Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Gisle Finstad Halvorsen

Utilizing Genetic Programming for System Identification of a High-Speed USV

Master's thesis in Cybernetics and Robotics
Supervisor: Kristin Y. Pettersen
Co-supervisors: Else-Line Ruud and Martin Syre Wiig
June 2019

NTNU
Norwegian University of
Science and Technology

Gisle Finstad Halvorsen

Utilizing Genetic Programming for System Identification of a High-Speed USV

Master's thesis in Cybernetics and Robotics
Supervisor: Kristin Y. Pettersen
Co-supervisors: Else-Line Ruud and Martin Syre Wiig
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Problem Description

Use of Genetic Programming for Modeling of a High-Speed USV

Identifying the dynamic model of a marine vehicle can be a time consuming and expensive task, involving the use of extensive sea trials, model trials in water tanks and hydrodynamic computing tools. The modelling task is especially challenging for high-speed vehicles, where the vehicle can operate in both the displacement and the planing phase. In the latter regime, the aerodynamic parameters of the vehicle become more dominant, and an analytical closed-loop expression for the vehicle dynamics can be challenging or even impossible to find using conventional methods. Finally, for a vehicle such as FFI's Odin USV, the vehicle configuration can change significantly from mission to mission, depending on the vehicle payload. Thus, the vehicle model can vary, and it is of interest to find a low-cost method to identify the vehicle dynamics.

The goal of this project assignment is to investigate the use of genetic programming in order to create a vehicle model. The student should investigate different approaches to genetic programming, in order to select an approach suitable for modelling of a marine vehicle. The chosen model should be applied both to simulated and real vehicle data, and the performance of the model should be compared with the performance of a standard displacement model.

The work in this assignment builds upon a preliminary work performed in the autumn of 2018.

Abstract

When designing a vehicle control system, a sufficiently accurate model is important for both control strategy, and simulations. For high-speed marine vessels, modelling is not a trivial task. The dynamics of the USV changes in the different phases, going from the displacement phase where the buoyancy forces dominate the hydrodynamics, to the planing phase where the aerodynamic lift and drag forces starts playing a role.

In this thesis, an approach to this modelling problem is presented, utilizing the data-driven method genetic programming. Different tools to implement the algorithm in code are examined, on a simple system. The algorithm's ability to find the correct solution with noisy data is investigated and shows promising results. Genetic programming was tested on a system with similar complexity to the real system to easier analyze the performance and weaknesses of the algorithm.

The method is used on data captured by the vehicle's sensors, in order to model the vehicle. Two vehicle models are derived, where one considers the actuator force as the input to the system, thus excluding the actuator dynamics from the model. The other model considers the control signals, sent from the autopilot to the actuators as the input to the system. Hence, the second model includes the actuator dynamics in the model. The models show decent goodness of fit on the validation data, and can be considered accurate enough for simulation and control purposes.

Sammendrag

Ved utforming av et fartøys kontrollsystem er en tilstrekkelig nøyaktig modell viktig for både kontrollstrategi og simuleringer. For marine høyhastighetsfartøyer er ikke modellering en triviell oppgave. Dynamikken til en USV endrer seg i de forskjellige fasene, fra forskyvningsfasen der oppdriftskreftene dominerer hydrodynamikken, til planingsfasen hvor den aerodynamiske løftekraften og luftmotstanden begynner å ha en innvirkning på systemet.

I denne avhandlingen presenteres en tilnærming til dette modelleringsproblemet ved å benytte den datadrevne metoden genetisk programmering. Ulike verktøy for å implementere algoritmen i kode er undersøkt på et enkelt system. Algoritmenes evne til å finne den riktige løsningen med støyete data blir også undersøkt, og viser lovende resultater. Genetisk programmering ble testet på et system med lignende kompleksitet til det virkelige systemet for å lettere analysere algoritmens styrker og svakheter.

Metoden brukes på data tatt opp av båtens sensorer, for å modellere fartøyet. To fartøysmodeller er utledet, hvor man i den ene betrakter aktuatorkraften som input til systemet, og utelukker dermed aktuatordynamikken fra modellen. Den andre modellen vurderer styresignalene, som sendes fra autopiloten til aktuatorene som input til systemet. Dermed inneholder den andre modellen aktuatordynamikken i modellen. Modellene passer bra til valideringsdataene, og kan dermed betraktes som nøyaktige nok for simulering og kontroll formål.

Preface

This thesis is written as a compulsory part of the Master's degree in Engineering Cybernetics at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). This thesis has been written in cooperation with FFI.

Preliminary work for this thesis was performed during the specialization project in the fall of 2018 [14]. The objective of the project was system identification of the USV Odin. For this project, the genetic programming toolbox for MATLAB was used. Several factors unveiled in this work made this tool unfit for this problem. This is further explained in Section 1.2.

The main part of this thesis uses the DEAP library for evolutionary programming, which can be found at `https://github.com/deap/deap`. As a part of the work on this thesis, several modifications were implemented. These are explained in detail in Chapter 5.

Along with great counselling, the 3-DOF model used in Chapter 6, the transformation from control inputs to force and moment used in Chapter 6 and 7, and the data used in Chapter 7 was provided by FFI.

I would like to thank my supervisors, Else-Line Ruud and Martin Syre Wiig. Their motivational support and helpful insights throughout this work have been of tremendous help to me!

Contents

List of Tables

List of Figures

List of Acronyms

GP	Genetic Programming
USV	Unmanned Surface Vehicle
ASV	Autonomous Surface Vehicle
LIP	Linear In Parameters
LS	Ordinary least squares
OLS	Orthogonal least squares
GPS	Global Positioning System
IMU	Internal Measurement Unit
FFI	Norwegian Defence Research Establishment/ Forsvarets Forskningsinstitutt
DOF	Degree Of Freedom
SOG	Speed Over Ground
ANN	Artificial Neural Networks
WLS	Weighted Least Squares
MSE	Mean Squared Error
MAE	Mean Absolute Error
MSD	Mass-Spring-Damper
NED	North East Down
GT	Ground Truth
RPM	Rounds Per Minute
LP	Low-Pass
ROS	Robot Operating System

Chapter 1

Introduction

1.1 Motivation

"*The availability of a sufficiently accurate USV model enabling effective control design is imperative for both control methodology design and simulation study purposes*" - Z. Liu et. al 2015 [20]

In the past two decades, autonomous vehicles, especially autonomous cars, have gained increasing attention. This is also the case for marine vehicles such as Unmanned Surface Vehicles (USV). The increased popularity for USVs can be attributed to the reduced cost of navigation equipment such as Global Positioning System (GPS) and Inertial Measurement Unit (IMU), but also that these units have become more compact and efficient [20]. USVs have a wide range of applications, from scientific and commercial to military uses. Benefits include a widened weather window of operations, reduced cost and increased personnel safety [6][20].

A sufficiently accurate dynamic model of a system is key to the development of the controller strategy and simulations. For advanced non-linear control strategies, attempting to design feedback control to cope with model uncertainty might lead to either robustness or adaptive control problems [17]. Model uncertainty also makes simulations less valuable, as the results are further from how the actual system behaves. Simulations are also useful when developing other systems on a USV than the control strategy, like collision avoidance and autonomous docking.

Marine vehicles that operate in the displacement phase is usually modelled according to T. Fossen's vectorial model [9]. This model is only valid for low velocities relative to the size of the vehicle. For so-called high-speed marine vehicles, there is no conventional way of modelling the dynamics. The literature on the subject usually tend towards data-driven approaches [6][2], but the list of publications is relatively short. The objective of this thesis is to model the dynamics the high-speed vehicle Odin.

1.2 Background

In the specialization project [14], a pre-project for this thesis, the proposed method of this report, using Genetic programming (GP) for system identification of a high-speed marine vehicle, showed promising results. The study used the GP toolbox for MATLAB [1], and points out several flaws in the library and the data used.

The report points to the preprocessing of the data, as it likely removed some of the dynamics of the USV and introduced time delay in the data. Further, the library's customization capabilities were shown to be limited. This includes the functions in the function set, and the difficulty in adding additional features. Finally, the data used in the project was insufficient. The data contained few manoeuvres, and all at low surge velocities.

The 3-DOF marine vehicle model, the transformation from control input to force used in Chapter 6 and 7, and data used in this thesis was provided by FFI.

1.3 Contributions

The main contributions of this thesis are:

- A thorough explanation of GP as a method for system identification of complex systems.
- To some extent, examination of the impact the different hyperparameters has on the performance of GP.
- Presenting the impact noise and time delay has on the performance of the GP algorithm.
- Verification that GP has the ability to find complex models based on a simulated system.
- Using GP to model the complex dynamics of the high-speed USV Odin, resulting in an interpretable model.

The code used for this thesis can be found at a Github repository [15].

1.4 Odin

The USV Odin is the system this thesis aims to model. This craft is a Rigid Buoyant Boat that the Norwegian Defence Research Establishment (Forsvarets Forskningsinstitutt, FFI) is currently working on. The main purpose of the craft is to function as a test platform for future mine countermeasures operations based on the use of unmanned systems.

Odin's propulsion system consists of two Hamilton jets which are located in the stern 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.

Figure 1.1: Odin

The position of the astern deflector is in the range $[-100, 100]$, where 100 is fully open, and -100 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.

1.5 Outline of the Thesis

The thesis begins with a brief history of GP and genetic computation, progresses in GP and other approaches to the problem in the literature review in Chapter 2. GP is further explained in the following chapter, Chapter 3, with an explanation of the evolutionary strategies. Then, different GP libraries and a toolbox are tested on the same system, to find out which is most suited for this thesis, in Chapter 4. Extensions to the DEAP library and the robustness of the algorithm towards noise and time delay is examined in Chapter 5. This concludes the first part of the thesis.

The next part of the thesis begins with Chapter 6. In this Chapter GP's ability to identify the model of a complex system is examined. Finally, real data is applied to the GP algorithm to find the model of Odin in Chapter 7. Here, models with different inputs to the system are found and compared to the standard model by T. Fossen [9].

The last part concludes the thesis with a discussion, conclusion and future work in Chapter 8 and 9.

Chapter 2

Literature Review

2.1 Genetic Programming

The use of evolutionary principles for problem solving originated in the 1950s [4]. One of the first descriptions of the use of an evolutionary process for computer problem solving appeared in the article by R. M. Friedberg in 1958 [10], and evolutionary computing developed from there on. Genetic programming is a part of Evolutionary algorithms which is a subgroup of evolutionary computing.

Some of the first results using Genetic Programming (GP) or BEAGLE (Biological Evolutionary Algorithm Generating Logical Expressions) was published by Richard Forsyth, in 1981 [7]. The approach was tested on a classification problem on, amongst other things, hospital admission data. The purpose was to classify if the person lived or died, and the approach showed decent results.

However, it was not until J. Koza patented and published his first book that GP flourished [18]. This book can be considered a seminal work in GP, as most of the articles presented in this section are referring to this book. Most of the articles on GP that are of interest for this thesis, by proposing new methods and not just applying it to new problems, are from about 1990 - 2005.

The article from 2004 by J. Madar et al. introduces the use of GP to create a linear in parameters model represented as a tree structure [22] . They applied Orthogonal Least Squares (OLS) to estimate the contribution of the branches of the tree. This is the first article, revealed in the literature review, that proposes to use Ordinary Least Squares (abbreviated to LS, as not to be confused with Orthogonal Least Squares) to estimate the parameters of the model.

Although the equation for LS listed in the article by J. Madar et al. [22], and another article by A. Gandomi et al. referring to it with the same equation did not work [12], the method shows great results. The LS equation used in this thesis is from J. Wiley's Classical Linear Regression [28], and shown in (2.1).

$$p = (F^T F)^{-1} F^T y \tag{2.1}$$

Where p is the optimal parameters, $F \epsilon \mathbb{R}^{nxm}$ with m functions values for each n data point and y is the ground truth. This is further elaborated in Chapter 5.

The most relevant article for this thesis is published by D. moreno-Salinas et al. [24]. The article proposes to use GP with symbolic regression to identify the 3-DOF model of marine surface vehicle. The data used in the article were gathered from a model of the original craft. The method shows good results on the data, but migth be a bit biased as the model were only tested on the same manoeuvre as it was trained on. The model found for the craft is valid in the displacement phase.

2.1.1 Mutation and Crossover

Crossover and mutation are the two main ways to evolve a program in GP. Koza's classic book on GP [18], advises to use a mutation probability of 0.

The article by L. Sean and S. Lee investigates the use of mutation and crossover [21]. The probabilities of choosing one, or a combination of the two, were systematically tested to find the optimal choice. The article highlights the complexity/non-linearity of the different choices of these hyperparameters, and concludes that crossover generally is more successful. However, for a small population, mutation can out-perform crossover. The results from the article does not cover truly large scale experiments, this is presumably due to computational limits as the article was published in 1997.

2.1.2 Bloat Control

A common challenge when working with GP is that the solutions tend to grow larger with more generations in the training. This phenomenon is often referred to as *survival of the fattest*, and is called bloat [5]. One of the main reasons to choose GP in this thesis, rather than other more powerful data-driven system identification techniques, would be the simpler representation of the model. Hence, bloat has to be dealt with.

There are some different ways to deal with bloat described in the literature. The seemingly most common way would be to apply Parsimony pressure [29], or to limit the max depth of the tree. Parsimony pressure is similar to regularization techniques in numerical optimization, where an additional term is added to the fitness measurement to punish large individuals. This is shown in (2.2), where n is the number of nodes in the tree structure.

$$fit = \sum (y - \hat{y})^2 + \lambda n \tag{2.2}$$

The problem with Parsimony pressure is that requires tuning of the Parsimony coefficient λ to make it viable. R. Poli et al. presents a solution to this issue with a method to find the optimal Parsimony coefficient is presented [25]. The method of setting a max depth requires some tuning as well. However, this is an integer between 5 to 10 for this thesis, thus requiring less tuning than the Parsimony coefficient.

A non-parametric way of combating bloat in GP is the Double Tournament and Proportional Tournament [19]. These are techniques that require no tuning, and show good results combined with a max tree dept limit.

2.2 Other Approaches

When modeling a marine craft, the vast majority of vessels are modeled as described by T.I. Fossen [9][8]. This will consequently be referred to as the standard model. This model is only valid for displacement vessels. How to calculate which phase the vessel is operating in is shown in Appendix D. Odin operates in all three phases, thus, this model cannot be used for higher surge velocities. This model is explained further in Chapter 6.

A model identification approach that works on high speed Autonomous Surface Vehicles (ASV) is presented by B. Eriksen and M. Breivik [6]. This data-driven approach uses empirical data to identify the parameters of the model. In this article, a new 2 Degree Of Freedom (DOF) model is proposed, rather than the conventional 3-DOF. The two vehicle velocities are Speed Over Ground (SOG) and yaw rate. This 2-DOF model is chosen because the ship is under-actuated, and hence *surge*, *sway* and *yaw* can't be controlled independently. This article also proposes to use the control inputs as the inputs to the model, rather than forces. The inputs to the model are hence chosen as throttle and rudder angle.

B. Eriksen and M. Breivik further create their data set by running the vehicle with different steps on input throttle and rudder angle, with steady states between. The measurements of damping can be taken from the steady states, and the inertia data is gathered from the step responses. Then the model parameters is fitted by the Weighted Least Squares (WLS) method.

Another data-driven system identification approach for a large tanker is presented by G. Rajesh and S. Bhattacharyya [26]. The proposed method is based on the standard model [9], but uses an Artificial Neural Network (ANN) to estimate the non-linear hydrodynamic derivatives of the equations. The article shows decent results.

Chapter 3

Genetic Programming

Genetic programming can be viewed as a supervised machine learning algorithm, inspired by natural selection where the fittest individuals survive, and reproduce to create even fitter individuals. This is done over multiple iterations, or generations, to eventually reach the best possible offspring. In GP, the individuals are computer programs, or models, and the fittest individuals are the ones that best fit the empirical data. The Flowcharts in Appendix G shows how the different parts of GP are coupled together.

3.1 Tree Structure and Function Set

In GP the individuals are represented as hierarchically structured trees containing nodes from the function set, also called primitive set, and terminal nodes. Terminal nodes are the leaf nodes and consists of variables and numerical constants. The function set is the set of operators that are chosen for the problem, and could be any function with arity larger or equal to 1.

Arity is the number of arguments that the function takes. A *sum* function usually has an arity of 2, the two inputs that will be summed, while *absolute* and *sine* functions have an arity of 1.

The function set usually consists of the most basic operands $\{+, -, *, /\}$, but also boolean logic operands like AND, OR, NOT can be used. The choice of function set is highly problem-related, and any function can in theory be used. An example of an individual, with the terminology, is shown in Figure 3.1 .

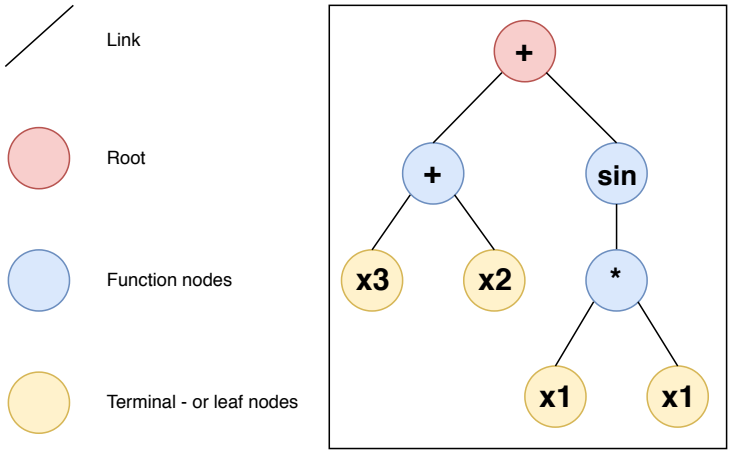


Figure 3.1: An individual, represented in a tree structure.

3.2 Tournament Selection and Fitness Function

In the initial population, the individuals are stochastically chosen. The fitness of an individual is measured with the fitness function to determine how well they fit to the empirical data. Common fitness functions are Mean Squared Error (MSE) and Mean Absolute Error (MAE), but other problem specific fitness functions are also used. The fitness function used in this thesis is further elaborated in Chapter 5. The fittest individuals will be chosen to evolve further, while the others will be discarded.

To set the selection pressure, how many individual's genes that go into the next generation, tournament selection can be used. In tournament selection a series of tournaments with a subset of individuals will be held, where the fittest among the subset is chosen to evolve further. With a large tournament size the selection pressure will be high, the algorithm tends to converge faster, but at the expense of the diversity in the population. With lower diversity in the population, the chance of finding the optimal solution decreases.

3.3 Crossover and mutation

The two ways that individuals evolve through generations are crossover and mutations. Crossover is done by randomly selecting one or two branches on the tree and replacing it with a branch from another tree. Figure 3.2 shows how one point crossover is done.

In mutation, a branch of an individual is mutated in different ways. Figure 3.3 shows three different ways to mutate an individual. A random non-terminal node in the tree-structure is chosen as the mutation point. In *Uniform* mutation, the chosen node and all its branches are replaced by a new randomly chosen branch. In *Node Replacement*, the chosen function node is replaced by another function with the same arity. While in *Shrink* mutation, the chosen node is replaced with one randomly chosen terminal node in this respective branch.

The probability of crossover and mutation is defined as p_c and p_m respectably, and chosen for the specific problem. Crossover is the main way of evolving an individual and it should be the most likely [21] [5]. If the sum of p_c and

p_m is less than 1, $1 - (p_c + p_m)$ is the probability of individuals being directly copied into the next generation, this is called direct reproduction.

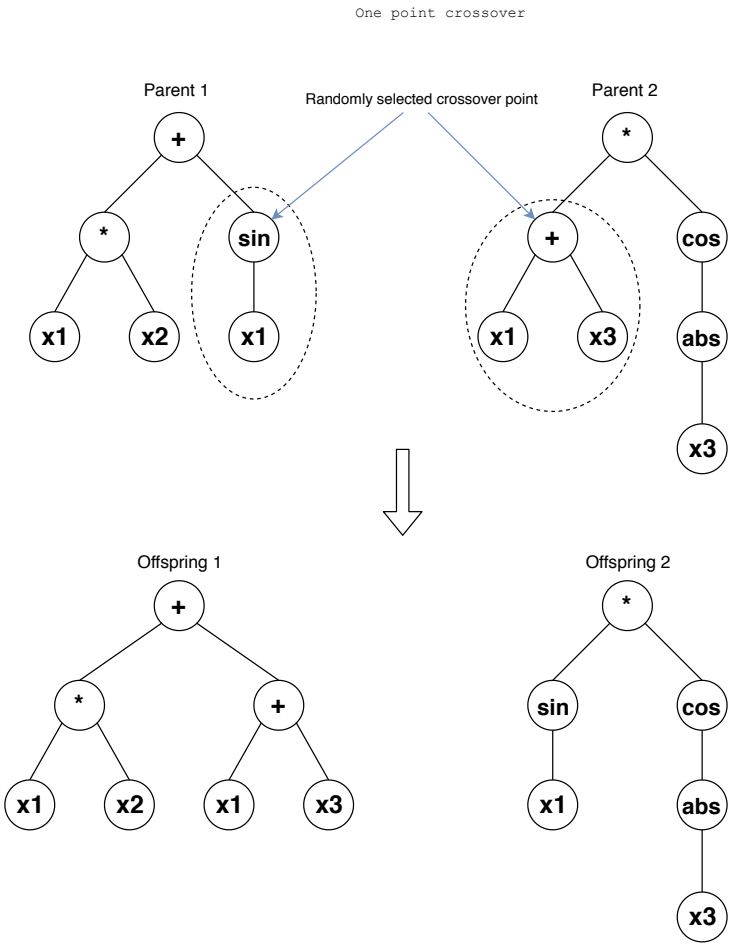


Figure 3.2: One point crossover

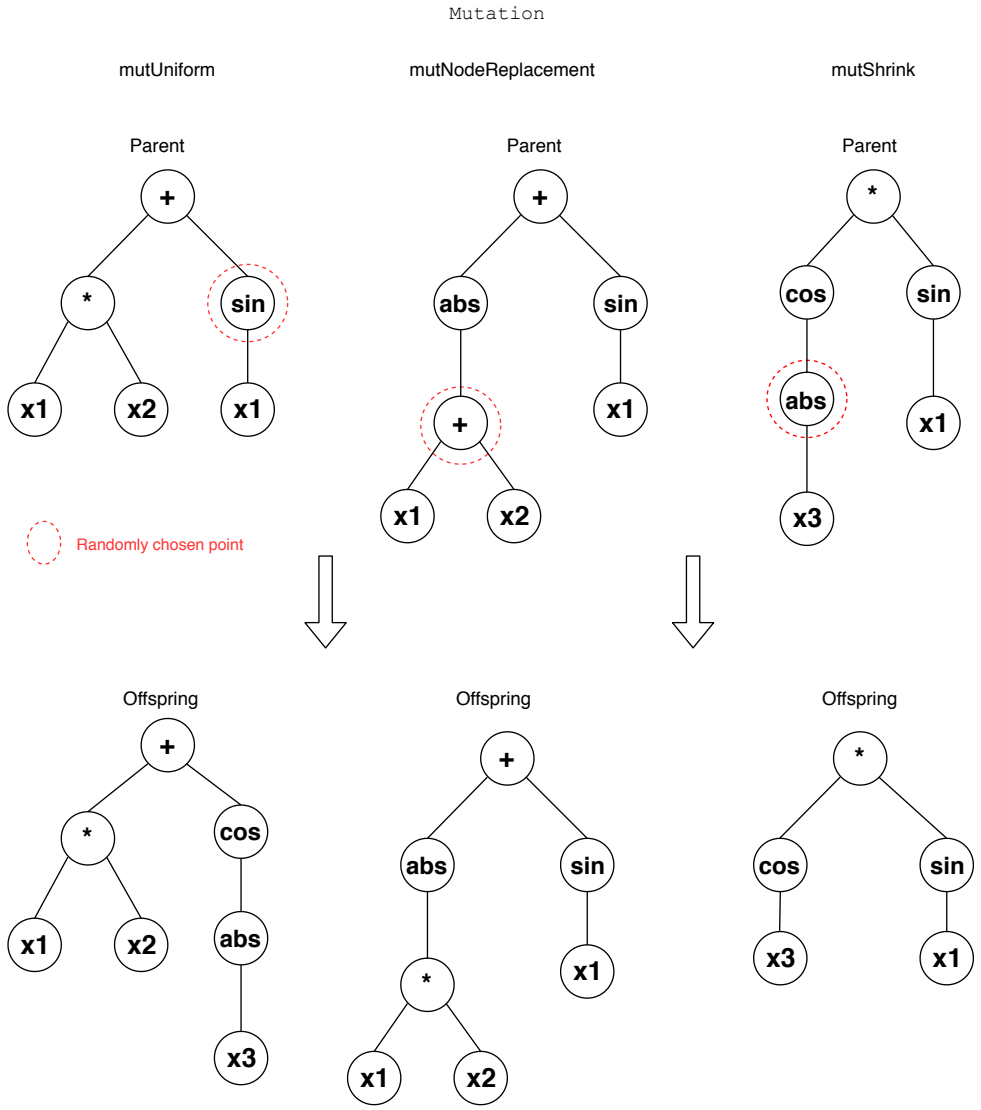


Figure 3.3: Three types on mutation

Chapter 4

Different GP tools

There are several different GP libraries for Python available, and a toolbox for MATLAB. Using these will be advantageous to programming GP from scratch, as it saves time, and would likely prove better preforming than anything created in the scope of this thesis. This section presents three of these, used to identify the mass-spring-damper system, and conclusions to which is preferable for this thesis will be drawn.

