that it gives a differential equation model, and that the article Moreno-Salinas et al. (2015) shows great results. The technique is fairly easy to implement for own data as the algorithm is implemented in a *MATLAB* toolbox. As the scope of this project is not that comprehensive, this seamed like an appropriate approach.

The algorithm showed great results on the simplest system, the mass-spring-damper. It was able to find the correct model, with only a few iterations, and even with a lot of noise. The results from the data generated by the 3-DOF ship model showed that the algorithm with the simplest operators had some flaws, but were able to find a model that had similar dynamics to the actual model.

The GP algorithm did not find a suitable model on the real data, but as an algorithm for model identification it performed well on the simulation generated data, so the algorithm in it self is probably able to find a suitable model for Odin.

Time delay

As discussed in Chapter 3, time delays in the system is not something that GP has the tools to deal with. This is why some manual brute force implementation that tries different time delays was implemented. For the simple MSD-system, this worked out as planned. This might be because of the simplicity of the system, and might not work on a more complex system. As shown in Figure 3.14, this method did not reveal the introduced time delay.

Another way to implement the time delay estimator would be to assume that the time delay is the same for all states. This is a valid assumption as it is most likely caused by a delay from the onboard computer to the actuators. If this does not work, then another more sophisticated approach, would be a better solution.

The brute force time delay estimator assumes that the GP algorithm finds the correct model when the time delay is compensated for, this may not be the case and that is a major flaw in the approach.

Prepossessing of the data

The data from the ships onboard IMU was quite noisy, at least when the accelerations were found by the Euler approximation. There probably is a better way to find the accelerations, that results in less noise.

Another aspect to consider in the prepossessing may be the smoothing of the data. As shown in Figure 4.4 the smoothing may cause some time delay in the signal. This is an issue that need to be addressed, and solved to create an optimal data set for the algorithm to train on.

Data

The real data used in training of the model comes from a zig-zag maneuver. This maneuver was intended for ships in the displacement phase, not in all three. The resulting models shown in Section.4.3 will therefore probably not fit well when the vessel is in higher speeds, where the aerodynamic forces starts playing a role.

A more suitable gathering of data for model identification would be similar to the method described in Eriksen and Breivik (2017). This is in agreement with the result found in Section 3.3, where the data that covered the dynamics of the vessel best was a series of steps.

Toolbox

Another issue that has been stressed throughout this report is the use on the toolbox Abonyi (2014). The limitations when it comes to the operators used in the resulting models may contribute a lot the poor model accuracy. This became apparent in eq. (3.17 and 3.18), where the model had an acceptable accuracy, but did not generalize to the validation data as well as it could have with another model.

Required accuracy of the model

The model does not have to fit the exact dynamics of the USV, to be able to design a controller that works well. A lot can be compensated for in the choice of controller. As seen in the report by Eriksen and Breivik (2017), the model does not fit the data perfectly but is still able to follow a desired reference. This does not mean that any of the model found in Section 4.3 is sufficiently accurate. But with some of the adjustments mentioned in this chapter, the chosen Genetic programming approach may result in the desired model.

Chapter 6

Conclusions and future work

6.1 Conclusion

In this report it is shown that the genetic programming algorithm is a valid tool for system identification, even on complex systems. But, like other machine learning algorithm, it is sensitive to the quality of the data. The real data captured from Odin, that was used in training was probably one of the reasons for the poor resulting models performance.

Another reason for the poor performance on the real data was the operators used in the resulting model. With more options in operators, the algorithm might perform better on the real data as shown in Section 3.3.

6.2 Future work

To conclude on the model identification capabilities of the GP algorithm on real data, some future line can be drawn:

- The algorithm should be tested on data gathered similarly to the presented method in Eriksen and Breivik (2017).
- Implementations of the simple time delay estimations should be done for the real data.
- Test the performance when more operators are available to the GP algorithm.