4.1 Mass-Spring-Damper System

The mass-spring-damper (MSD) is a second order system. The system consists of a mass connected to a spring, which is connected to a fixed point in space. The mass is sliding across a surface. The damping term is due to the friction, both in the surface and in the spring, and is assumed to be linear with the velocity of the mass.

The system can be modeled as shown in (4.1), where m is the mass, d is the damper coefficient, k is the spring stiffness and τ is the input force acting on the mass. $\ddot{x}$ can be extracted as shown in (4.2). An illustration of the system is shown in Figure 4.1

$$m\ddot{x} + d\dot{x} + kx = \tau \tag{4.1}$$

$$\ddot{x} = \frac{1}{m}(\tau - d\dot{x} - kx) \tag{4.2}$$

From a GP point of view, τ, x and $\dot{x}$ are the inputs and $\ddot{x}$ is the output or ground truth. The inputs to the GP algorithm is usually denoted X and the output y. A simulation of the system was implemented in Matlab and Python and the data from the simulations were used to compare different GP libraries.

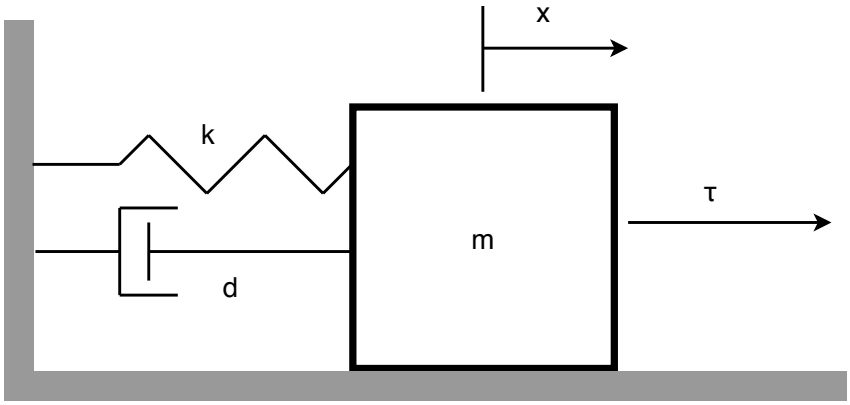


Figure 4.1: Mass-spring-damper system.

4.2 Genetic Programming Matlab Toolbox

The Genetic Programming Toolbox for MATLAB [1], was used in the specialization project leading up to this thesis [14], and showed excellent results on the MSD system. It is, however, not straight forward to implement new features, and makes it difficult to experiment with additional features not already implemented.

This library does not allow more than the most basic primitive set operators $\{+, -, *, /\}$. This is a clear disadvantage as most systems would require more complex functions like, trigonometric functions, absolute value and more. If only the absolute value was in question, this could be worked around with adding the absolute values of the variables as new variables.

4.3 GPlearn

GPLearn is a genetic programming library for python [27]. This library was created to solve solving symbolic regression problems. It has a simplistic user interface which makes it easy and fast to use, resembling the commonly used *scikit-learn* library. It also allows the user to create own functions which is useful for some systems that contains unusual operators.

The drawback of this library is that it does not weight the inputs or have any parameters at all. The way it solves $y = 4x$ is by a multiplying a constant scalar node of 4 and x or summing four x leaf nodes. This makes it a hard task to identify dynamic systems.

Bloat control is implemented using the Parsimony pressure. However, this method requires a significant amount of tuning, as explained in Chapter 3, to work correctly. While the library offers the option to automatically tune

this parameter while going through the generations, the individuals tend to converge to a tree with a single node.

GPLearn was tested on data from simulation of the MSD system. The equation that the algorithm is trying to converge to is shown in (4.3). It did not converge to the correct solution in any of the runs. The best result is shown in (4.4). This end result is not that poor, but considering the simplicity of the system, a better result is to be expected. Only one out of 15 runs converged to a reasonable solution.

$$\ddot{x} = 0.05\tau - 0.5\dot{x} - 0.6x \tag{4.3}$$

$$\ddot{x} = 0.04\tau - 0.354\dot{x} - 0.531x \tag{4.4}$$

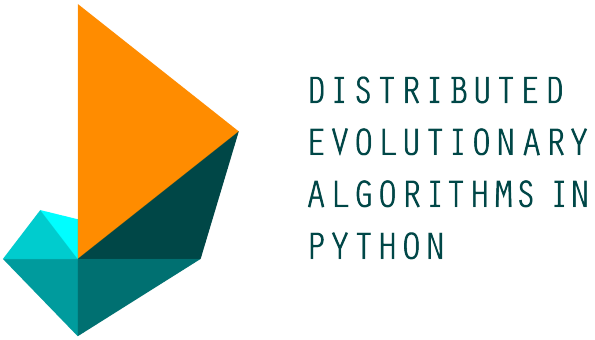
It is likely that the algorithm would perform better with parameters on the nodes, and regression of these, after each generation. The option to set the *max depth* would likely be an effective bloat control. Implementation of the orthogonal least squares to remove the branches that contributes only a little to the fitness of the solution, may improve the performance significantly, as bloat is a problem with this library.

This library is not programmed in a way that allows for much additional implementation of the new functionalities explained above, and this is the drawback of the simplicity of this library.

4.4 Distributed Evolutionary Algorithms

Distributed Evolutionary Algorithms in Python (DEAP) is a library with an extensive set of features[11]. This library does not limit itself to just GP, but also other evolutionary programming algorithms.

The advantage of DEAP is that it's easily customizable, such as what type of crossover and what mutation method to use, and the possibility to create user-defined functions in the function set. The library does not provide a given fitness function, so this has to be created for the given problem, which provides a lot of flexibility.



One of the drawbacks of this library, as was the case for GPlearn, is that it does not handle the constant terms and parameters well with the given functionalities in the library. This library does not have ordinary least squares or regression algorithm built in. But because it does not provide a predefined fitness measure, this can be built into

the algorithm to fit the need. Such a method has been implemented for this thesis, and is described in further detail in Chapter 5

The bloat control methods implemented in DEAP is *Max depth*, which limits the max tree size, and double tournament selection [19]. Parsimony pressure is easy to implement, but not implemented in the current version.

Using the default functions in DEAP gives a slightly better result than with GPlearn when it comes to finding the correct solution to MSD system, but not nearly as well as the Matlab Toolbox. It's more by chance that makes it able to find a suitable model than the algorithm itself, and with a more complex model the chance of finding it becomes slim and time-consuming.

However, due to the flexibility and the large set of features offered by this library, it is possible to customize to the problem and achieve better results. Hence, for the remainder of the thesis, DEAP will be the library to use in GP.

Chapter 5

Customizations to the DEAP Library

In this chapter, extensions to the DEAP library such as ordinary least squares, linear in parameters model and orthogonal least squares method will be elaborated. How the ideas presented in this chapter are incorporated in the GP algorithm is shown in Appendix G

The hyperparameters of the GP algorithm will be tested on the mass-spring-damper system, to find well suited numerical values for this implementation. Finally, the algorithm's robustness to noise and time delay is examined.

5.1 Extending the Algorithm

By utilizing that GP outputs models that are Linear-In-Parameter (LIP), the LS method can be used to find the optimal parameters. LIP models can be described by (5.1), where y is the desired output (ground truth), k is the k-th time instant, p_i is the parameter to the function F_i, X(k) is the state and input vector, and M is the number of functions.

$$y(k) = \sum_{i=1}^{M} p_i F_i(X(k)) \tag{5.1}$$

To find the functions $F_1, ..., F_M$ the model needs to decomposed into the smallest nonlinear parts. This is done by traversing the tree, by starting at the root and splitting when a linear node is found. Linear nodes can be $\{+, -\}$. If a non-linear or terminal node is found this branch is saved as a function. This is shown in the following example.

5.1.1 LIP Decomposition Example

Consider the individual shown in Figure 5.1. The root is a linear node, hence the tree is split into the left and right subtree. The left subtree has a non-linear root and is therefore saved as F_1. The right subtree has a linear root and

is split once more. This continues until all nodes are explored. When a terminal (leaf) node is found it is saved as a function. The result of this decomposition is shown in Figure 5.2.

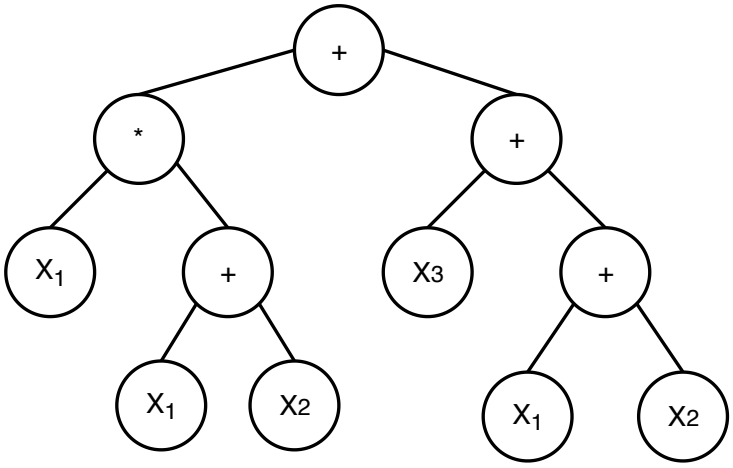


Figure 5.1: An individual shown in a tree structure.

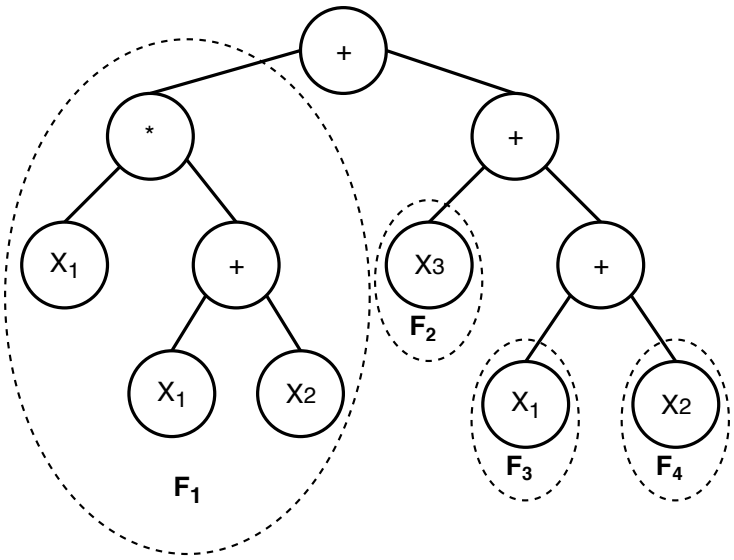


Figure 5.2: An individual split into functions.

The complete function resulting from this procedure is shown in (5.3). Whether or not to keep p_0 nonzero is problem dependent.

$$y = p_0 + p_1 F_1(\mathbf{x}) + p_2 F_2(\mathbf{x}) + p_3 F_3(\mathbf{x}) + p_4 F_4(\mathbf{x}) \tag{5.2}$$

$$y = p_0 + p_1 x_1 (x_1 + x_2) + p_2 x_3 + p_3 x_1 + p_4 x_2 \tag{5.3}$$

5.1.2 Optimal Parameters with Ordinary Least Squares

LS was implemented in the Matlab toolbox, the following shows how it was implemented together with DEAP. This method is fast, as it can be solved explicitly rather than iteratively. After the individual is split into the LIP model, the parameters $p_0, \ldots, p_M$ needs to be identified. This is done by solving for p in (5.4).

$$e = \frac{1}{N}\sum_{k=1}^{N}(y(k) - \sum_{i=1}^{M} p_i F_i(X(k)))^2 \tag{5.4}$$

The result of solving for for p is shown in (5.5) [28].

$$p = (F^T F)^{-1} F^T y \tag{5.5}$$

Where F is shown in (5.6).

$$F = \begin{bmatrix} F_1(\mathbf{X}(1)) & \dots & F_M((\mathbf{X}(1)) \\ \vdots & \ddots & \vdots \\ F_1(\mathbf{X}(N)) & \dots & F_M(\mathbf{X}(N)) \end{bmatrix} \tag{5.6}$$

The issue with (5.5), is that the Gramian matrix $F^T F$ might be singular/ linearly dependent, in that case the inverse does not exist. The Gramian will be singular if two functions are equal. The data might also make it singular. If $x_1 = x_2$ and $F_1 = x_1$, $F_2 = x_2$ then the function values will be the same.

This limitation has to be safeguarded for in the code, such that if the determinant of the Gramian is zero the program does not try to calculate the inverse. The fitness of an individual with a zero determinant of the Gramian will be set high, e.g. MSE = 1000. The Gramian will be singular if two functions are equal. This is not a problem as the optimal solution does not have to have two equal functions. But this might slow the evolution process a bit as these solutions will be regarded as having poor fitness, and thus, useless.

This has not caused any issues for the work this thesis, as the variables are not equal, and the individuals that contain equal functions will be selected out of the population when they occur.

5.1.3 Further dividing the functions of the individual

When an individual is divided, as shown in section 5.1.1, the smallest linear in parameters model will not be found in every case. An example of this is shown below, where the method explained in the section 5.1.1 gives (5.8). In (5.9), F_1 is divided into the smallest LIP parts. Weighting all the parts optimally, gives equal or better fit.

$$F_1 = (a + b)(c + d) \tag{5.7}$$

$$pF_1 = p_1(a + b)(c + d) \tag{5.8}$$

$$pF_1 = p_1 ac + p_2 ad + p_3 bc + p_4 bd \tag{5.9}$$

The selected implementation of this concept utilizes *Sympy*, a symbolic mathematics library for Python. In *Sympy* the function *expand* expands all the brackets of a symbolic expression. The model containing the smallest non-linear parts is found by expanding the function, and checking if they are different from the original. Then split at + and − if they are.

This implementation does not give a better final results, but should in theory provide faster convergence as more individuals are viable. An implementation of the standard model [9], with control inputs is shown in Chapter 6, and shows a LIP model. The ideal model might look similar to this.

This concept is only implemented in the chapter with real data from Odin's sensors, as it was in this work the idea was conceived.

5.1.4 Orthogonal Least Squares

Orthogonal least squares (OLS) is presented in several of the papers reviewed in the literature, and can help reduce bloat and make the evolution converge faster [22][12]. OLS is a technique that transforms the columns of the matrix F into a set of orthogonal basis vectors. These can be used to examine the individual function's contributions to the fitness [3]. In this context it can be used to find out how useful a branch of the tree is, and set a threshold of remove useless branches. This is explained in further detail by J. Madár and J. Abonyi [23].

OLS was implemented in the GP algorithm for this thesis, however, it did not give a faster convergence in run time. As OLS is computed for each individual, in each generation, this makes the GP algorithm slower per generation. The added computational time might also be attributed to the implementation of the method. After several test runs, OLS were considered to not give a positive contribution to the algorithm, and hence, discarded.

5.2 Optimal Hyperparameters

In this section different hyperparameters will be experimented with, to find the parameter that give the best performance. The model that will be identified is the MSD system. The parameter that is not tested in the sections will be the parameters listed in Table 5.1. The input to the MSD system will be a square wave, show in in Figure 5.3, for all tests as this will give the most unambiguous results.

Hyperparameters and Evolutionary Strategies	Type / Value
Operators	+, -, *, abs,
Tournament size	3
Max Depth	7
Fitness Evaluation	LS with MSE
Mutation Method	mutUniform
Crossover Method	cxOnePoint, with min = 0 max = 2

Table 5.1: List of hyperparameters and evolutionary strategies for evolving a model for the MSD system.

5.2.1 Generations vs Populations Size

The two most obvious parameters to test is the number of generations and the population size. The choice of these have the most impact in the run time of the algorithm. The tests will be be run on a grid with population size and number of generations shown in (5.10) and (5.11). The GP algorithm was run 10 times for each choice of parameter.

$$N_{gen} = [2, 5, 10, 20, 50, 100, 200] \tag{5.10}$$

$$N_{pop} = [5, 20, 50, 200, 500] \tag{5.11}$$

The results are shown in Table 5.2, and it is clear that the population must be large enough, about 200 to 500, to reliably converge to the correct solution. For this system, the number of generations does not seem affect the final result as much as the size of the population. This is a consequence of this system being relatively simple, as the initial population can contain something close to, or the actual correct solution.

		Population				
		5	20	50	200	500
Generations	2	0.00139, 1	0.0011073, 1	0.000580, 5	0.00034, 7	1.22339e-08, 10
	5	0.00149, 0	0.00094241, 2	0.0006533, 4	1.2144e-08, 10	1.11468e-08, 10
	10	0.00151, 0	0.0011000, 2	0.0007514, 3	8.09751e-09, 10	7.92189e-09, 10
	20	0.00149, 0	0.001093, 1	0.000687, 4	9.145276e-09, 10	4.67887e-09, 10
	50	0.001448, 0	0.0011429, 1	0.000397, 6	9.111321e-09, 10	2.134450e-09, 10
	100	0.00139, 1	0.000624, 4	0.000558, 5	0.0001178, 9	2.456480e-09, 10
	200	0.00146, 0	0.001170, 0	0.000235, 8	8.06421e-09, 10	2.11144e-09, 10

Table 5.2: Average MSE, and number of times the correct solution were found on the test set. The algorithm were ran 10 times.

5.2.2 Mutation vs Crossover ratios

To explore the effect of the different evolution strategies, the ratio of crossover and mutation will be tested. A grid off different ratios will be tested with a few different number of generations and population sizes. The grid of ratios will be [0, 0.2, 0.4, 0.6 0.8, 1] for both evolution methods. The algorithm will be run 10 times for each configuration of parameters.

The average MSE is not a great measurement of how the algorithm is performing, as an average is heavily skewed with outliers. How many times out of the ten times the algorithm converges to the correct solution is a more accurate representation. 5e-6 was found to be a good divider, as solutions below this threshold for all tests viewed were the correct solution.

The mutation method used is the *mutUniform* method, and the crossover method is the *cxOnePoint* method, both from the DEAP library. The result from using a population size of 50 and 20 generations is shown in Table 5.3.

With a higher population size the algorithm finds the correct solution most of the time, regardless of the mutation and crossover ratio, as this is a simple system. This table shows that crossover is the most important part of the evolution process for this system, as expected [5], and that a pure mutations strategy is inferior.

		Mutation ratio					
		0	0.2	0.4	0.6	0.8	1
Crossover ratio	0	0	0	0	0	10	10
	0.2	40	40	40	30	50	-
	0.4	50	40	90	30	-	-
	0.6	30	50	40	-	-	-
	0.8	90	60	-	-	-	-
	1	50	-	-	-	-	-

Table 5.3: Percentage of correct solution found when the population size and the number of generations is 50 and 20 respectively. To be considered a correct solution the MSE have to be below 5e-6.

This test is not comprehensive enough to conclude which is the optimal combination. However, the results shows how much of the algorithm is driven by stochastic actions. Thus, to get an unambiguous result, would require a vast number of runs.

5.3 Noise and Time Delay

The algorithm's robustness to noisy and time delayed data was tested, where this introduced in the simulation data. This has been tested before, in the specialization project [14]. However, as this is a new library, with new implementations shown in Section 5.1, new test were ran.

5.3.1 Noise on the output

White noise with different amplitude were added on the output, $\ddot{x}$. The noise was generated as shown in (5.12), with a random number generator from *Numpy.random*.

$$\ddot{x} = \ddot{x} + amp * rand([-1, 1]) \tag{5.12}$$

A MSD system with m = 10, d = 13 and k = 5 was simulated and noise with different amplitude was added. The input to the system is force in a square wave with amplitude of 1 [N], duty cycle of 50% and a period of 6π. The input is shown in Figure 5.3 The system was simulated for 50 seconds with a sampling time of 0.1s. The initial conditions, x_0 and $\dot{x}_0$, were set to 0.

The equation the GP algorithm seeks to identify is (5.13). The results are shown in Figure 5.4. The results with other amplitudes of noise are shown in Appendix A. From these figures it is clear that the algorithm finds the correct solution, even with a substantial amount of noise added to the output, $\ddot{x}$.

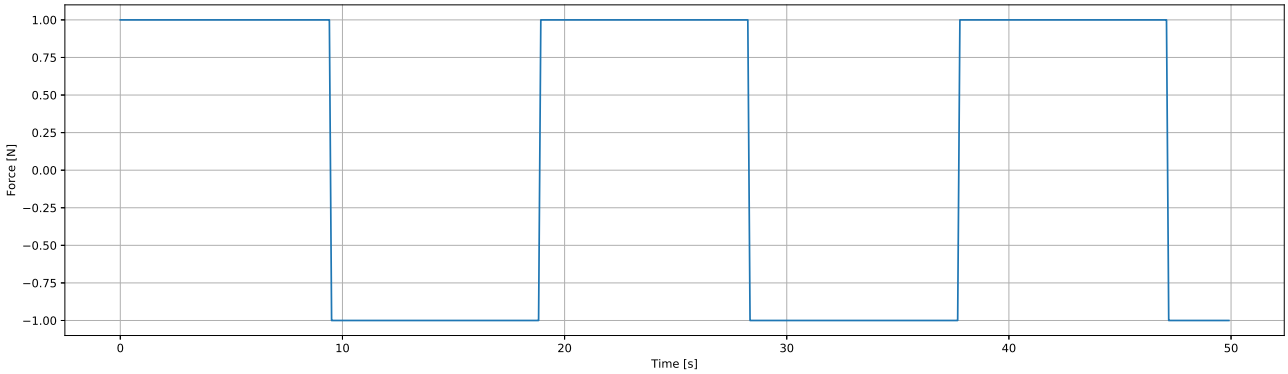


Figure 5.3: The input, τ, to the MSD system when generating data.

$$\ddot{x} = -1.3\dot{x} + 0.1\tau - 0.5x \tag{5.13}$$

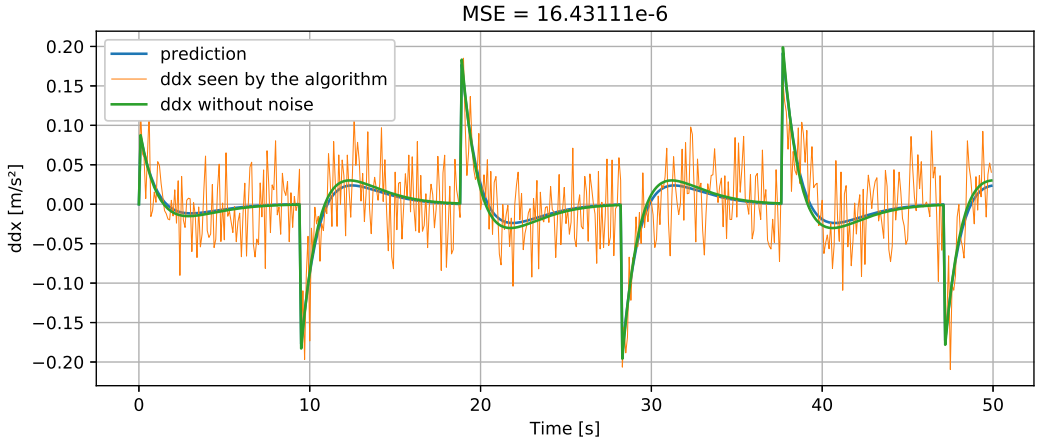


Figure 5.4: Result from adding high amplitude noise on $\ddot{x}$, The calculated MSE is between the noise-free $\ddot{x}$ and the found solution. Noise with amp ± 0.1, the equation found was: $\ddot{x} = -1.187\dot{x} + 0.0965 * \tau - 0.480 * x$

5.3.2 Noise on the inputs

For the real system, Odin, noise will be present on all the variables. The following section investigates how the GP algorithm handles noise on the inputs, $\dot{x}, x$ and τ. The added noise were set to the same scale as the individual signals as shown in (5.14), where the Noise Factor, NF_{all}, were set equal for all three variables. The noise on the inputs are shown in figure 5.5

$$
\begin{aligned}
NF_{all} &= a \\
NF_x &= max(x) - min(x) \\
x_{noisy} &= x + a * NF_x * rand([-1, 1])
\end{aligned} \tag{5.14}
$$

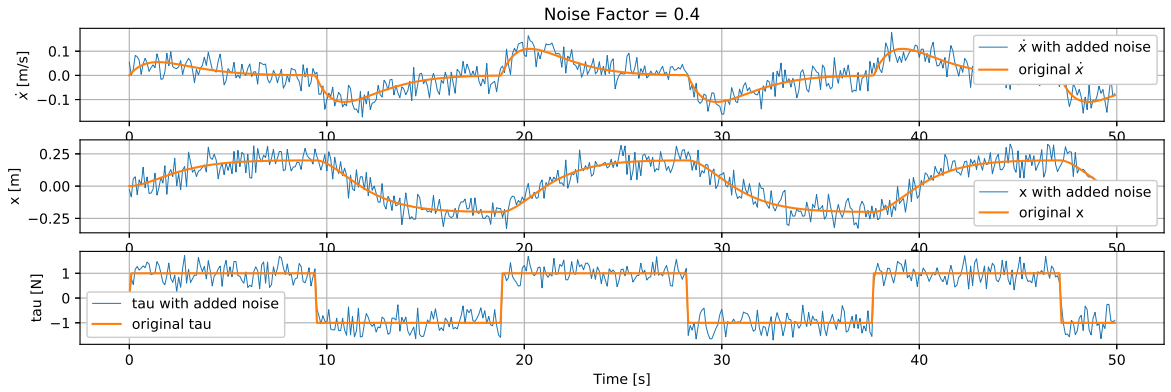


Figure 5.5: Noise added on the inputs.

The amplitude of the added noise on the inputs affect the algorithm's ability to find the correct model, more than with the same amplitude on $\ddot{x}$. This is shown in Figure 5.6, where a noise factor of 0.3 on the inputs results in a wrong equation found. More plots with different amplitudes are shown in Appendix A.

A better result would be found if the number of generations or a larger population was be chosen, but is meant to illustrate the difficulties when handling input noise compared to noise on the output, $\ddot{x}$.

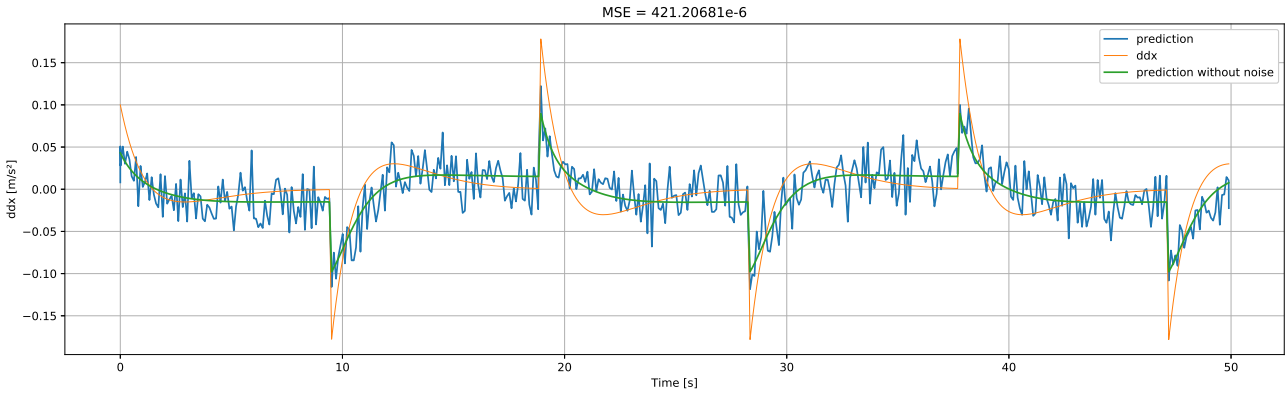


Figure 5.6: Results from adding noise to the signal of $\dot{x}$, x and τ. The MSE is calculated with the noise free variables and $\ddot{x}$. Noise Factor = 0.3, found equation: $\ddot{x} = (1.0742\dot{x} - 0.4060)(1.0742\dot{x} - 0.1061\tau + 0.7137x)$

5.3.3 Noise on all variables

Testing how different amplitudes of noise affects the performance of the algorithm. The logarithmic array shown in (5.15) was used as the noise factor for inputs and outputs. It becomes apparent that the algorithm performs poorer on noise on the inputs compared to the outputs from Figure 5.7

$$arr = \begin{bmatrix} 0.0 & 0.01 & 0.0193 & 0.037 & 0.0719 & 0.138 & 0.268 & 0.517 & 1.0 \end{bmatrix} \tag{5.15}$$

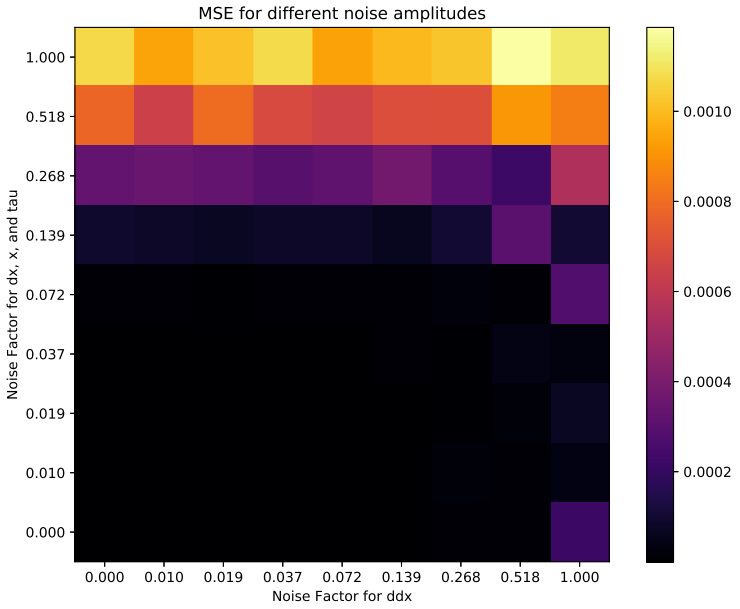


Figure 5.7: How noise on inputs vs outputs affects the performance on the algorithm.

5.3.4 Time delay

The algorithms robustness with regard to time delays is tested. A time delay on the input was added. The time delay in the system was implemented as shown in Figure 5.8. With a time delay of 0.5s the algorithm did not find the correct solution. To deal with this, a simple brute force method was implemented to account for the time delay. This approach was implemented in the specialization project, and showed good results [14]. The assumption this approach builds on is that with the adjustment in the input, the algorithm finds the correct or close to the correct solution, resulting in a good fit when the time delay is accounted for.

The input force, τ, was shifted n timesteps into the 'future' as shown in code 5.1, and the algorithm were run to find a suitable model. This were tested for several different delays, and the result with a time delay of 0.5 is shown in Figure 5.9. This plot shows a clear trend, and it is easy to see what time delay that were implemented in the data.

Listing 5.1: The implemented time delay compensation

```
for dela in numpy.arange(0,1,0.05):
        i = 0
        while dela > t[i]:
            i = i+1

        for j in range(len(ddx)):
                if j-i < 0:
                        tau[j] = tau_orig[0]
                else:
                        tau[j] = tau_orig[j-i]
```

11

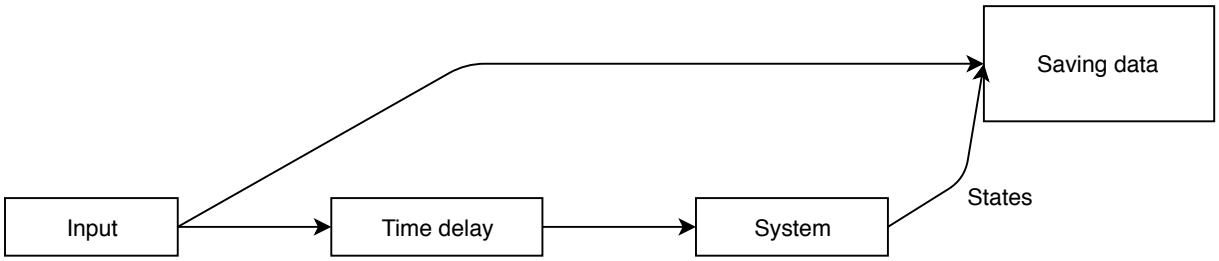


Figure 5.8: The implemented time delay

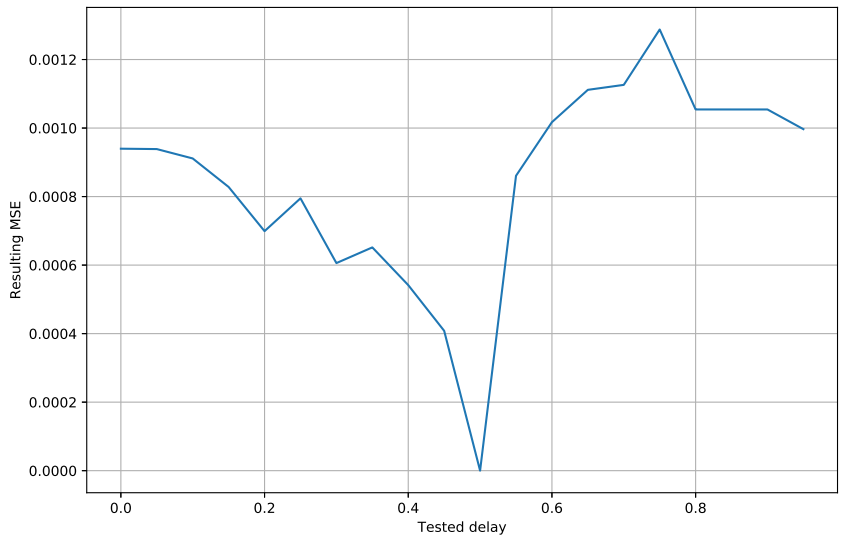


Figure 5.9: Resulting MSE when tested different time delay corrections. The implemented time delay in this case were 0.5s

5.3.5 Noise and time delay

As shown in the previous sections, the algorithm were able to find the correct solution with added noise up to a certain amplitude. The brute force method were able to find, and compensate for the added time delay in the input signal to the system. The brute force method were built on the assumption that the algorithm finds a model with the least MSE when the delay are completely compensated for. In a real system there might be both noise and time delay, and the algorithm should be able to deal with this. The results from the previous chapters would suggest that the algorithm is able to handle this, as it finds the correct solution when noise up to a certain amplitude of the signal are included.

The results from adding noise on $\ddot{x}$, and then on all the variables $\ddot{x}, \dot{x}, x$ and τ, are shown in Figure 5.10 and 5.11. These results suggests that the brute force method works even with noisy signals, but its not as clear what the time delay is when noise on all the variables are included.

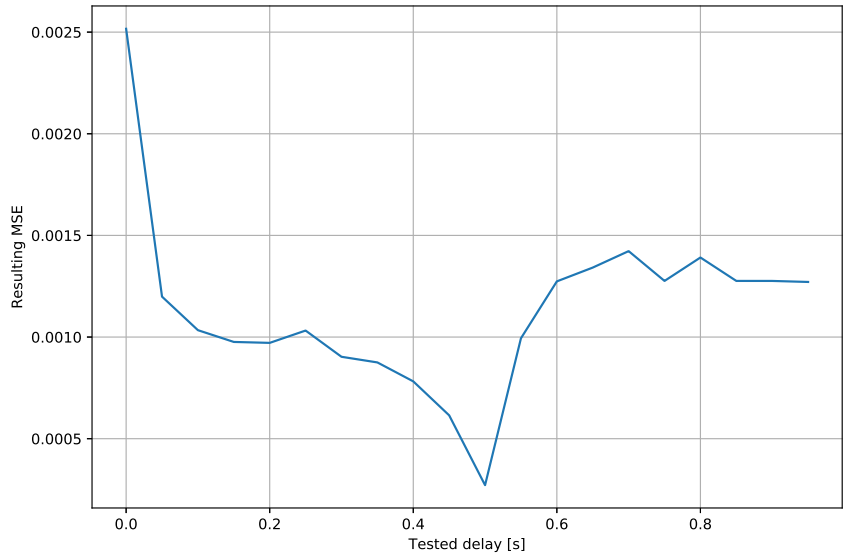


Figure 5.10: Resulting MSE when different time delay corrections were tested with noise on the output. Noise were added with noise factor = 0.1. The implemented time delay in this case were 0.5s.

This approach to time delay shows promising results. However, for the actual data from the sensors of Odin, this approach proved to be unnecessary as the GP algorithm worked well without it.

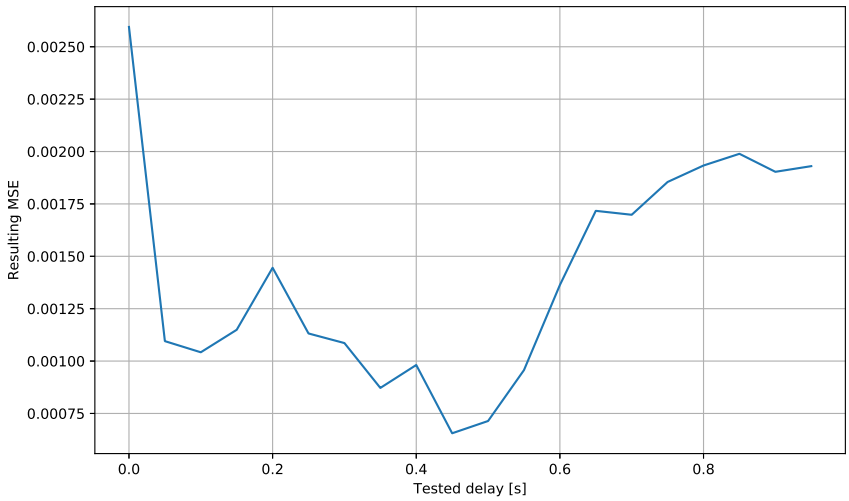


Figure 5.11: Resulting MSE when different time delay corrections were tested with noise. Noise were added on $\ddot{x}, \dot{x}, x$ and τ, with noise factor = 0.1. The implemented time delay in this case were 0.5s.

Chapter 6

USV Simulation

The GP algorithm with the implemented functions explained in the previous chapter works well for the MSD system. To further test how it performs on a more complex system, the natural selection would be the standard model, as shown in this chapter.

6.1 3-DOF model

A 3-DOF model of Odin was created by FFI and serves as a way of testing how the algorithm performs on more complex systems. The model also contains a model of the water jets.

The 3-DOF model is shown in (6.1) and (6.2). In this model only *surge*, *sway* and *yaw* is modeled. *heave*, *roll* and *pitch* ar omitted as they are dominated by noise from waves, and it's not interesting to control the vehicle after. Hence, in the real data a filter will try mitigate these disturbances. All 6-DOF are shown in Figure 6.1.

$$Kinematics:$$

$$\begin{bmatrix} \dot{x}_b^n \\ \dot{y}_b^n \\ \dot{\psi}_b^n \end{bmatrix} = \begin{bmatrix} \cos(\psi_b^n) & -\sin(\psi_b^n) & 0 \\ \sin(\psi_b^n) & \cos(\psi_b^n) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_b \\ v_b \\ r_b \end{bmatrix} \tag{6.1}$$

$$Kinetics:$$

$$\mathbf{M}\dot{\nu}_{b/n}^b + \mathbf{C}(\nu_{b/n}^b)\nu_{b/n}^b + \mathbf{D}(\nu_{b/n}^b)\nu_{b/n}^b = \tau \tag{6.2}$$

Where n is the North East Down (NED) coordinate frame, 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. τ is the force and moments acting on the body. $\nu = [u,\ v,\ r]^T$ is the state vector for surge sway and yaw respectably.

The model that the algorithm outputs will have the form shown in (6.3)- (6.5) for force and moment inputs to the system. τ_x and τ_y is the force acting on the body in surge and sway respectably, while τ_z is the moment acting on the body in yaw.

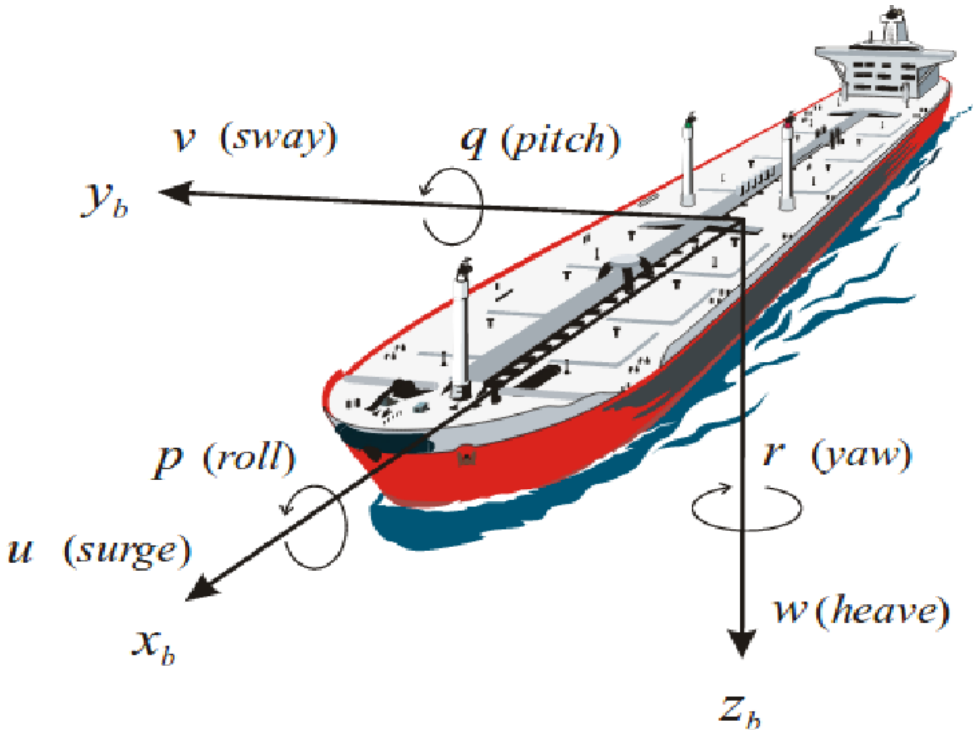


Figure 6.1: 6-DOF of a marine vehicle [16]

$$\dot{u} = f_u(u(t), v(t), r(t), \tau_x(t), \tau_y(t), \tau_z(t)) \tag{6.3}$$
$$\dot{v} = f_v(u(t), v(t), r(t), \tau_x(t), \tau_y(t), \tau_z(t)) \tag{6.4}$$
$$\dot{r} = f_r(u(t), v(t), r(t), \tau_x(t), \tau_y(t), \tau_z(t)) \tag{6.5}$$

To compare the model from the GP algorithm and the true model in the simulation, the model in matrix form in (6.2) have to be derived on component form. The system on component form is shown in (6.6)- (6.8), and this is the task of the GP algorithm to find. The matrices used to derive this model is shown in Appendix C.1.

$$\dot{u} = 2.026 \cdot 10^{-4}\tau_x + vr - 0.0101u - 0.0492|u|u \tag{6.6}$$
$$\dot{v} = 2.026 \cdot 10^{-4}\tau_y - ur - 0.045v - 0.4052|v|v \tag{6.7}$$
$$\dot{r} = 4.78 \cdot 10^{-5}\tau_z - 0.0612(v + r) - 0.1422|r|r \tag{6.8}$$

6.1.1 Water Jets

The real system, Odin, uses 2 Hamilton water jets as propulsion. With the exception of docking, the two jets have the same nozzle angle and the same motor RPM. The system is then viewed to have 2 inputs, and hence the system is under actuated. The two inputs will be denoted as δ_n and δ_t for the nozzle angle and motor RPM respectably, throughout this thesis.

The conversion from the inputs, motor RPM and nozzle angle, to forces is shown in this section. This control allocation was developed by FFI, using curve fitting on the data provided by Hamilton. In the data, the surge

velocity were given in knots, hence the conversion from m/s to knots in shown in (6.15).