Appendix A

My Appendix

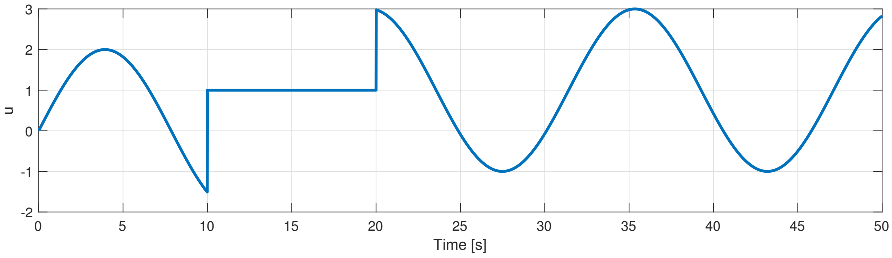


Figure A.1: Input forced on the mass spring damper system

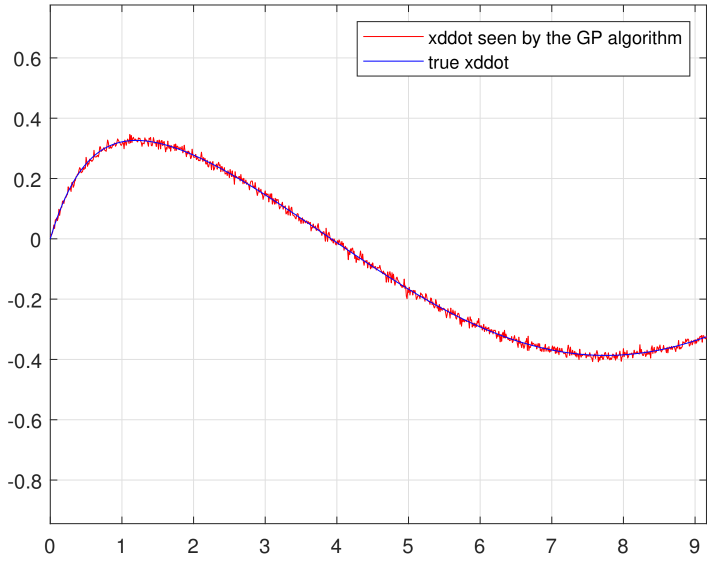


Figure A.2: White noise with amplitude = 0.01

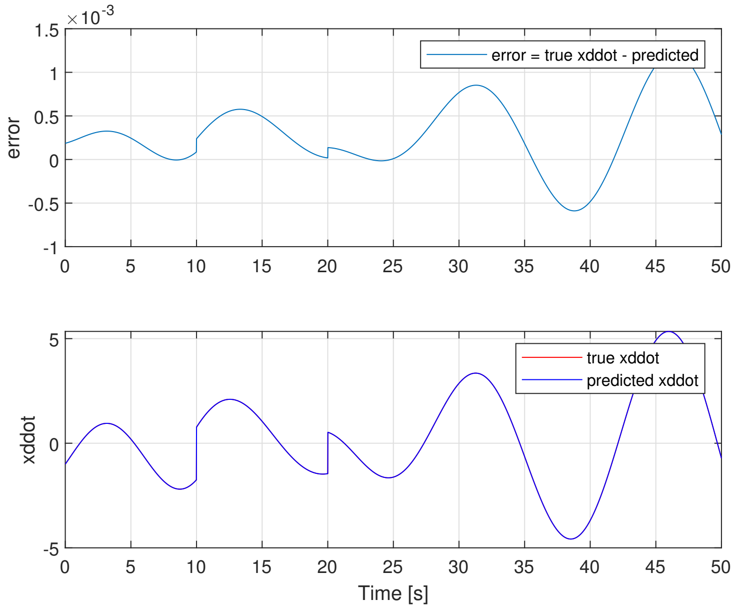


Figure A.3: Prediction with noise amplitude = 0.01

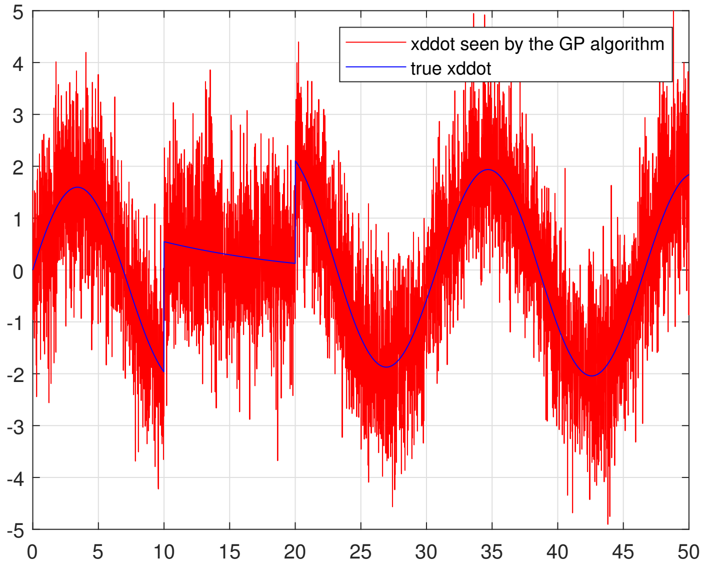


Figure A.4: Noise with amplitude on the same scale as the states

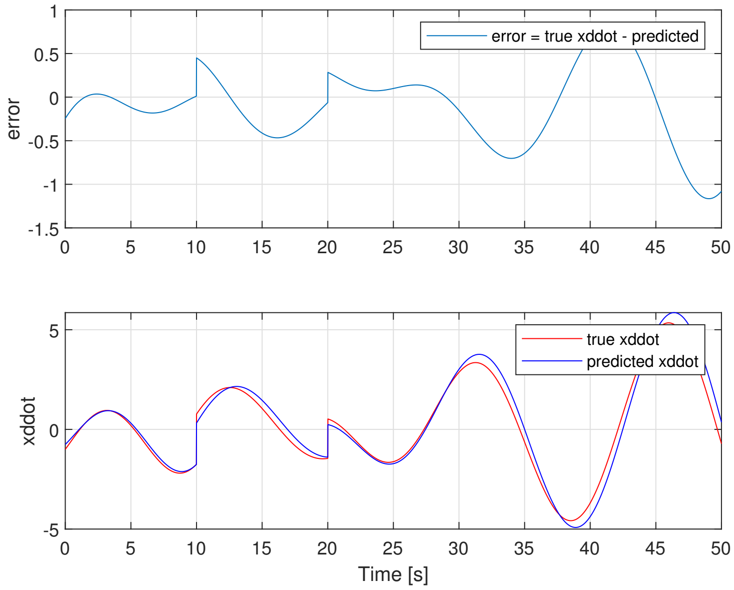


Figure A.5: Resulting model with the noise amplitude on the same scale as the states

References

Abonyi, J. (2014). Genetic Programming MATLAB Toolbox.
URL: *https://se.mathworks.com/matlabcentral/fileexchange/47197-genetic-programming-matlab-toolbox*

Eiben, A. E. and Smith, J. E. (2012). *Introduction to Evolutionary Computing Genetic Algorithms.*

Eriksen, B.-o. H. and Breivik, M. (2017). Modeling, Identification and Control of High-Speed ASVs: Theory and Experiments, **474**(June).
URL: *http://link.springer.com/10.1007/978-3-319-55372-6*

Faltinsen, O. m. (2005). *Hydrodynamics of High-Speed Marine Vehicles.*

Fossen, T. I. (2002). *Marine Control Systems: Guidance Navigation, and Control of Ships, Rigs and Underwater Vheicles.*

Fossen, T. I. (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control.*

Madar, J., Abonyi, J. and Szeifert, F. (2002). Genetic Programming for System Identification, *Technical report.*

Moreno-Salinas, D., Besada-Portas, E., López-Orozco, J. A., Chaos, D., De La Cruz, J. M. and Aranda, J. (2015). Symbolic regression for marine vehicles identification, *IFAC-PapersOnLine* **28**(16): 210–216.
URL: *http://dx.doi.org/10.1016/j.ifacol.2015.10.282*

Rajesh, G. and Bhattacharyya, S. K. (2008). System identification for nonlinear maneuvering of large tankers using artificial neural network, *Applied Ocean Research* **30**(4): 256–263.
URL: *http://dx.doi.org/10.1016/j.apor.2008.10.003*