The model from the GP algorithm will have the form shown in (6.9)- (6.11).

$$\dot{u} = f_u(u(t), v(t), r(t), \delta_r(t), \delta_t(t)) \tag{6.9}$$
$$\dot{v} = f_v(u(t), v(t), r(t), \delta_r(t), \delta_t(t)) \tag{6.10}$$
$$\dot{r} = f_r(u(t), v(t), r(t), \delta_r(t), \delta_t(t)) \tag{6.11}$$

The equations from δ_n and δ_t to $\tau_{x,y,z}$ is shown in (6.12) - (6.14)

$$\tau_x = 2 \cdot thrust \cos(\delta_n) \tag{6.12}$$
$$\tau_y = 2 \cdot thrust \sin(\delta_n) \tag{6.13}$$
$$\tau_z = -3.82\tau_y \tag{6.14}$$

Where thrust is calculated as shown in (6.15) - (6.18):

$$\hat{u} = 1.94u \tag{6.15}$$
$$thrust = (a_0 + a_1\hat{u} + a2\hat{u}^2) \cdot (1/4530(r_0 + r_1\delta_t + r2\delta_t^2)) \tag{6.16}$$
$$thrust = 118.31 - 2.47\delta_t + 7.35 \cdot 10^{-3}\delta_t^2 - 6.56u + 0.14u\delta_t - 4.07 \cdot 10^{-4}u\delta_t^2 + 0.063u^2 \tag{6.17}$$
$$- 1.31 \cdot 10^{-3}u^2\delta_t + 3.90 \cdot 10^{-6}u^2\delta_t^2 \tag{6.18}$$

The final model, when combining (6.6)- (6.8) and (6.12) - (6.14) with thrust from (6.18), is shown in (6.19) - (6.23)

$$g(\delta_t, u) = 2.026 \cdot 10^{-4} \cdot 2 \cdot thrust \tag{6.19}$$
$$g(\delta_t, u) = 0.048 - 1.0 \cdot 10^{-3}\delta_t + 2.98 \cdot 10^{-6}\delta_t^2 - 2.65 \cdot 10^{-3}u + 5.67 \cdot 10^{-5}u\delta_t - 1.65 \cdot 10^{-7}u\delta_t^2$$
$$+ 2.55 \cdot 10^{-5}u^2 - 5.31 \cdot 10^{-7}u^2\delta_t + 1.58 \cdot 10^{-9}u^2\delta_t^2 \tag{6.20}$$
$$\dot{u} = g(\delta_t, u) \cdot \cos(\delta_n) + vr - 0.0101u - 0.0492|u|u \tag{6.21}$$
$$\dot{v} = g(\delta_t, u) \cdot \sin(\delta_n) - ur - 0.045v - 0.4052|v|v \tag{6.22}$$
$$\dot{r} = -3.82g(\delta_t, u) \cdot \sin(\delta_n) - 0.0612(v + r) - 0.1422|r|r \tag{6.23}$$

6.2 Validation and Test set

Normal practice, when working with data-driven modelling, is to divide the data into two or three sets, training, validation and test set. The training set is what the algorithm trains on, validation is used during training, while the test set is there to make sure that the result is unbiased with respect to the validation set. This procedure is to make sure the end result is not overfitted to the data. Overfitting is when the algorithm finds a great result to the data it has got, but not to unseen new data, i.e. not generalizing well.

When using a simulation of a system, the test and validation sets can, for instance, be the same model but with different inputs. For the simulation of a USV in this chapter, the data is tested on data from one set of inputs and validated on another. When the MSE of the test set reaches a given threshold, the found model is tested on the validation set. If the fitness of the validation set is low as well, the algorithm terminates, as it probably has found a suitable solution.

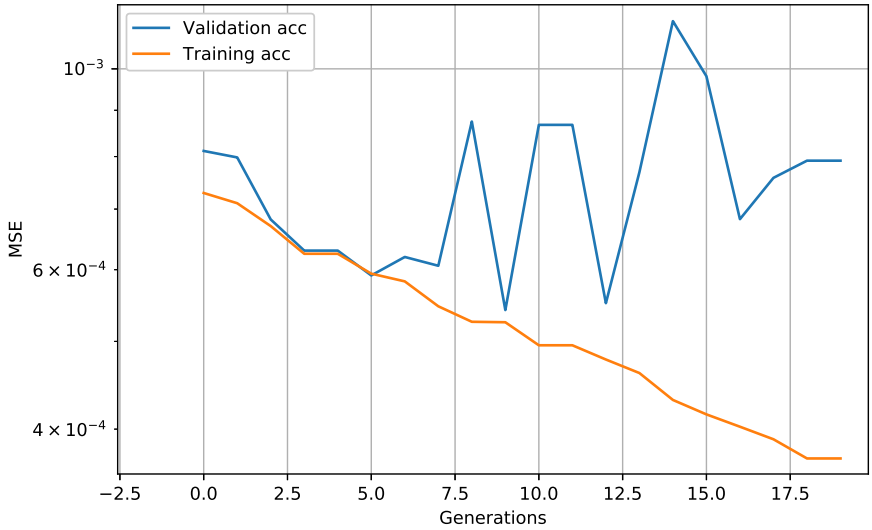


Figure 6.2: Validation and training set MSE over generations. This shows where the GP algorithm tries to fit the data from the training set perfectly, and hence overfits to the training data.

This works well for the simulation data, where the exact solution is known and gives an MSE close to zero. For the real data, this method does not work as well. For the real data, the training tends to look as shown in Figure 6.2. As there is some noise in the data, the model will not have a perfect fitness. The approach chosen in this thesis, when working with real data, is to save the model that gave the best validation score. The algorithm is terminated if the validation score has not decreased by n generations. This is usually referred to in machine learning as early stopping.

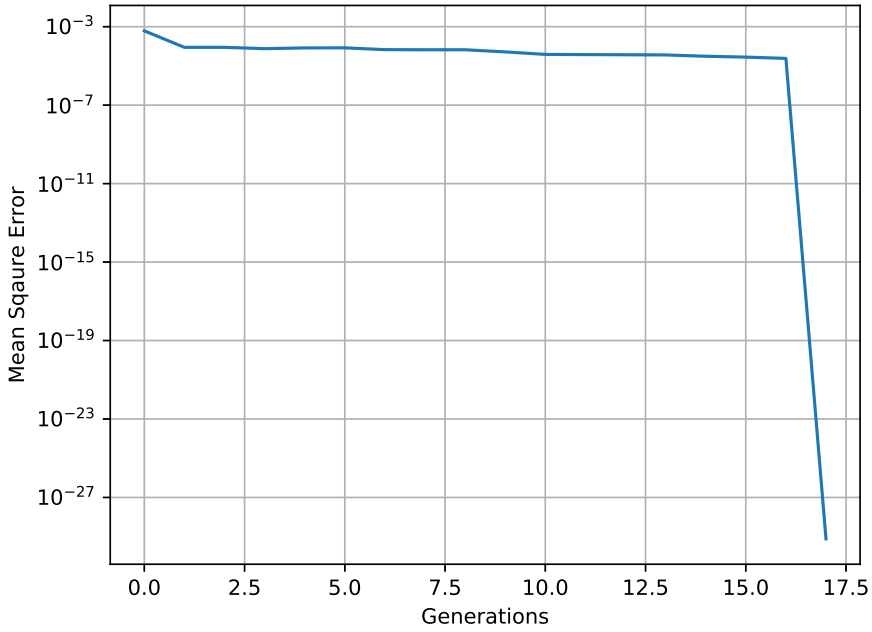


Figure 6.3: How the MSE looks when the algorithm finds the correct solution. This history plot is from the training of the USV model with force as input

6.3 Model with Force as Inputs

In this section, a model for the vehicle dynamics will be found using GP. The inputs to the system that is modelled, will be forces in surge and sway and moment in yaw.

6.3.1 Inputs

What inputs to force on the system is important, to get data that describes the entire dynamic of the vehicle. A couple of different inputs was forced on the system, these are shown in Figure 6.4 - 6.6 and is named input set 1 - 3 respectably.

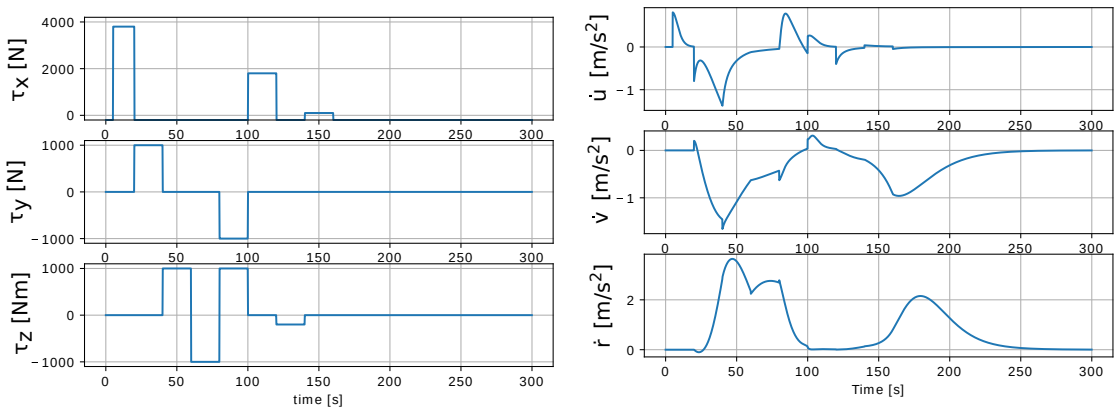


Figure 6.4: Input set 1, showing the input force and the resulting acceleration.

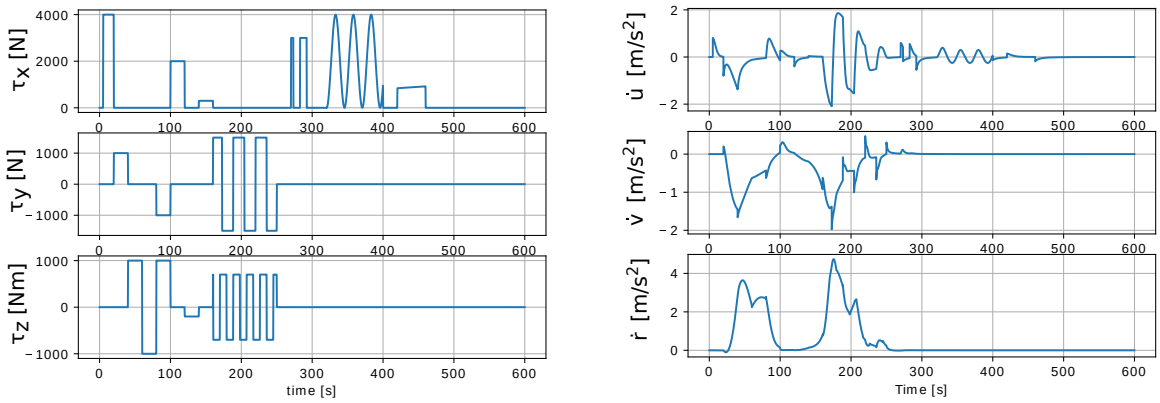


Figure 6.5: Input set 2, showing the input force and the resulting acceleration.

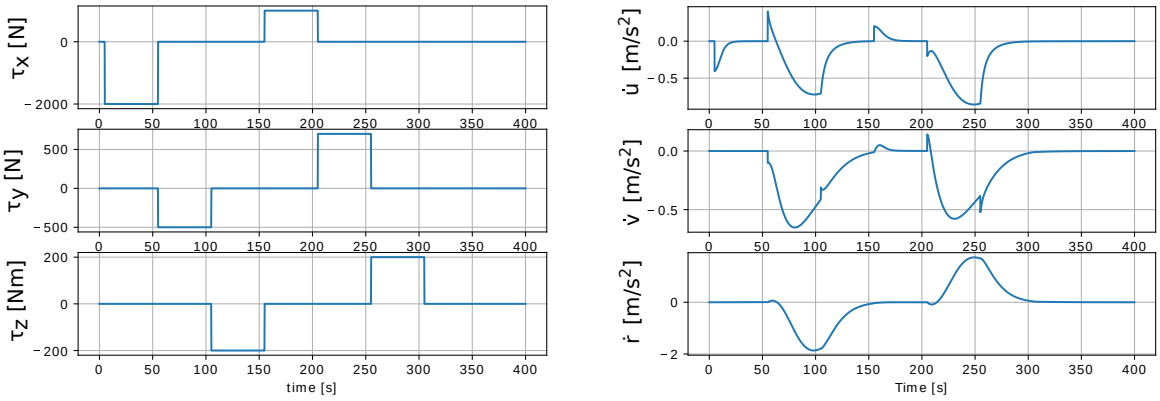


Figure 6.6: Input set 3, showing the input force and the resulting acceleration.

6.3.2 Result

The GP algorithm was run with the data logged from forcing the inputs shown in Section 6.3.1 on the system. The configuration that is used are shown in Table 6.1.

$$input\ 1:$$

$$\dot{u} = 1.0rv + 0.00020259\tau_x - 0.010096u - 0.049200 * |u^2| \tag{6.24}$$

$$\dot{v} = -1.0ru + 0.00020260\tau_y - 0.40520v|v| - 0.04499 * v \tag{6.25}$$

$$\dot{r} = -0.14221r|r| - 0.06123r + 4.7800 \cdot 10^{-5}\tau_z - 0.0613v \tag{6.26}$$

Hyperparameters and Evolutionary Strategies	Type / Value
Operators	+, *, abs
Ephemeral Constant	None
Tournament size	5
Max Depth	10
Fitness Evaluation	LS with MSE
Population	5000
Prob. Mutation	0.3
Prob. Crossover	0.5
Mutation Method	mutUniform
Crossover Method	cxOnePoint, with min = 0 max = 2
Generations	30 (or to the point where the validation result are better than 1e-8)

Table 6.1: The hyperparameters used when identifying the 3-DOF model with force as input

input 2 :

$$\dot{u} = +0.99999rv + 0.0002026\tau_x - 0.049200u^2 - 0.010099u \tag{6.27}$$

$$\dot{v} = -1.00ru + 0.00020260\tau_y - 0.405200v|v| - 0.044999v \tag{6.28}$$

$$\dot{r} = -0.142341r|r| - 0.06131r + 4.780000 \cdot 10^{-5}\tau_z - 0.06121v \tag{6.29}$$

input 3 :

$$\dot{u} = 1.0rv + 0.00020260\tau_x - 0.049199u|u| - 0.010100u \tag{6.30}$$

$$\dot{v} = -1.0ru + 0.0002026 * \tau_y - 0.405200v|v| - 0.0449999v \tag{6.31}$$

$$\dot{r} = -0.14220r|r| - 0.06111r + 4.780 \cdot 10^{-5}\tau_z - 0.061123v \tag{6.32}$$

The algorithm converged to the correct solution, (6.6) - (6.8), after 6 to 15 generations, on the first run for each equation and set of inputs. This suggests that the algorithm is powerful enough to find the model to less complex systems, with limited data.

An interesting observation is that the equations for $\dot{u}$ changes for the different inputs. In the simulation, expression is $|u|u$, while only the model with input 3 has this term in it. This is a result of input 1 and 2 only containing positive u-values, while in input 3 there are also negative values. With only positive u-values, u^2, $|u^2|$ and $|u|u$ are equal.

6.4 Model with Water Jets as Inputs

In the previous section a model for the vehicle dynamics were found. In this section a model containing the actuator dynamic and the vehicle dynamics is the objective of the GP algorithm. The model is known in advance, and described in section 6.1.1. This is a more challenging task for the algorithm, as the dynamics are more complex and trigonometric functions, sine and cosine, is needed to find the exact solution.

6.4.1 Inputs

Four different inputs will be forced on the system to see what inputs give the best result. The inputs are shown in Figure 6.7 - 6.9, and is named input set 1 to 3 respectably.

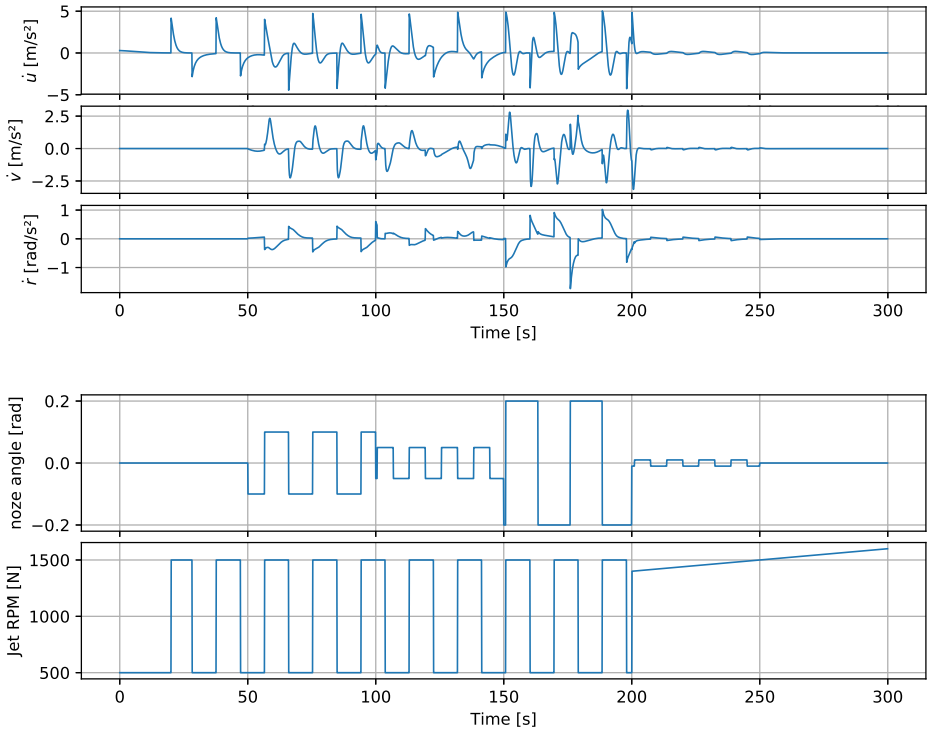


Figure 6.7: Water jet inputs and the acceleration for input 1.

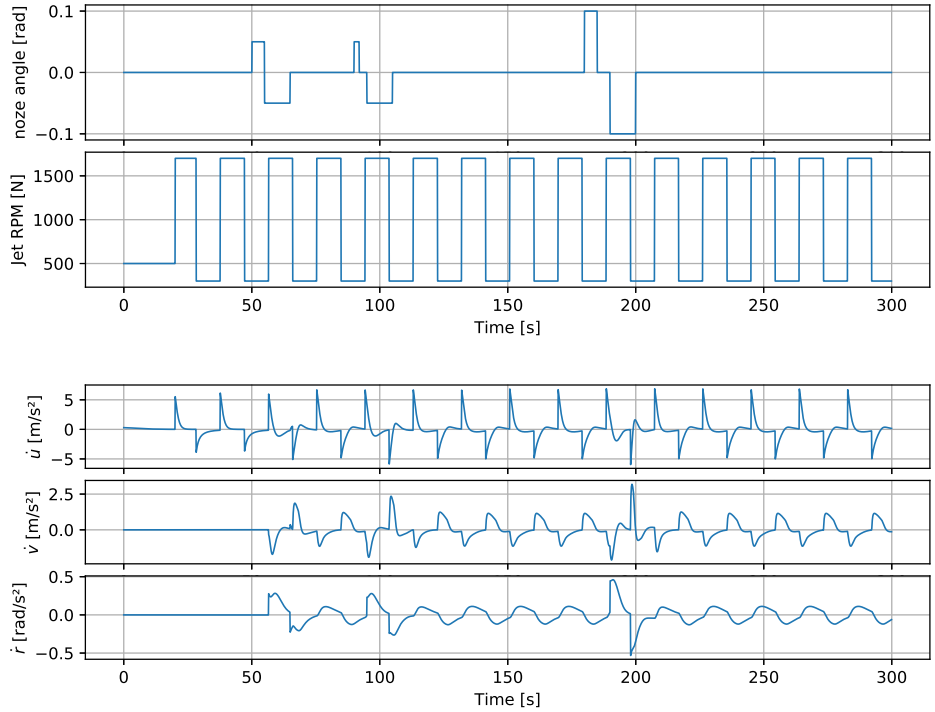


Figure 6.8: Water jet inputs and the acceleration for input 2.

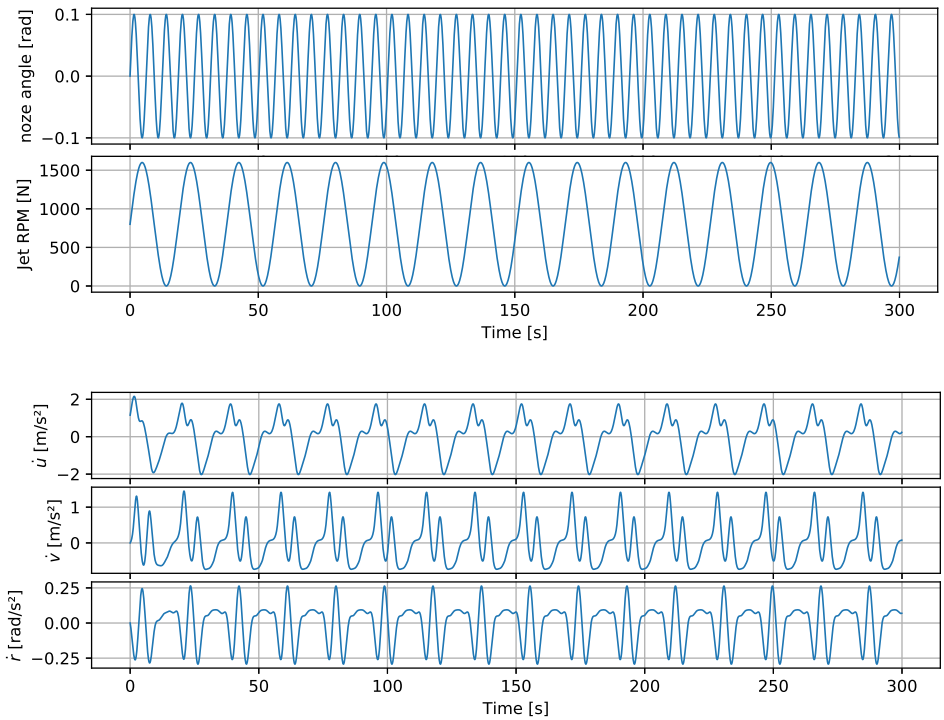


Figure 6.9: Water jet inputs and the acceleration for input 3.

6.4.2 Result

In this section the best results from running the GP algorithm on the inputs shown in the previous section. Unlike the results when using force as input in Section 6.3, the algorithm does not find a model that fits to the training and validation sets every time. This is likely due to the difference in complexity of the systems.

The criteria to terminate the run is that both the training and validation sets has a MSE less than 10^{-8}. When the threshold were set lower, the GP algorithm overfitted the model with respect to the validation set. This is a strict threshold, and would not be reasonable when noise and time delays is present but, since the system is ideal, this works fine. The hyperparameters used in this section is given in Table 6.2

Hyperparameters and Evolutionary Strategies	Type / Value
Operators	+, *, abs, sine, cosine
Ephemeral Constant	None
Tournament size	5
Max Depth	10
Fitness Evaluation	LS with MSE
Population	5000
Prob. Mutation	0.3
Prob. Crossover	0.5
Mutation Method	mutUniform
Crossover Method	cxOnePoint, with min = 0 max = 2
Generations	100 (or to the point where the validation result are better than 1e-8)

Table 6.2: Hyperparameters used when identifying the 3-DOF model with control inputs

6.4.2.1 Forward acceleration, $\dot{u}$

The equation the GP algorithm is trying to find is shown in Eq 6.33.

$$\begin{aligned}\dot{u} = {} & 0.048\cos(\delta_n) - 1.0\cdot 10^{-3}\delta_t\cos(\delta_n) + 2.98\cdot 10^{-6}\delta_t^2\cos(\delta_n) - 2.56\cdot 10^{-3}u\cos(\delta_n) \\ & + 5.67\cdot 10^{-5}u\delta_t\cos(\delta_n) - 1.65\cdot 10^{-7}u\delta_t^2\cos(\delta_n) + 2.55\cdot 10^{-5}u^2\cos(\delta_n) \\ & - 5.31\cdot 10^{-7}u^2\delta_t\cos(\delta_n) + 1.58\cdot 10^{-9}u^2\delta_t^2\cos(\delta_n) + vr - 0.0101u - 0.0492|u|u \end{aligned} \tag{6.33}$$

With input set 1 as the training set, and input set 2 as validation, the algorithm found the solution shown in Eq 6.34. This gives a near perfect result on data from input set 1,2 and 3.

$$\begin{aligned}\dot{u} = {} & 0.1325\delta_n^2\sin(\cos(\delta_n)) * \cos(\delta_n) + 1.3905\cdot 10^{-9}(\delta_t^2u^2 + \delta_tu^3) - 2.174\cdot 10^{-7}\delta_t * (\delta_t + \\ & u(\delta_t + u))\sin(\cos(\delta_n)) - 7.563\cdot 10^{-7}\delta_t|\delta_t| - 0.001005\delta_t + 1.000016rv - 0.0493117u(|\delta_t| + |u|) \\ & + 0.04936 * u * |\delta_t| - 0.0125u + 1.8141738\cdot 10^{-8}(\delta_tu + u^2)|\delta_t| + 4.6541\cdot 10^{-6} * (\delta_n + \delta_t(\delta_t + u) + \delta_t + r \\ & + \cos(\delta_n))\sin(\cos(\delta_n)) + 0.0477811\cos(\delta_n) - 0.000307|\delta_n| \end{aligned} \tag{6.34}$$

Rewriting Eq 6.34, and replacing $|\delta_t|$ with δ_t since $\delta_t \geq 0\ \forall\delta_t$, gives the following equation:

$$\begin{aligned}\dot{u} = &-0.0125u + \delta_t(4.61 \cdot 10^{-6} \sin(\cos(\delta_n)) - 0.001) + rv + 4.61 \cdot 10^{-6}\delta_t u \sin\cos\delta_n - 1.99 \cdot 10^{-7}u^2\delta_t \\ &- 1.99 \cdot 10^{-7}\delta_t^2 u + \delta_t^2(4.61 \cdot 10^{-6} \sin(\cos(\delta_n)) - 7.56 \cdot 10^{-7}) - 3.7 \cdot 10^{-4}|\delta_n| + 0.048\cos(\delta_n) - 0.049|u|u \\ &+ 1.61 \cdot 10^{-6}\delta_n \sin(\cos(\delta_n)) + 0.132\delta_n^2 \sin(\cos(\delta_n))\cos(\delta_n) + 1.39 \cdot 10^{-9}\delta_t^2 u^2 \\ &+ 1.39 \cdot 10^{-9}\delta_t u^3 + 4.61 \cdot 10^{-}6r\sin(cos(\delta_n))\end{aligned} \tag{6.35}$$

When Eq 6.35 is divided into smaller parts is becomes clear why the equation is a good fit. The following partial equations breaks down the main equation, and compares it to the actual equation for the system.

$Part\ 1:$

$$-0.0125u \quad \approx \quad -0.0101u \tag{6.36}$$

$Part\ 2:$

$$\delta_t(4.61 \cdot 10^{-6} \sin(\cos(\delta_n)) - 0.001) \quad \approx \quad -1.0 \cdot 10^{-3}\delta_t \cos(\delta_n) \tag{6.37}$$

$Part\ 3:$

$$rv = rv \tag{6.38}$$

$Part\ 4:$

$$4.61 \cdot 10^{-6}\delta_t u \sin\cos\delta_n \quad \approx \quad 5.67 \cdot 10^{-5}u\delta_t \cos(\delta_n) \tag{6.39}$$

$Part\ 5:$

$$-1.99 \cdot 10^{-7}u^2\delta_t \quad \approx \quad -5.31 \cdot 10^{-7}u^2\delta_t \cos(\delta_n) \tag{6.40}$$

$Part\ 6:$

$$-1.99 \cdot 10^{-7}\delta_t^2 u \quad \approx \quad -1.65 \cdot 10^{-7}u\delta_t^2 \cos(\delta_n) \tag{6.41}$$

$Part\ 7:$

$$\delta_t^2(4.61 \cdot 10^{-6} \sin(\cos(\delta_n)) - 7.56 \cdot 10^{-7})) \quad \approx \quad 2.98 \cdot 10^{-6}\delta_t^2 \cos(\delta_n) \tag{6.42}$$

$Part\ 8:$

$$-3.7 \cdot 10^{-4}|\delta_n| + 1.61 \cdot 10^{-6}\delta_n \sin(\cos(\delta_n)) \quad \approx \quad - \tag{6.43}$$

$Part\ 9:$

$$0.048\cos(\delta_n) \quad = \quad 0.048\cos(\delta_n) \tag{6.44}$$

$Part\ 10:$

$$-0.049|u|u = -0.0492|u|u \tag{6.45}$$

$Part\ 11:$

$$0.132\delta_n^2 \sin(\cos(\delta_n))\cos(\delta_n) \quad \approx \quad - \tag{6.46}$$

$Part\ 12:$

$$1.39 \cdot 10^{-9}\delta_t^2 u^2 \quad \approx \quad 1.58 \cdot 10^{-9}u^2\delta_t^2 \cos(\delta_n) \tag{6.47}$$

$Part\ 12:$

$$1.39 \cdot 10^{-9}\delta_t u^3 \quad \approx \quad - \tag{6.48}$$

$Part\ 13:$

$$4.61 \cdot 10^{-6}r\sin(\cos(\delta_n) \quad \approx \quad - \tag{6.49}$$

δ_n is in the range ±0.47 rad, this in turn places the term $\sin(\cos(\delta_n))$ in the range [0.77, 0.84], and for input 1 and 2 the range is [0.82, 0.84]. Hence, this term could then viewed as a constant.

In most of these parts, $\cos(\delta_n)$ is in the correct equation, but not in the one from the GP algorithm. Converging to something close like $\cos(\delta_n) = 1$ is far more likely, and does not affect the over all fitness of the model in a significant way, as most of these terms are quite small. The one part that actually consists of $\cos(\delta_n)$ on both sides is Part 9, which is the term with the largest parameter.

The parts that does not have a direct counter part is marked with −. These terms should not be in the model, but likely appear because other terms in the linear in parameter model was a great fit. An example of this is the term from (6.34), $1.3905 \cdot 10^{-9}(\delta_t^2 u^2 + \delta_t u^3)$. The $\delta_t^2 u^2$ is the term that makes the fit improve, but with the cost of adding $\delta_t u^3$. This explains why the parameters in some of the other parts are not correct, as the algorithm needs to find a trade-off between keeping the good parts, and weighting the poor ones with a number close to zero.

Input set 2 as training and input set 1 as validation:
When input 2 were used as training set and input 1 as validation, the algorithm did not find any solution. The algorithm overfitted to the train set. This is likely because of the lack of diversity in the input. This shows the importance of data gathered with a large variety of maneuvers, to contain the entire dynamics of the vehicle.

Input set 3 as training and input set 1 as validation:
With input 3 as training set and input 1 as validation, the model was close to finding a suitable model to describe the system. The best validation accuracy found were $MSE = 2.275e-07$. Input 3 is the simplest of the inputs, just a sine function. But the decent performance may be due to the fact that it contains a lot of different values in the input space.

6.4.2.2 Sideways acceleration, $\dot{v}$

The model that the GP algorithm seeks to find is shown in (6.50)

$$\begin{aligned}\dot{v} = {} & 0.048\sin(\delta_n) - 1.0 \cdot 10^{-3}\delta_t \sin(\delta_n) + 2.98 \cdot 10^{-6}\delta_t^2 \sin(\delta_n) - 2.56 \cdot 10^{-3} u \sin(\delta_n) \\ & + 5.67 \cdot 10^{-5} u\delta_t \sin(\delta_n) - 1.65 \cdot 10^{-7} u\delta_t^2 \sin(\delta_n) + 2.55 \cdot 10^{-5} u^2 \sin(\delta_n) \\ & - 5.31 \cdot 10^{-7} u^2 \delta_t \sin(\delta_n) + 1.58 \cdot 10^{-9} u^2 \delta_t^2 \sin(\delta_n) - ur - 0.045v - 0.4052|v|v \end{aligned} \tag{6.50}$$

The GP algorithm found (6.51), with input 1 as training set and input 2 as validation. Input 3 was set as a test set, and the model fitted equally well on this set. The threshold was reached at generation 49.

$$\begin{aligned}\dot{v} = {} & -0.00137\delta_n u(\delta_n + u) - 1.6820 \cdot 10^{-7}\delta_n(\delta_n(\delta_n + v) + \delta_n + u(\delta_t^2 + \cos(\delta_t)) + \cos(\delta_n) + |r| \\ & + |\sin(\delta_n)|) - 0.000837\delta_t \sin(\delta_n) + 1.9735 \cdot 10^{-9}\delta_t - 0.9999ru + 5.6403 \cdot 10^{-5} u \sin(\delta_n) * |\delta_t + u| - 0.405240v|v| \\ & - 0.0405v + 2.89830 \cdot 10^{-6}(\delta_t + u)(\delta_t + u^2)\sin(\delta_n) - 1.00476 \cdot 10^{-5}\cos(\delta_n) - 4.0313 \cdot 10^{-5}|\sin(r)| \end{aligned} \tag{6.51}$$

Rewriting (6.51) gives:

$$\begin{aligned}\dot{v} = &-ru - 0.405v|v| - 0.0405v + \delta_n\alpha + u(-0.0014\delta_n^2 - 1.69\cdot 10^{-7}\delta_n\cos(\delta_n)) + \delta_t(-8.37\cdot 10^{-4}\sin(\delta_n)\\ &+ 1.97\cdot 10^{-9}) + 5.93\cdot 10^{-5} - u^2(-0.0014\delta_n + 5.64\cdot 10^{-5}\sin(\delta_n)) + 2.89\cdot 10^{-6}u^2\delta_t\sin(\delta_n)\\ &+ 2.89\cdot 10^{6}\delta_t^2\sin(\delta_n) - 1.68\cdot 10^{-7}\delta_t^2 u - 4.03\cdot 10^{-5}|\sin(r)| - 1.0\cdot 10^{-5}\cos(\delta_n) + 2.89\cdot 10^{-6}u^3\sin(\delta_n)\end{aligned} \tag{6.52}$$

where :

$$\alpha = -1.68\cdot 10^{-7}(\delta_n^2 + \delta_n + \cos(\delta_n) + |sin(\delta_n)| + |r| + \delta_n v) \tag{6.53}$$

Equation (6.52) does not exactly match (6.50), but contains a lot of the components. The only significantly wrong term is $2.89\cdot 10^{-6}u^3\sin(\delta_n)$, which will worsen with a large value of u. Since input 1 and 2 does not contain u with a value larger than 5 this term does not cause the fitness to degrade. Creating an input with a larger u would probably make the algorithm settle for another model not containing terms like u^3

The other wrong terms can be explained by $sin(\theta) \approx \theta$ for small values of θ, or that they are dominated by larger terms.

6.4.2.3 Yaw Acceleration, $\dot{r}$

Equation (6.54) is the actual equation for the system with water jets as input.

$$\begin{aligned}\dot{r} = &-0.18\sin(\delta_n) - 3.82\cdot 10^{-3}\delta_t\sin(\delta_n) - 1.14\cdot 10^{-5}\delta_t^2\sin(\delta_n) + 9.78\cdot 10^{-3}u\sin(\delta_n)\\ &- 2.17\cdot 10^{-4}u\delta_t\sin(\delta_n) + 6.30\cdot 10^{-7}u\delta_t^2\sin(\delta_n) - 9.74\cdot 10^{-5}u^2\sin(\delta_n)\\ &+ 2.03\cdot 10^{-6}u^2\delta_t\sin(\delta_n) - 6.04\cdot 10^{-9}u^2\delta_t^2\sin(\delta_n) - 0.0612(v+r) - 0.1422|r|r\end{aligned} \tag{6.54}$$

With input 1 as test and input 2 as validation, the system converged to (6.55). This equation fits the data well, and contains several of the terms from the actual model.

$$\begin{aligned}\dot{r} = &\ 1.672\cdot 10^{-10}\delta_n^2\delta_t^2(\delta_n\delta_t + u + |v|) - 6.7006\cdot 10^{-13}\delta_n\delta_t^2 u(\delta_t * u + u)\\ &+ 0.000247\delta_n\delta_t u - 2.58\cdot 10^{-6}\delta_n\delta_t(\delta_t + |\delta_n|) + 0.000715\delta_n\delta_t\\ &- 0.0734\delta_n u - 0.142r|r| - 0.0612r - 0.0612v + 0.00457\sin(\delta_n)\end{aligned} \tag{6.55}$$

6.5 Capabilities of GP

In this chapter, GP programming has been shown to find the correct model when the actual model was relatively simple as for the model with force as input. The algorithm converged within a few generations, each time the algorithm was run. This shows promise to GP as a system identification approach.

The inputs sets for the model with force as input, though relatively simple, all proved to be enough to find the correct model. An advantage when creating this data set is that the individual forces can be chosen independently of each other. From this, the individual forces' impact on the system can be interpreted by the algorithm. This is not the case for the data sets created for the system identification of the model with actuator dynamics included.

Here, the forces in sway and moment in yaw can't be chosen independently of each other. This is likely one of the reasons the algorithm did not find the correct solution for this system.

Another reason that the GP algorithm did not find the correct solution for the model with actuator dynamics included, might be the complexity of the system. This model contains three times as many terms and two additional functions in the function set. There is possibly a limit to what is achievable by GP, as it did not find the exact model, even without noise in the data.

However, the performance is more than high enough for control and simulation purposes. The model including the actuator dynamics got a $MSE < 10^{-}8$ on the validation set. From this, it can be concluded the GP approach find a suitable model, given a data set that covers the operational space of the craft.

Chapter 7

GP on Data from Odin

In this chapter, data from Odin's sensors are used to find the dynamic model through GP. How the data used in this work looks like is explained and shown, along with what manoeuvres the data consists of. What preprocessing was performed on the data is presented, and discussed. The result from the GP algorithm is presented and analyzed, and compared to the standard model. Finally, a goodness of fit analysis of the models concludes the chapter. The notations used in this chapter is the same as defined in Chapter 6.

7.1 Data

The data were gathered in December 2018, using several different maneuvers. The data is plotted in Appendix F. These plots show that the vehicle is performing several zig-zag maneuvers at different surge velocities. Additionally, there are straight line steps in surge velocities, in different headings.

The data set was recorded in a single day, with the same environmental disturbances. As seen in the longitude-latitude plot in Appendix F, the vehicle is heading in different directions. This makes the effect of wind- and current disturbances negated. Furthermore, the sea state will be about the same for the entire data.

In Figure 7.1, forward velocity, u, is plotted along with a line that shows the chosen as the validation- and training set. From this plot it is clear that maneuvers with different magnitudes of u are executed. This can also be seen from Figure 7.2 where the RPM has several steps. From this insight, the data sets containing validation and training should be drawn from each of these steps, as the training and validation data should contain all the different dynamics of the vehicle at different velocities.

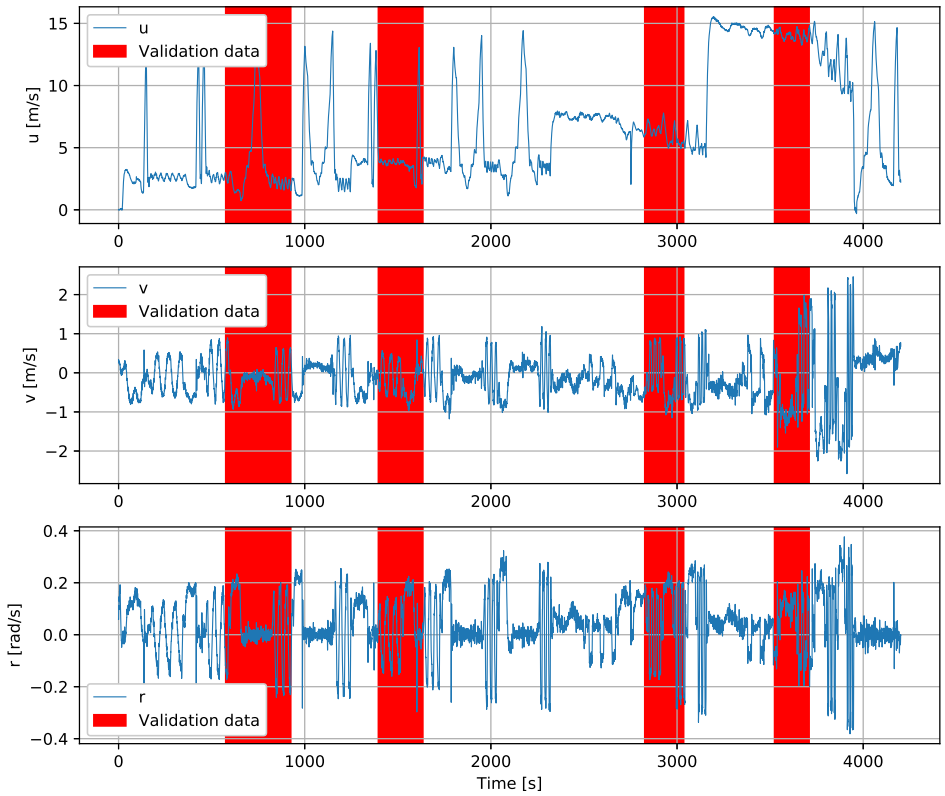


Figure 7.1: u, v and r for the preprocessed data. The red areas was chosen as validation and the remaining is the training data

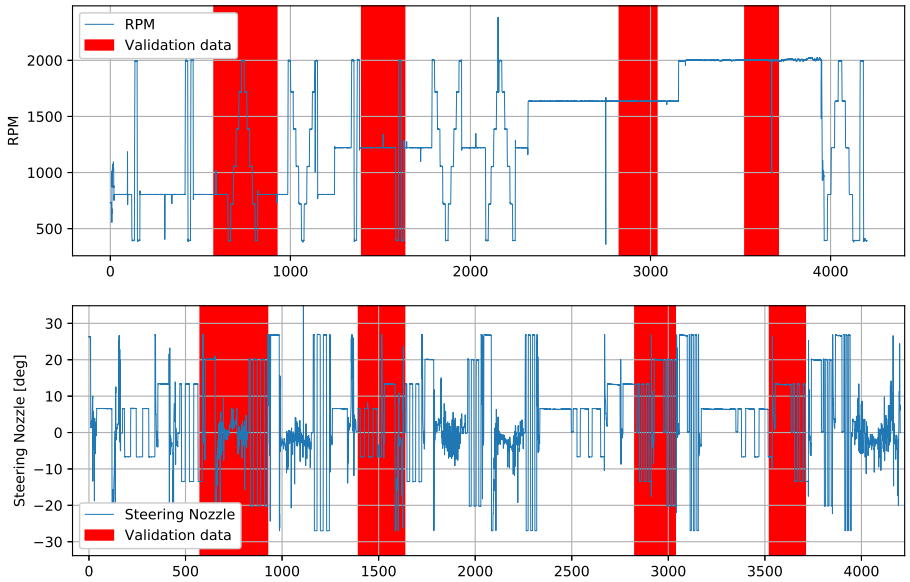


Figure 7.2: The input RPM and nozzle angle. The red areas was chosen as validation and the remaining is the training data

7.2 Model

As in Chapter 6, there are two ways defining the model that will be identified:

- 1. Finding a model with the actuator dynamics included. Thus, the inputs to the GP algorithm is RPM and nozzle angle.
- 2. Under the assumption that the water jet model described in 6 is sufficiently accurate, forces and moment can be calculated from the RPM and nozzle angle and used as inputs in the GP algorithm.

There are trade-offs with both approaches. The first is not relying on an accurate actuator model work. The drawback is that the model becomes more complex as it also contains the actuator dynamics. Method 2, should result in a simpler model. A simpler model is less prone to overfitting and might result in a better model, as the data set is recorded in a single day. Both these approaches are examined in this thesis and shown in Section 7.4.

A third approach would be to divide the model into 3 parts, one for each phase. In the displacement phase, the model could be the standard model or a new model. The model found with GP might have some resemblance to the standard model. In the two other phases, a new model would be required. However, the resulting models using this approach showed inferior results to method 1 and 2. Hence, this approach was discarded.

7.3 Preprocessing

Preprocessing is altercations of the data, as a step between raw data and the data the GP algorithm is training on. For this thesis, preprocessing consists of noise removal.

The autopilot on Odin is using Robot Operating System (ROS), and stores sensor data in ROS bags. In the ROS bags, surge and sway velocities, yaw rate, jet RPM and nozzle angle are stored. The acceleration data has to be extracted from the velocities with a numerical differentiation method. The method used in this thesis is shown in (7.1), and is known as the Euler approximation method.

$$\dot{y}_n = \frac{y_{n+1} - y_n}{\Delta t} \tag{7.1}$$

7.3.1 Interpolation and Filtering

Filtering of the data is necessary as it contains noise. Odin has a built-in Kalman filter to reduce the noise, but when the derivatives are derived from the states, the data needs to be smoothed out. A plot showing what the derivatives look like when no filtering is applied on the data is shown in Figure 7.3

A low-pass (LP) filter is a good choice, as it removes high-frequency parts of the signal, and retains the low-frequency parts. The time steps between samples are not constant, thus, in order to make the LP filter work, the data were first interpolated to a sample rate. A sample rate of 20 points/s was chosen and implemented using the function *numpy.interp*.

A second order Butterworth filter was chosen as the LF filter and created using *scipy.signal.butter*. Then the filter was applied on the data using *scipy.signal.filtfilt* to filter forward and backwards. Filtering both ways is common in offline signal processing and results in a zero-phase filtering. Zero-phase filtering is used to preserve the features in a time waveform where they occur in the unfiltered signal [13].

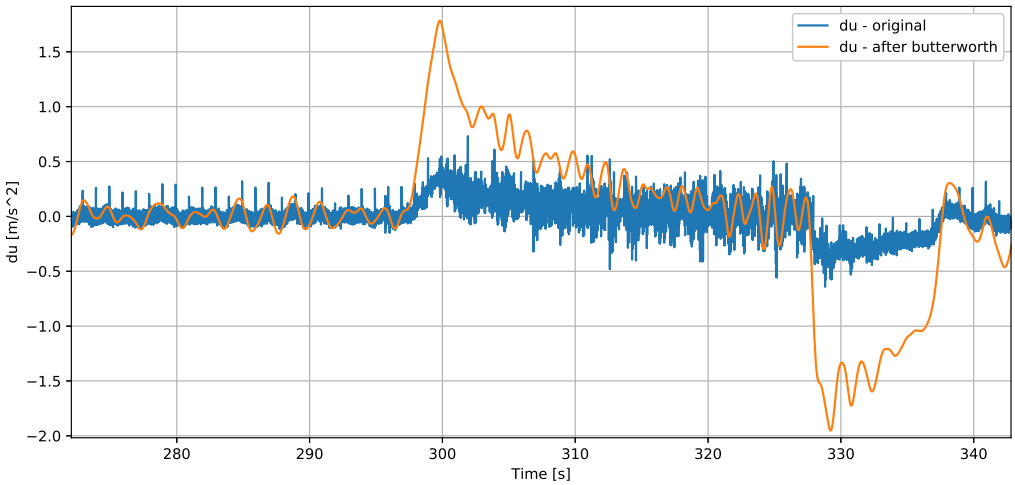


Figure 7.3: $\dot{u}$ when u is filtered and when the raw data, is used to find the acceleration.

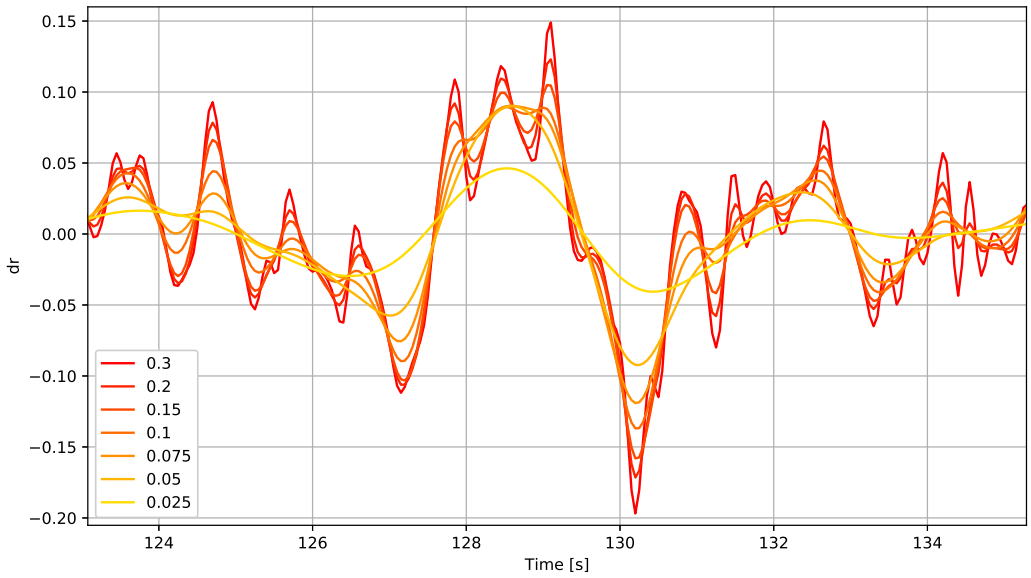


Figure 7.4: Different cutoff frequencies for the Butterworth filter, and the impact this has on $\dot{r}$. These are normalized frequencies from 0 to 1

The implemented filter is a digital filter, and hence the cutoff frequencies are Normalized Frequency (NF), this measurement of NF is equivalent to cycles/sample, rather than analogue filters with hertz or rad/s. Hence, it's not straight forward to give a physical meaning to the chosen frequency.

The cutoff frequency of the Butterworth filter will be a trade-off between removing the noise and keeping the dynamics of the USV in the data. Different cutoff frequencies are shown in Figure 7.4. From this figure, it's clear that the lowest cutoff frequency, 0.025, filters away more than is wanted, as the amplitude of the signal is significantly lower than the actual signal. From Figure 7.5 it's clear that the signal 'reacts' before an input is applied, for the lower cutoff frequencies.

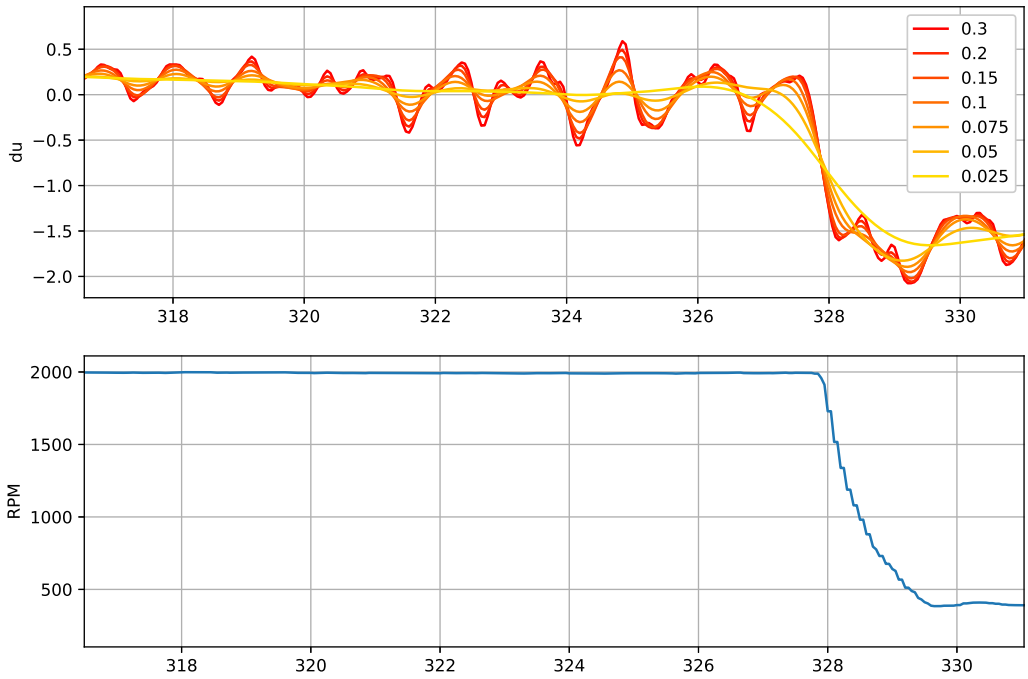


Figure 7.5: Different cutoff frequencies for $\dot{u}$ along with the jet RPM.

While the highest frequency introduces a lot of noise in the data, this can most likely be attributed to the disturbances from wave and/or wind, as these altercations in the signal occur without any inputs to the system.

The genetic programming algorithm is shown to work well with noise up to a certain point, so disturbance generated altercation in the signal is not a major issue. There is more important to conserve the result of the dynamics of the vehicle in the data. Hence, the chosen normalized cutoff frequency for the Butterworth filter is chosen to 0.1 as this retains most of the dynamics in the data while removing some of the noise.

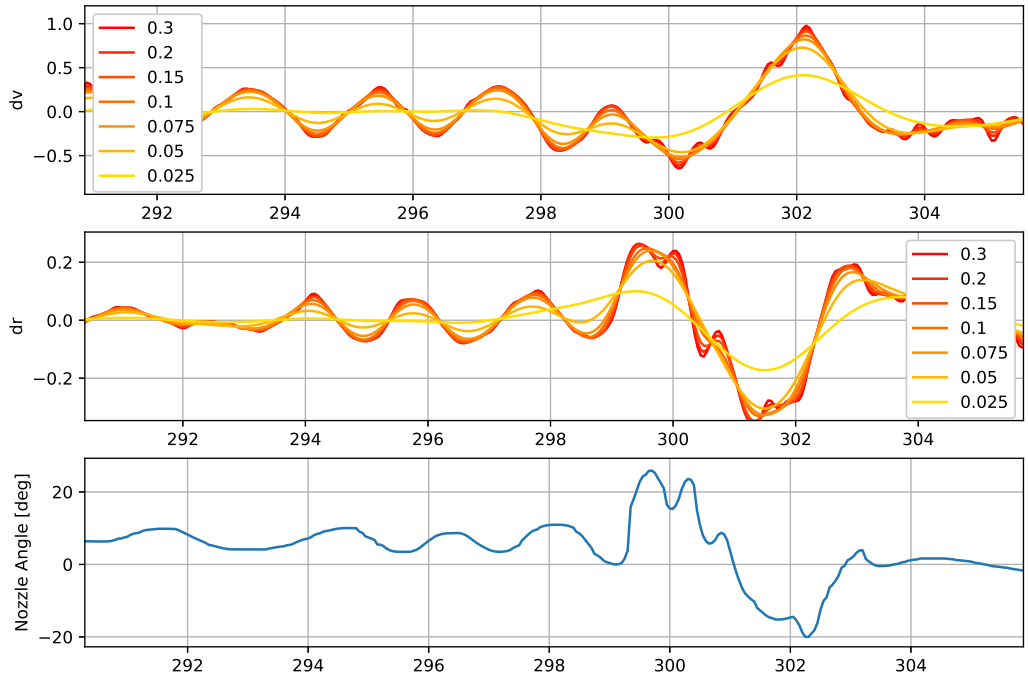


Figure 7.6: Different cutoff frequencies for $\dot{v}$ and $\dot{r}$ along with the nozzle angle

7.4 Results

In this section the genetic programming approach to identifying a dynamic model, is used on the data shown in Section 7.1 and in Appendix F. The models found using GP is listed and analyzed. The model performance is shown in the subsequent section, 7.5, where they are compared to the standard model. The last section contains a goodness of fit analysis.

7.4.1 Model with Control Inputs

This section presents the resulting equations from the GP algorithm, for a single model for the whole operational space of the USV. The inputs to the system, in this section, will be the jet engine RPM δ_t and the nozzle angle δ_n. The equations that best fit the data, while conserving the relative simplicity, are listed in (7.2) - (7.4). Table 7.1 shows the hyperparameters and evolutionary strategies used in this section.

$$\begin{aligned}\dot{u} = &- 6.401 \cdot 10^{-7}\delta_t^2|r| - 1.751 \cdot 10^{-7}\delta_t^2|v| + 9.536 \cdot 10^{-14}\delta_t^3|\delta_t| - 0.000289\delta_t u|r| + 5.812 \cdot 10^{-5}\delta_t u|v| + \\ &1.623 \cdot 10^{-8}\delta_t u^2|\delta_t| + 1.675 \cdot 10^{-7}\delta_t^2 u|r| - 1.772 \cdot 10^{-8}\delta_t^2 u|v| + 1.899 \cdot 10^{-14}\delta_t^3 u|\delta_t| - 7.242 \cdot 10^{-12}\delta_t^2 u^2|\delta_t| \\ &- 2.386 \cdot 10^{-11}\delta_t^2 u|\delta_t| - 0.155 * u - 0.000137\delta_t u + 0.000856\delta_t \end{aligned} \tag{7.2}$$

Hyperparameters and Evolutionary Strategies	Type / Value
Operators	+, *, abs, sine, cosine
Ephemeral Constant	None
Tournament size	5
Max Depth	7
Fitness Evaluation	LS with MSE
Population	5000
Prob. Mutation	0.3
Prob. Crossover	0.5
Mutation Method	mutUniform
Crossover Method	cxOnePoint, with min = 0 max = 2
Generations	30 (or to the point where the validation result stopped improving)

Table 7.1: The hyperparameters used in GP for model with actuator dynamic included.

$$\begin{aligned}\dot{v} = &- 0.120\delta_n u - 0.0425\delta_n v + 2.696 \cdot 10^{-6}\delta_t u - 0.000295\delta_t v - 0.207ru - 0.00694rv - 0.00295u|v| \\ &+ 0.0188v|v| \end{aligned} \tag{7.3}$$

$$\begin{aligned}\dot{r} = &0.00512\delta_n^3 + 0.00561v^3 + 1.005 \cdot 10^{-7}\delta_n\delta_t^2 + 0.00160\delta_n^2 u - 0.00145uv^2 + 7.504 \cdot 10^{-8}\delta_t^2 v \\ &- 6.876 \cdot 10^{-5}\delta_n^2 * \delta_t + 9.290 \cdot 10^{-6}\delta_t v^2 + 0.112\delta_n v^2 + 0.139\delta_n^2 v + 3.122 \cdot 10^{-6}\delta_n\delta_t u - 6.894 \cdot 10^{-6}\delta_t uv \\ &+ 0.00697\delta_n uv - 3.744 \cdot 10^{-5}\delta_n\delta_t v \end{aligned} \tag{7.4}$$

Some of the solutions provided by the GP algorithm had a better fit to the validation and training data, however, these have only slightly better fit, and higher complexity (more terms in the equation). This is the case for $\dot{r}$ where the equation with the best validation score was a mean square error of 0.001567, while the equation presented above had a validation score of 0.0016604.

These larger equations were not chosen, even though they fit better, because the equation's ability to generalize is key to retain. The more complex an equation becomes, the more likely it is that it overfits to the data. As previously explained the data were gathered in one day with similar environmental disturbances, and it's highly likely that the more complex equations try to fit this, and would be less precise with other data.

7.4.1.1 Model Analysis

To better understand why this model fits well to the data, the individual parts of the equations were plotted separately. This is shown in the Appendix B.1. It's not straight forward to understand why these equations were found to fit the data, but some insights will be examined further.

The first thing to note about these equations is that, despite having the trigonometric functions sine and cosine in the function set, the equations with the best fitness does not contain either. This might be because the angle δ_n, that would make sense to have as inputs to these functions, have a low minimum and maximum value. And would result in a about the same as just using δ_n for sine, and about a constant of 1 for cosine.

Forward Acceleration, $\dot{u}$

Most expressions that $\dot{u}$ is made up of contains δ_t. This makes sense as the surge acceleration is highly dependent on the jet engine RPM. This equation also has a lot of small parameters, which is natural as δ_t contains large values.

When the variables v and r appears, it is the absolute values of these. This is possibly because it does not matter what sign these have, as long as they have a non-zero value, they will most likely cause a reduction in surge acceleration.

This model resembles the model for the water jets, or actuator model as shown in Section 6.1.1, as it has several terms like $\delta_t^n u^k$, for n and k in the range $[1, 2, 3, 4]$.

Sideways Acceleration, $\dot{v}$

The equations for sway acceleration is the simplest of the three. The model is simple because the best equation that was found, was from generation 2, where the initial population is considered generation 0. The more generations, the better individuals tend to be larger. Hence, the is probably a better equation for $\dot{v}$, but for this thesis, this will suffice.

This equation contains some of the elements from the standard model, like $|v|v$ and ru. The variable δ_t is used in some of the terms in the equation. This might be to account for the higher acceleration in sway when the vehicle is at higher surge velocities.

Yaw Acceleration, $\dot{r}$

The yaw acceleration is highly dependent on the jet's nozzle angle δ_n. This is apparent from the equation, as several of the terms contain this variable.

As for the equation for surge acceleration, the equation for yaw acceleration also contains elements resembling the actuator model shown in Section 6.1.1.

The yaw rate, r, is not present in this equation, as one might assume it would be. The explanation for this might be that there is enough information in the other variables, and hence make it obsolete. This model shows accurate performance on the data and should generalize to new data well, as the approach used to validate should be sufficient.

7.4.2 Model with Force inputs

Under the assumption that the actuator model shown in Chapter 6 is sufficiently accurate, the forces from the actuators can be found from the control inputs in the data set. As seen in Chapter 6, this can greatly reduce the model complexity, and the trigonometric functions *sine* and *cosine* can be excluded from the primitive set.

The training and validation sets remain the same as in the model with actuator dynamics included. The equations that had the best fitness is shown in (7.5) - (7.7). The hyperparameters and evolutionary strategies used in this section is listed in Table 7.2.

$$\begin{aligned}\dot{u} = {} & 0.000265 * \tau_x - 5.104 \cdot 10^{-7}\tau_x u^3 - 2.774 \cdot 10^{-6}|\tau_z v| + 2.649 \cdot 10^{-9}\tau_x|\tau_x| - 0.0001052 rv|\tau_x| - 0.0370u^2 \\ & + 0.00208u^3 - 1.722 \cdot 10^{-5}\tau_x r - 9.437 \cdot 10^{-5}\tau_x u + 1.348 \cdot 10^{-5}\tau_x u^2 + 0.0478ru + 2.2407rv \\ & + 0.00947uv - 0.000720u^2 v \end{aligned} \tag{7.5}$$

Hyperparameters and Evolutionary Strategies	Type / Value
Operators	+, *, abs
Ephemeral Constant	None
Tournament size	5
Max Depth	6
Fitness Evaluation	LS with MSE
Population	5000
Prob. Mutation	0.3
Prob. Crossover	0.5
Mutation Method	mutUniform
Crossover Method	cxOnePoint, with min = 0 max = 2
Generations	30 (or to the point where the validation result stopped improving)

Table 7.2: The hyperparameters used in GP for model with actuator dynamic included.

$$\begin{aligned}\dot{v} = {} & 0.0001772u^2 - 2.982 \cdot 10^{-7}\tau_x u - 5.111 \cdot 10^{-5}\tau_x v + 4.192 \cdot 10^{-6}\tau_z u + 2.131 \cdot 10^{-7}\tau_z v \\ & - 0.2079ru + 0.2776rv - 0.008344uv + 5.113 \cdot 10^{-6}u|\tau_y| - 1.061 \cdot 10^{-5}v|\tau_y|\end{aligned} \tag{7.6}$$

$$\begin{aligned}\dot{r} = {} & 1.084 \cdot 10^{-5}\tau_x v - 5.190 \cdot 10^{-9}\tau_x u^2 - 5.619 \cdot 10^{-6}\tau_y v - 2.186 \cdot 10^{-8}\tau_y u^2 - 4.100 \cdot 10^{-9}\tau_x^2 r \\ & + 6.760 \cdot 10^{-12}\tau_x^2 u - 1.464 \cdot 10^{-9}\tau_x\tau_y r + 3.040 \cdot 10^{-10}\tau_x\tau_y u + 1.907 \cdot 10^{-6}\tau_x ru - 3.853 \cdot 10^{-6}\tau_y ru \\ & + 2.526 \cdot 10^{-5}\tau_y\end{aligned} \tag{7.7}$$

7.4.2.1 Model Analysis

The models listed in the previous section is as complex as the models with actuator dynamics. This is due to the max depth constraint allowing the models to evolve to a depth of 6. When tested with a max depth of less than 6, the accuracy was greatly reduced. Hence, this is actually a more complex model than the one with actuator dynamics included.

The different parts of these equations are shown in Appendix B. These plots look similar to the ones for the model with actuator dynamics. With some of the terms for lower surge velocities, others for higher. While some looks to be there to cancel out terms for different velocities.

7.5 Comparison to the Standard Model

The standard model for marine vehicles are shown in chapter 6 and is what the new model should be compared to, at low surge velocities, as this model is only valid for the displacement phase.

7.5.1 Optimize the Standard Model

When the 3-DOF vehicle model used in chapter 6 was tested on real data, the results were poor. This is shown in Figure 7.7, and it's clear that the model parameters needs tuning. This figure shows the performance of the model in all three phases and shows an overall poor accuracy.

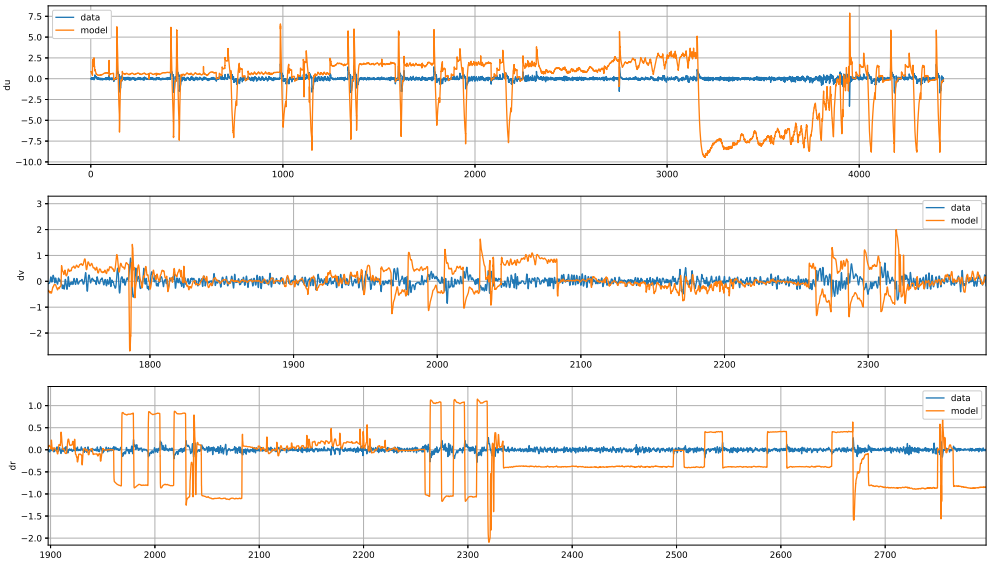


Figure 7.7: $\dot{u}$, $\dot{v}$ and $\dot{r}$. Displays a subsection of the entire data to show the performance better for $\dot{v}$ and $\dot{r}$

Assuming that the actuator model, the transformation between motor RPM and nozzle angle to force, is good enough, the parameters of the standard model can be found through non-linear optimization. There is no guarantee that the found parameters are globally optimal, however this approach should considerably increase the accuracy of the model.

In the library *Scipy.optimize* there is several optimization techniques. This implementation uses the function *least_squares*, which solves a non-linear least squares problem. The method uses the Trust Region Reflective algorithm to minimize the cost function. In this case the cost function is the squared error between the accelerations in the data and from the model as shown in (7.8). $cost \in \mathbb{R}^n$, where n is the length of the vectors $\dot{u}$, $\dot{v}$ and $\dot{r}$. The signs $-$ and $+$ and $|.|^2$, denotes element-wise operators.

$$cost = (\dot{\mathbf{u}} - \hat{\dot{\mathbf{u}}})^2 + (\dot{\mathbf{v}} - \hat{\dot{\mathbf{v}}})^2 + (\dot{\mathbf{r}} - \hat{\dot{\mathbf{r}}})^2 \tag{7.8}$$

After some runs, the non-linear least squares algorithm seemed to focus on minimizing $\dot{u}$ and $\dot{v}$, and gave poor result on $\dot{r}$. This might be because $\dot{r}$ has the lowest mean values, and errors from the data would be less significant. To make the algorithm focus more on $\dot{r}$ the cost function were altered to weight the squared error 'cost' more for $\dot{r}$. The altered cost function is shown in 7.9, where W_n is the weights. Setting W_2 higher than the others makes the algorithm focus more on minimizing this part of the cost function.

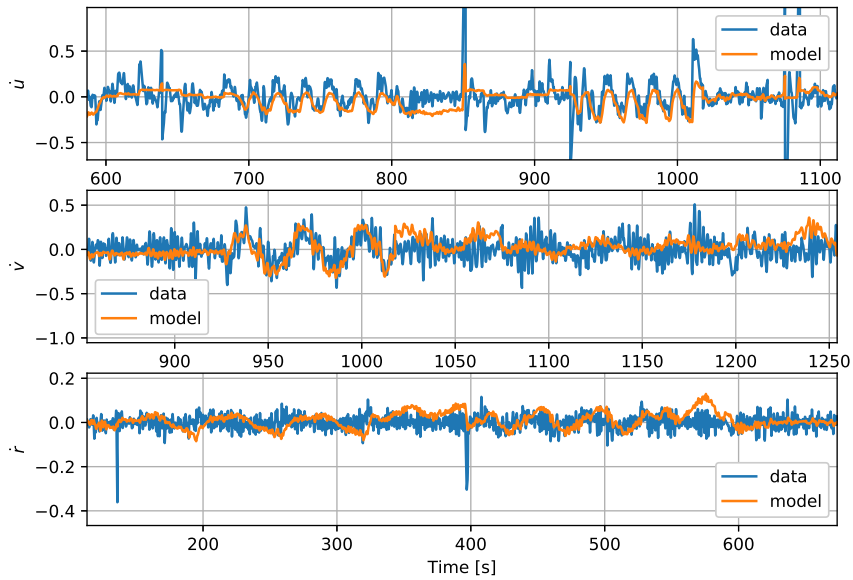


Figure 7.8: $\dot{u}$, $\dot{r}$ and $\dot{r}$ when the vehicle is in the displacement phase.

$$cost_2 = W_0(\dot{\mathbf{u}} - \hat{\dot{\mathbf{u}}})^2 + W_1(\dot{\mathbf{v}} - \hat{\dot{\mathbf{v}}})^2 + W_2(\dot{\mathbf{r}} - \hat{\dot{\mathbf{r}}})^2 \tag{7.9}$$

This is one way of dealing with skewness in the data. Another more elegant solution would be to normalize the data to approximately the same scale. This implementation would not require tuning. However, as the weighted cost function is the simplest, and works well, this is the chosen implementation for this problem. The weights were set to, $W_0 = W_1 = 1$ and $W_2 = 10$, as this proved to be a good ratio.

When finding parameters that are not overfitted to the data, the data set needs to be divided into training and validation sets as discussed in Section 6.2. The model is optimized for the displacement phase. This model structure is not designed to fit the other phases, and optimizing the parameters in the model for these gave poor results.

There are a total of 14 parameters to be found in the model. The only known parameter is the mass of the vehicle, which is about 5000 kg. The parameters to be estimated, and the complete structure of the model, are listed in Appendix C.2. The optimization of the parameters converged to a local minima for most of the runs, the parameters that best fit the data are listed in Appendix C. The standard model for the displacement phase is shown in Figure 7.8.

Most of the parameters are increased quite a bit. This might be a result of the performance increased when some of the parameters increased, and as a result, the others needed to compensate for this increase by having larger values. The model parameters should be seen in connection with each other, but the connection is quite complex. The final parameters do not necessarily make physical sense after the optimization.

These new model parameter probably does not give credit to the standard model as this is not the best way to find the parameters of the model. The model is not meant to found by data-driven/ optimization techniques, but rather by physical properties. Another limitation of the implemented standard model is that it does input all the forces that it should, as it only considers the actuator inputs. The data used in this thesis were logged when there were some environmental disturbances, such as wind and wave. This might be one of the main reason for the lack of

accuracy in the standard model, and that the new models outperform it.

7.5.2 Comparing the Models Visually

In this section, the models found by GP, and the standard model will be compared to each other, and the ground truth. There are a lot of data to compare, so only a representative subset of the models' performance is displayed.

7.5.2.1 Displacement Phase

In the displacement phase, the standard model should perform similarly to the new model found by GP. However, as discussed in Section 7.5.1, this implementation has some flaws. Hence, the new models from the GP algorithm outperform the standard model in this phase as well. Comparisons of the models are shown in Figure 7.9 - 7.11.

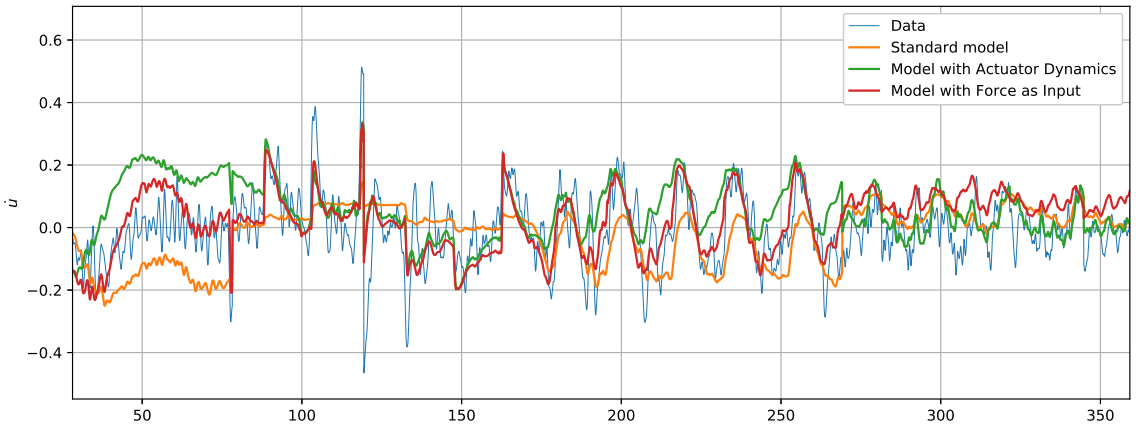


Figure 7.9: Validation data in the displacement phase. Where three different models are shown together with the data (GT)

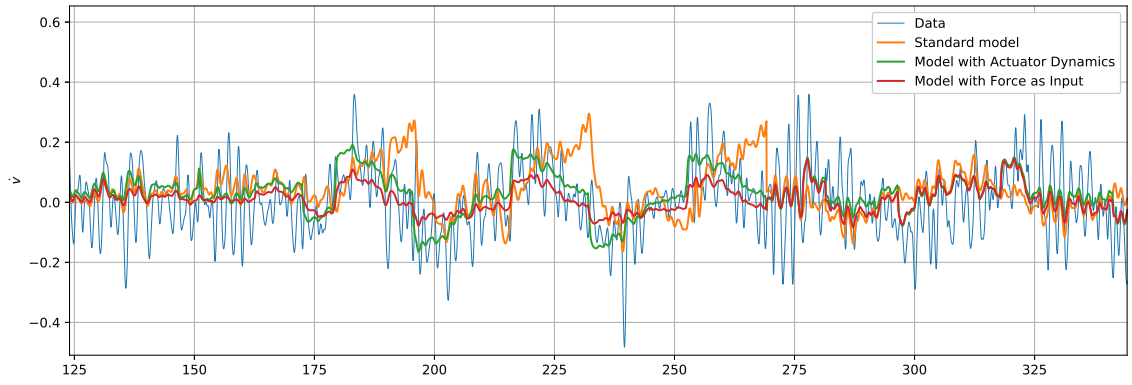


Figure 7.10: Validation data in the displacement phase. Where three different models are shown together with the data (GT)

What the true acceleration are, are unknown, as it's derived from velocities and filtered. The data also contains a fair bit of noise, even after the prepossessing, from environmental disturbances. This makes it difficult to tell which the best performing model actually is.

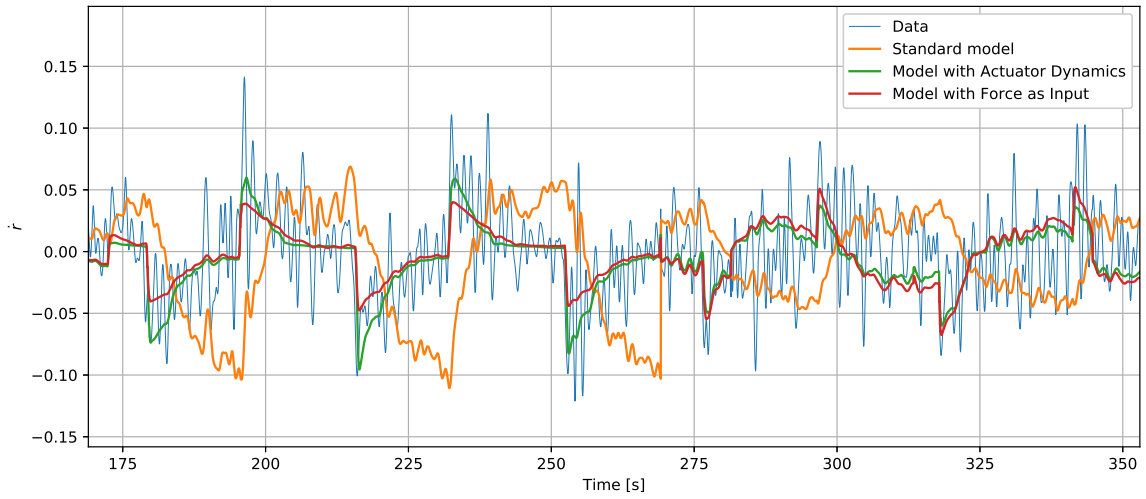


Figure 7.11: Validation data in the displacement phase. Where three different models are shown together with the data (GT)

From these plots, it would seem that the model with force as input performs better for surge acceleration, while the model with actuator dynamics performs better for yaw and sway acceleration. Both the new models perform quite equal.

7.5.2.2 All three Phases

In all three phases, the optimization algorithm did not find any suitable solutions for the parameters of the standard model. Hence, the following is a comparison between the two models found with GP. Figure 7.12 - 7.14 shows the performance of the two models, on the validation set. As for the displacement phase, the model with actuator dynamics visually seems to outperform the other model in the rest of the operational space of the vehicle.

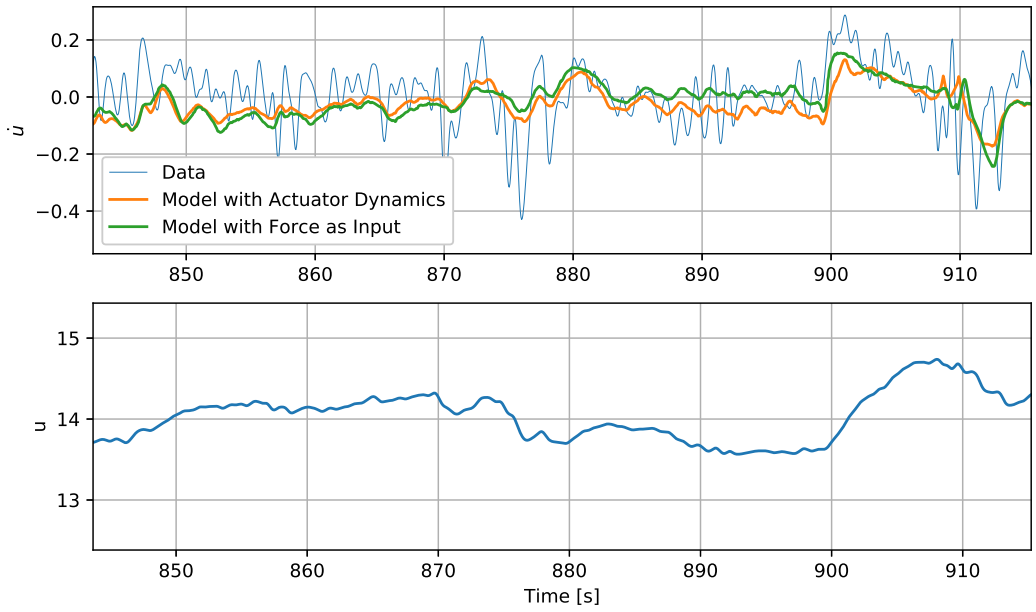


Figure 7.12: Validation data for $\dot{u}$ in the planing phase, together with surge speed u. Where three different models are shown together with the data (GT)

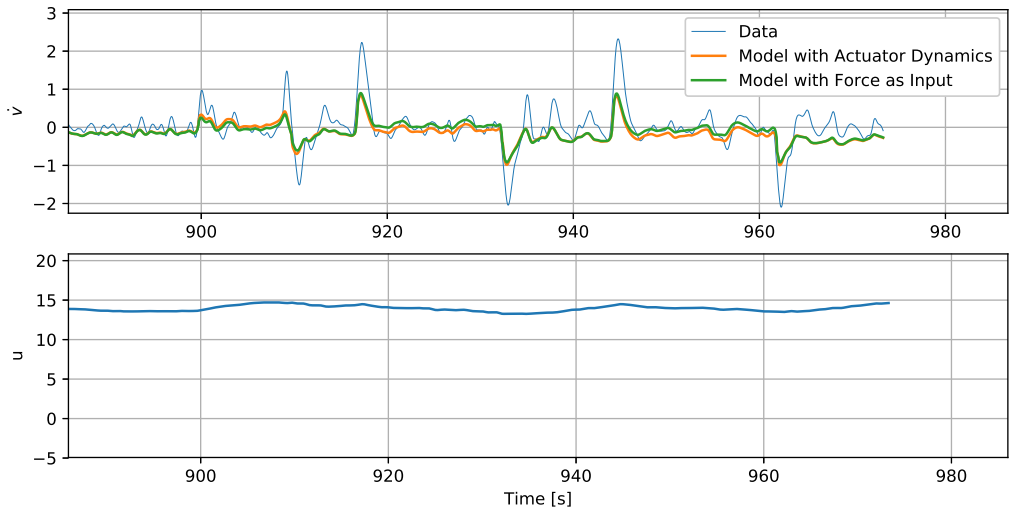


Figure 7.13: Validation data for $\dot{v}$ in the planing phase, together with surge speed u. Where three different models are shown together with the data (GT)

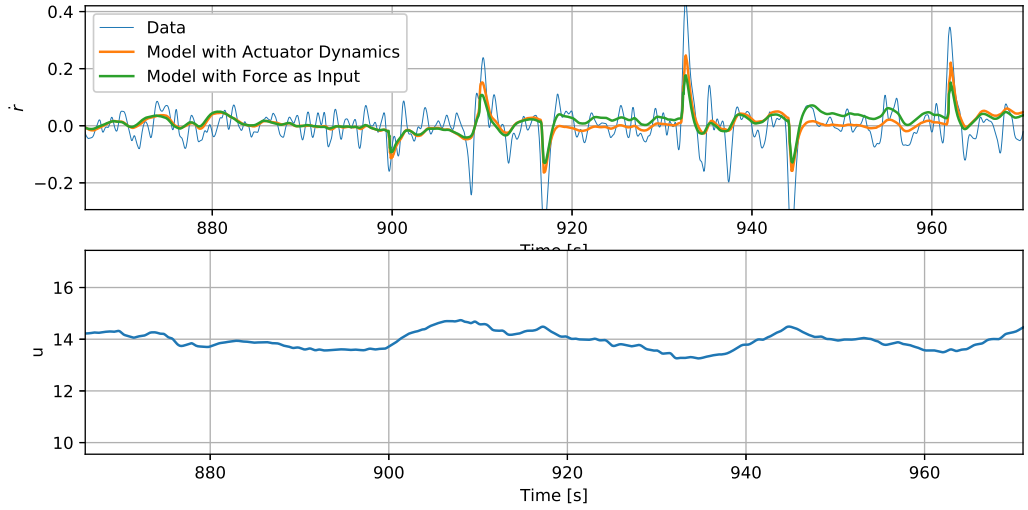


Figure 7.14: Validation data for $\dot{r}$ in the planing phase, together with surge speed u. Where three different models are shown together with the data (GT)

7.6 Goodness of Fit

As the amount of data in the validation set is too vast to plot, statistical measures of fitness or *goodness of fit*, is included in this section to quantify the fitness and easily compare the performance of the models. The data used in this section is the same for all the models, and preprocessed in the same way as when trained in GP and fitted for the standard model. The statistical measurements include; Mallow's Cp, Adjusted R^2 and Root Mean Squared Error (RMSE).

Mallows's C_p addresses the issue of overfitting, as it penalizes complex models more than simpler. For this measure, a low value suggests a good fit to the data. Adjusted R^2, or adjusted coefficient of determination, measures how well the data is replicated by the model, based on how much of the total variation in the data is explained by the model. For R^2, values close to 1 suggests a good fit, while lower values suggest poorer fit.

RMSE takes the root mean squared distances between the data and the predictions of the model. A low value indicates a good fit. How to calculate these statistical properties are shown in the appendix E.

Displacement Phase

		RMSE	Cp	Ad. R^2
$\dot{u}$	Force Input	0.107	0.0115	0.478
	Control Input	0.126	0.0159	0.275
	ST Model	0.143	0.0205	0.0645
$\dot{v}$	Force Input	0.108	0.0117	0.172
	Control Input	0.110	0.0121	0.137
	ST Model	0.1442	0.0208	-0.479
$\dot{r}$	Force Input	0.0316	0.00100	0.204
	Control Input	0.0309	0.000958	0.238
	ST Model	0.0511	0.00261	-1.081

Table 7.3: Statistics for the models in the displacement phase

All Phases

		RMSE	Cp	Ad. R^2
$\dot{u}$	Force Input	0.110	0.0121	0.726
	Control Input	0.120	0.0146	0.672
$\dot{v}$	Force Input	0.179	0.0322	0.370
	Control Input	0.185	0.0341	0.334
$\dot{r}$	Force Input	0.0391	0.00152	0.426
	Control Input	0.0390	0.00153	0.424

Table 7.4: Statistics for the models in all phases

The results are shown in Table 7.3 - 7.4. The model with force as input preforms better for $\dot{u}$ and slightly better for $\dot{v}$. While the model with actuator dynamics performs somewhat better for $\dot{r}$. The standard model, as presented in the previous section, preforms poorly overall.

From this statistical analysis of the fitness, it's clear that sway, v, is the hardest for the GP algorithm to find a suitable equations for. While on the other hand, surge is fits rather well to the data. The overall low Mallow's Cp scores suggests that the models are not overfitted to the data.

7.6.1 Confidence Interval with Bootstrapping

Another measure of the models' performance is the Confidence Interval (CI). The confidence intervals were fund trough bootstrapping. Bootstrapping for this task is a process where samples of the absolute of the residuals ($|y - \hat{y}|$) are chosen randomly, and may be chosen again each time a sample is selected. The mean of these residuals calculated for each iteration of the process, which the rule of thumb suggests 10000 repetitions is sufficient. The average mean and standard deviation for each repetition is used to create the CI. The confidence interval is 95%. This is further shown with code in Appendix E.

With overlapping CIs, there is insufficient evidence that the models are better than one another. But without overlapping CIs the conclusion is that, for this data set, that the model is significantly better, with a significance level of 5%. The result of this process is shown in Table 7.5, where the standard model is only tested for the displacement phase.

From these result, it's clear that the model with force as input outperforms the standard model, in all three degrees of freedom. The model with force also outperforms the model with actuator dynamics, for $\dot{u}$, for both the displacement phase and all phases. For the other acceleration and phases, the models can't with a 95% confidence be said to be better than one another.

Confidence Interval

		Displacement Phase			All Phases		
		mean	interval		mean	interval	
$\dot{u}$:	Force Input	0.0823	0.0809	0.0837	0.0839	0.0828	0.0848
	Control Inputs	0.0933	0.0916	0.0950	0.0901	0.0889	0.0912
	ST Model	0.0950	0.0928	0.0971	-	-	-
$\dot{v}$:	Force Input	0.0844	0.0830	0.0857	0.120	0.118	0.122
	Control Inputs	0.0859	0.0845	0.0873	0.125	0.123	0.127
	ST Model	0.114	0.112	0.115	-	-	-
$\dot{r}$:	Force Input	0.0244	0.0240	0.0248	0.0280	0.0276	0.0283
	Control Inputs	0.0238	0.0234	0.0242	0.0279	0.0275	0.0283
	ST Model	0.0391	0.0385	0.0398	-	-	-

Table 7.5: 95% confidence interval for the models.

Chapter 8

Discussion

8.1 Data and Validation

The data were gathered after the specialization project but before the work on this thesis. There are more data available, but this data was captured with the intent of data-driven system identification. This data contains several different maneuvers at different velocities, to capture the dynamics of the vehicle, and different headings to account for environmental disturbances. This is shown longitude-latitude plot in Appendix F.

The results shown in section 7.4, shows good results on the validation data. The use of validation and training sets should make the models unbiased with respect to all the data, but might be a bit skewed as all the data were captured in the same weather conditions. This was not investigated in this thesis, but might be worth looking into if the model is used in some capacity.

The observant reader might have noticed that in chapter 7, the data were divided into training and validation sets, but not a test set. This is due to the fact that the amount of data available was limited, and dividing it into three sets containing the entire operational space of the USV in each, would have made the training set sparse. To fully conclude the accuracy of the models listed in the chapter, the models should be tested on new data.

8.2 Preprocessing

The preprocessing of the data were done with both interpolation and low-pass filtration. This was necessary to find the acceleration in the 3 DOF. After this step, there was still noise in the data. This most likely wave induced noise and could be filtered away with a notch filter, as the noise looks to have a similar frequency for similar velocities.

However, the noise in the data has about zero mean. This can be seen in Figure 8.1 where no difference in the actuators, results in zero mean noise. This should not affect the GP algorithms performance, as shown in Appendix A. One reason to still use a notch filter would be that the algorithm might converge to a solution faster, as a good solution would have a lower MSE.

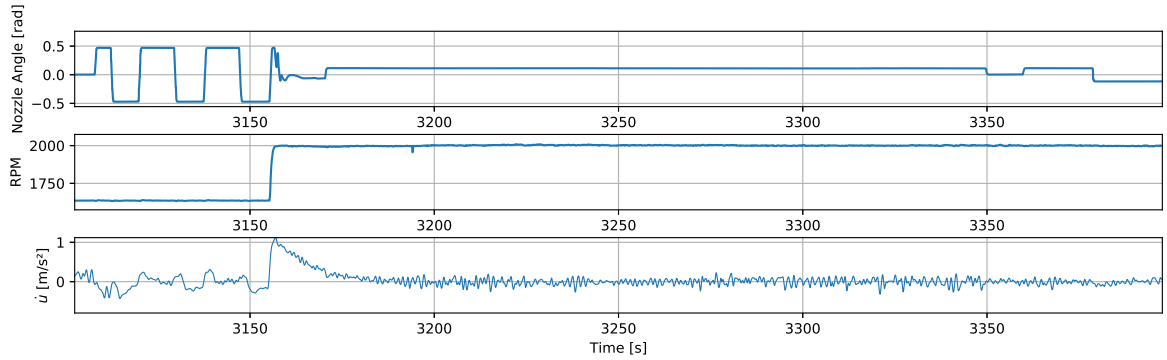


Figure 8.1: Zero mean noise.

8.3 Model Structure

The GP algorithms find the model structure that best fit the data, but there are still some choices to be made regarding the resulting structure. What functions to have in the primitive set is one of those. And equally important, what to input to the GP algorithm.

In the article by Eriksen and Breivik [6], the model states consisted of Speed over Ground, U, and Rate of Turn, r. While in this thesis, the model states were chosen to be surge-, sway- and yaw rate as in the article by D. Moreno-Salinas et al. [24]. Both approaches are viable, however, explicitly modeling three degrees of freedom makes it easier to give a physical interpretation of the result, and ensures that the resulting model does not contain any instabilities on yaw.

Both the articles by Eriksen and Breivik and D. Moreno-Salinas et al., the input to the system was the controller inputs, throttle and rudder [6][24]. The choice of inputs to the system was experimented with in this thesis and showed that force input gave slightly better results.

Wind, wave and current forces are included in the standard model [9], but not the implementation in this thesis, and may be one source of inaccuracy in the model. As the data from Odin also includes wind measurements, this could be taken into the model to further increase the accuracy. For this to be implemented, more data with different weather conditions would have to be captured/ used.

The hardest part of this problem is that the entire dynamics of the vehicle changes as is goes from pushing the water in front, to glide on top of the water surface when planing. A lot of the dynamics that fit the data for the lower velocities needs to accounted for in the higher velocities, and vice versa, because the model must hold for all the phases. To account for this the use of three models, one for each of the phases, could be found with GP. Intuitively this would be a good strategy, as the dynamics of the vehicle is similar within each phase.

8.4 Genetic Programming Approach to Finding Complex Models

Genetic programming has been tested on several different systems throughout this thesis. This approach has shown good results, both with and without noise present in the data. GP has shown that it's capable of finding the correct, or close to the correct model when the model is known. When the model is unknown, as for the data from Odin, the approach showed that it was able to find a model that fit the data well.

The positive part about this approach is that it is fairly easy to implement using the DEAP library for python, or the GP library for MATLAB. GP is an intuitive data-driven method to identify systems, with some prior knowledge

of natural selection, as this is where the idea originated from. This is not the case for other complex machine learning approaches like deep or shallow neural networks.

Another great advantage of the approach is the resulting models. The models are mathematical expressions, hence, with some knowledge of mathematics, the results are easily interpretable. Symbolic models also makes it easier to make a controller for the system, when non-linear controller strategies like feedback linearization are to be used. But then again, the models proposed in this thesis might not be accurate enough to cancel out terms reliably, as this would require a truly accurate model of the system.

The results in Section 6.4, where the model with the actuator dynamics was identified from the simulation data. The GP algorithm did not find the correct model, even with several generations, a large initial population and a handful of runs for all of the equations. For these tests, no noise or time delay was introduced in the data. GP algorithms of the form used in this thesis was unable to find the exact solution when the complexity became too high. However, the solutions found by the GP algorithm had high fitness on the data, and likely got stuck in a local minima.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In this thesis, the data-driven system identification approach, genetic programming, has been tested for simulated data, and actual data from Odin's sensors. This approach was implemented using the evolutionary algorithm library DEAP, while also introducing new functionalities that this problem required. Experiments with data containing noise were performed, and the GP algorithm proved to be robust with respect to noise.

A complex model of a 3-DOF vehicle was simulated and the data from these simulations were used to identify the system with GP. For this task, GP showed excellent results. Finally, the data from Odin's sensors were used to derive two different models, one with force as input to the system, and another with the actuator dynamics included. These models showed good results on the data and should be accurate enough for simulation and control purposes.

Throughout this thesis, GP has been showed to be a powerful system identification technique, while still resulting in an interpretable model.

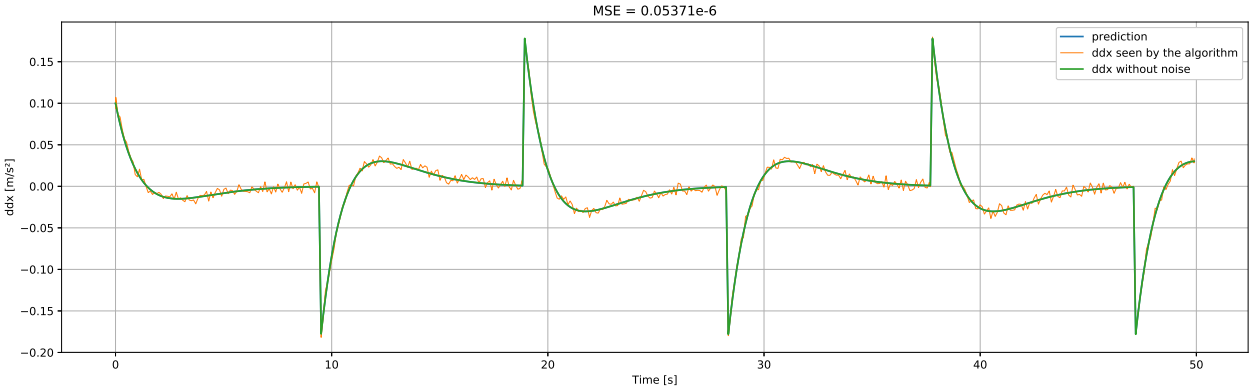
9.2 Future Work

There are several lines of research that can be further investigated:

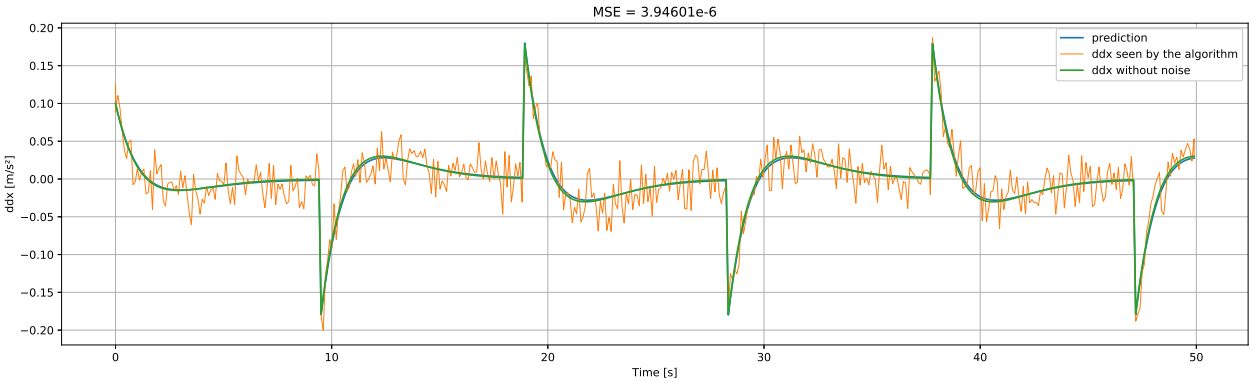
- Including environmental forces and moments as inputs to the system in the model.
- Further verification of the models by applying them to more data.
- Applying the model to feedback linearizing control regime
- Applying GP to a cascade of simpler models rather than a single model of high complexity
- Implementing GP as a tool to adaptively update an existing model during operations.

Appendices

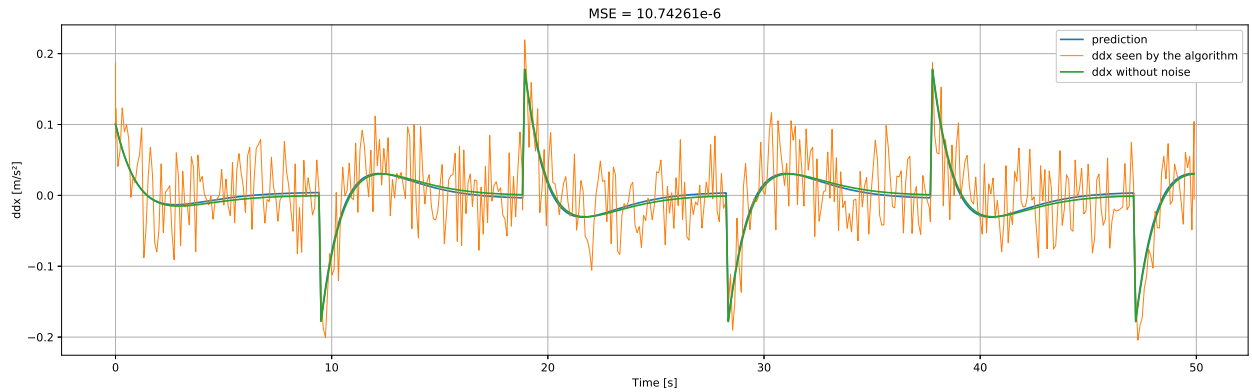
A Noise



(a) With added noise with amp ±0.01, the equation found was: $\ddot{x} = -1.2968\dot{x} + 0.0996\tau - 0.4974x$.

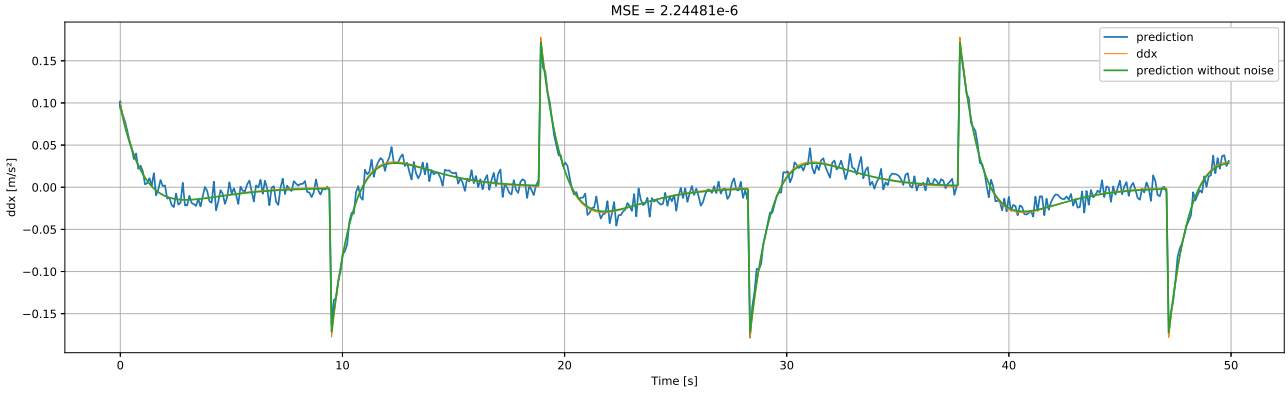


(b) With added noise with amp ±0.05, the equation found was: $\ddot{x} = -1.2687\dot{x} + 0.1000\tau - 0.5060x$

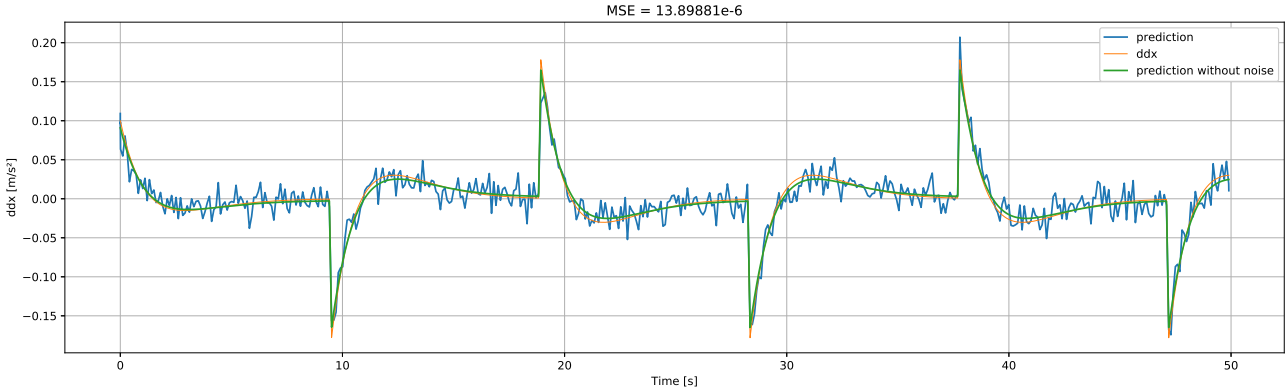


(c) With added noise with amp ±0.1, the equation found was: $\ddot{x} = -1.3340\dot{x} + 0.10139\tau - 0.48503x$

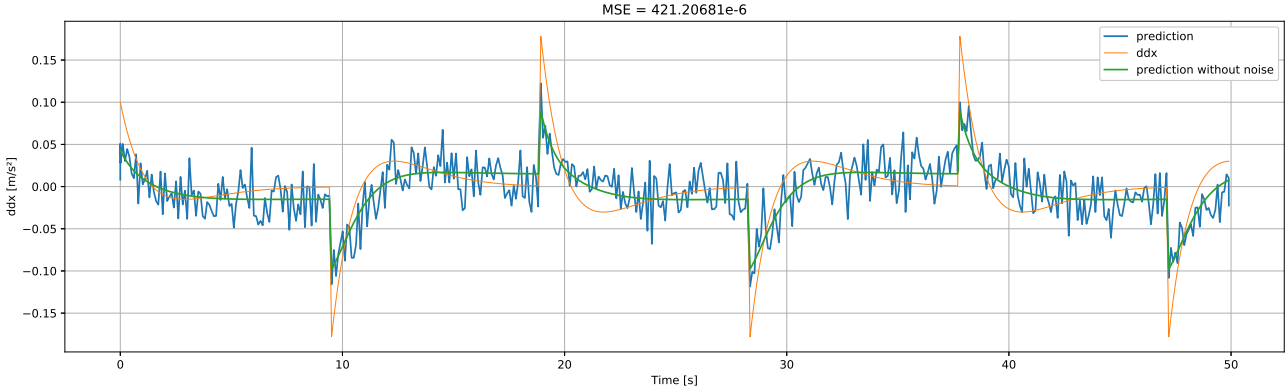
Results from adding noise on $\ddot{x}$, with different amplitudes. The calculated MSE is between the noise-free $\ddot{x}$ and the found solution



Results from adding noise on $\dot{x}, x$ and τ. The MSE is calculated with the noise free variables and the $\ddot{x}$. Noise Factor = 0.05, $\ddot{x} = -1.2370\dot{x} + 0.0957\tau - 0.4847x$



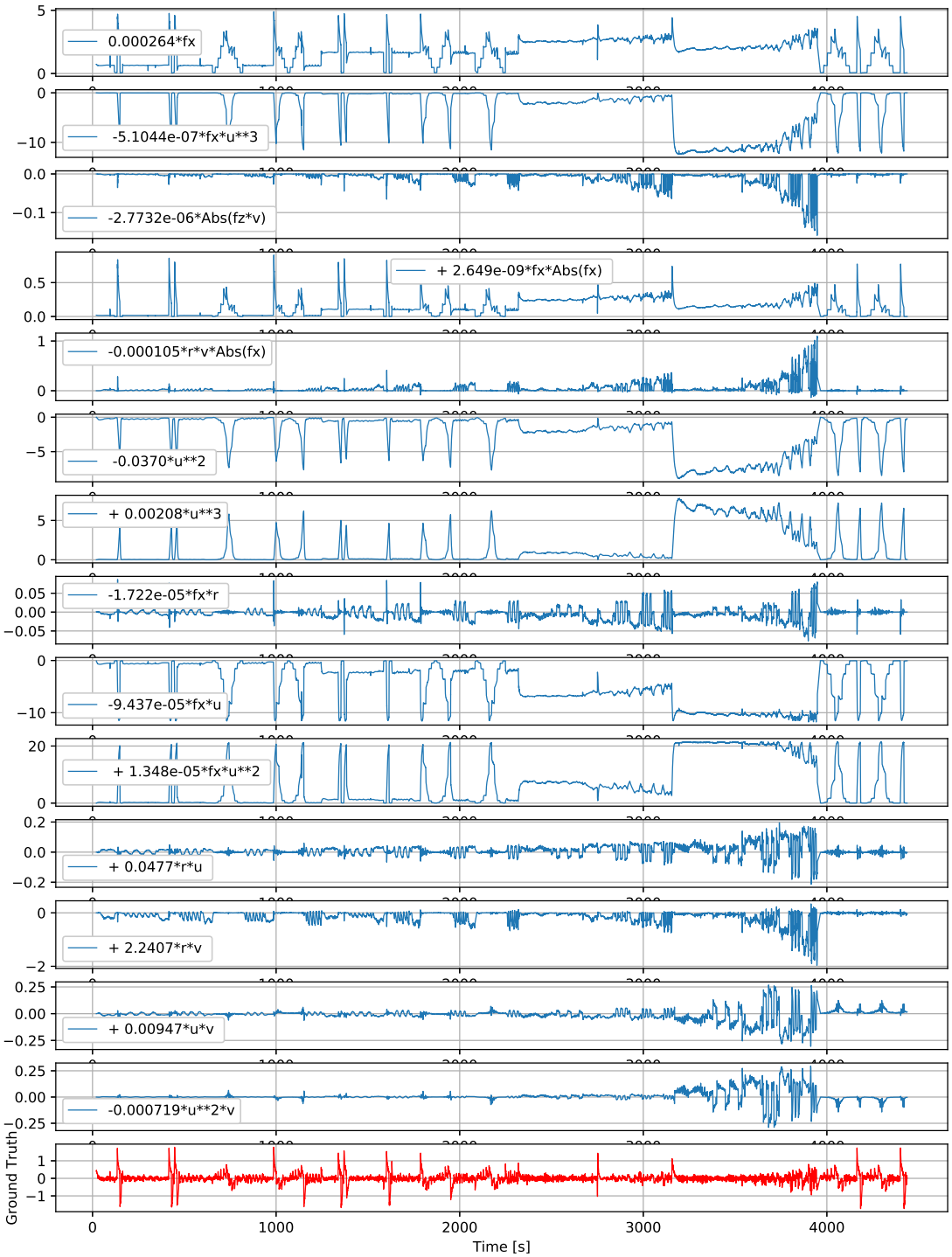
Results from adding noise on $\dot{x}, x$ and τ. The MSE is calculated with the noise free variables and the $\ddot{x}$. Noise Factor = 0.1, $\ddot{x} = 0.090888\tau + 1.1311\dot{x} + 0.4619x$



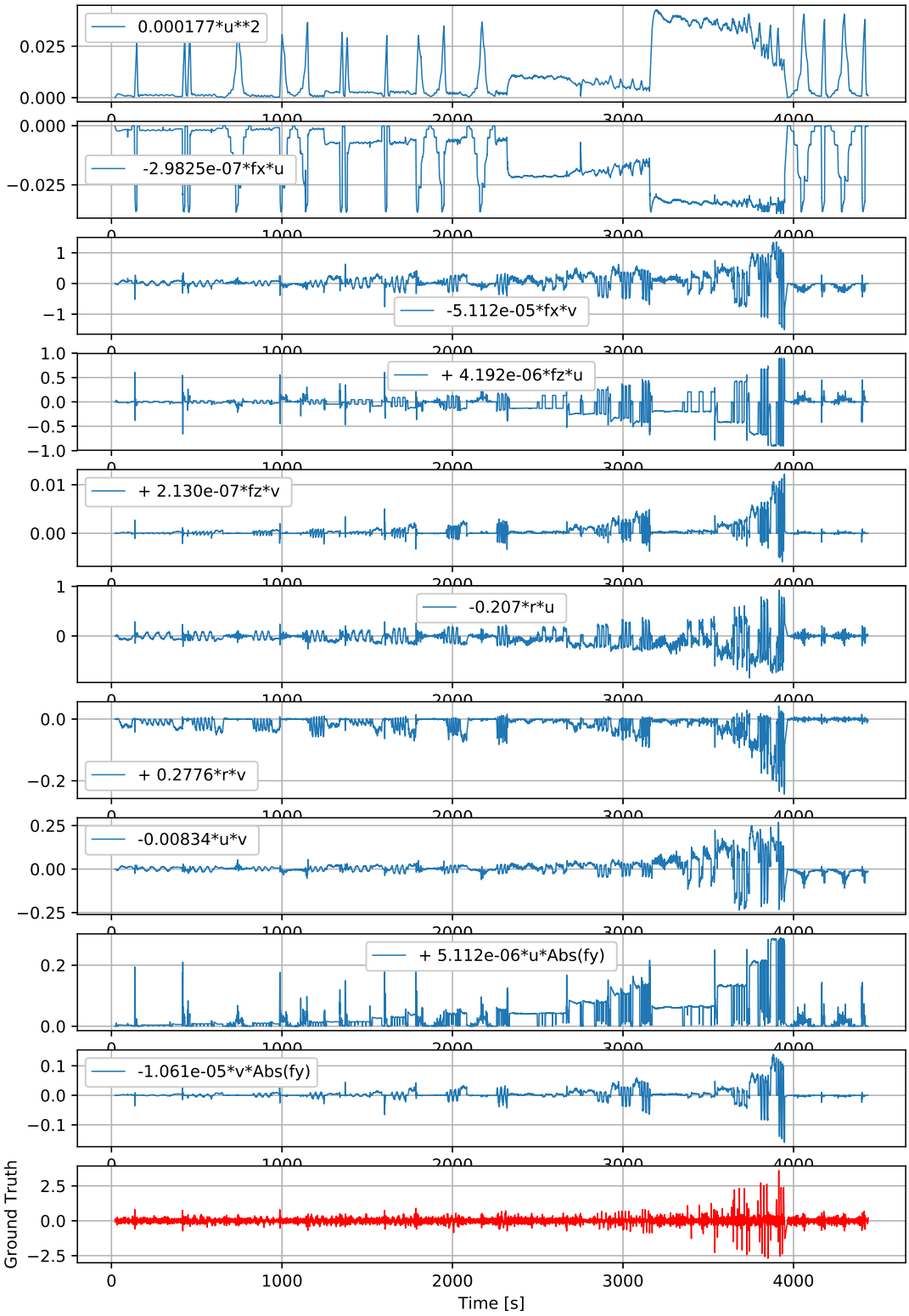
Results from adding noise on $\dot{x}, x$ and τ. The MSE is calculated with the noise free variables and the $\ddot{x}$. Noise Factor = 0.3, $\ddot{x} = (1.0742\dot{x} - 0.4060) * (1.0742\dot{x} - 0.1061\tau + 0.7137 * x)$

B Part-Wise plots

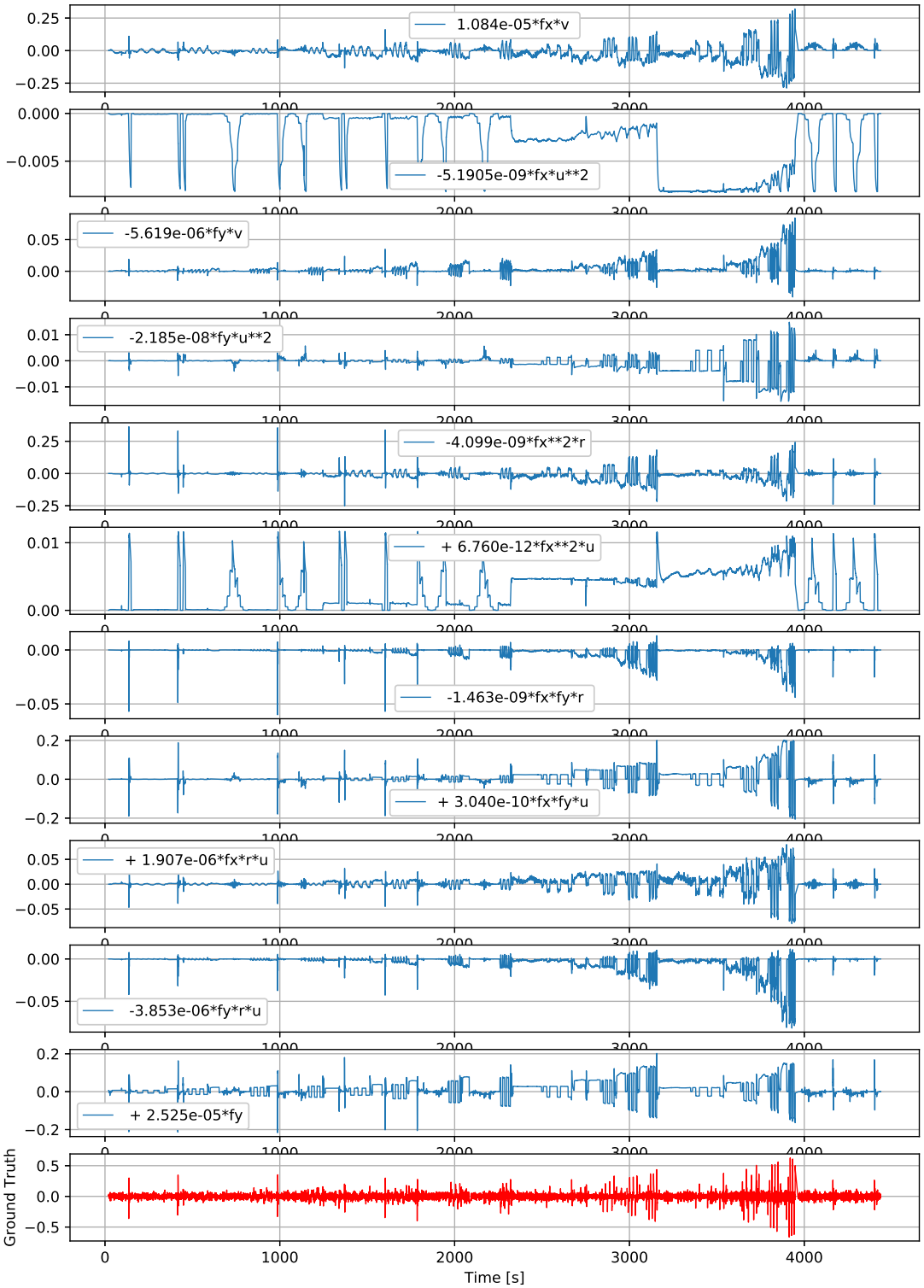
B.1 Part-Wise Model Plots with Force as Inputs



$\dot{u}$ plotted part-wise, for the model with force as inputs to the system

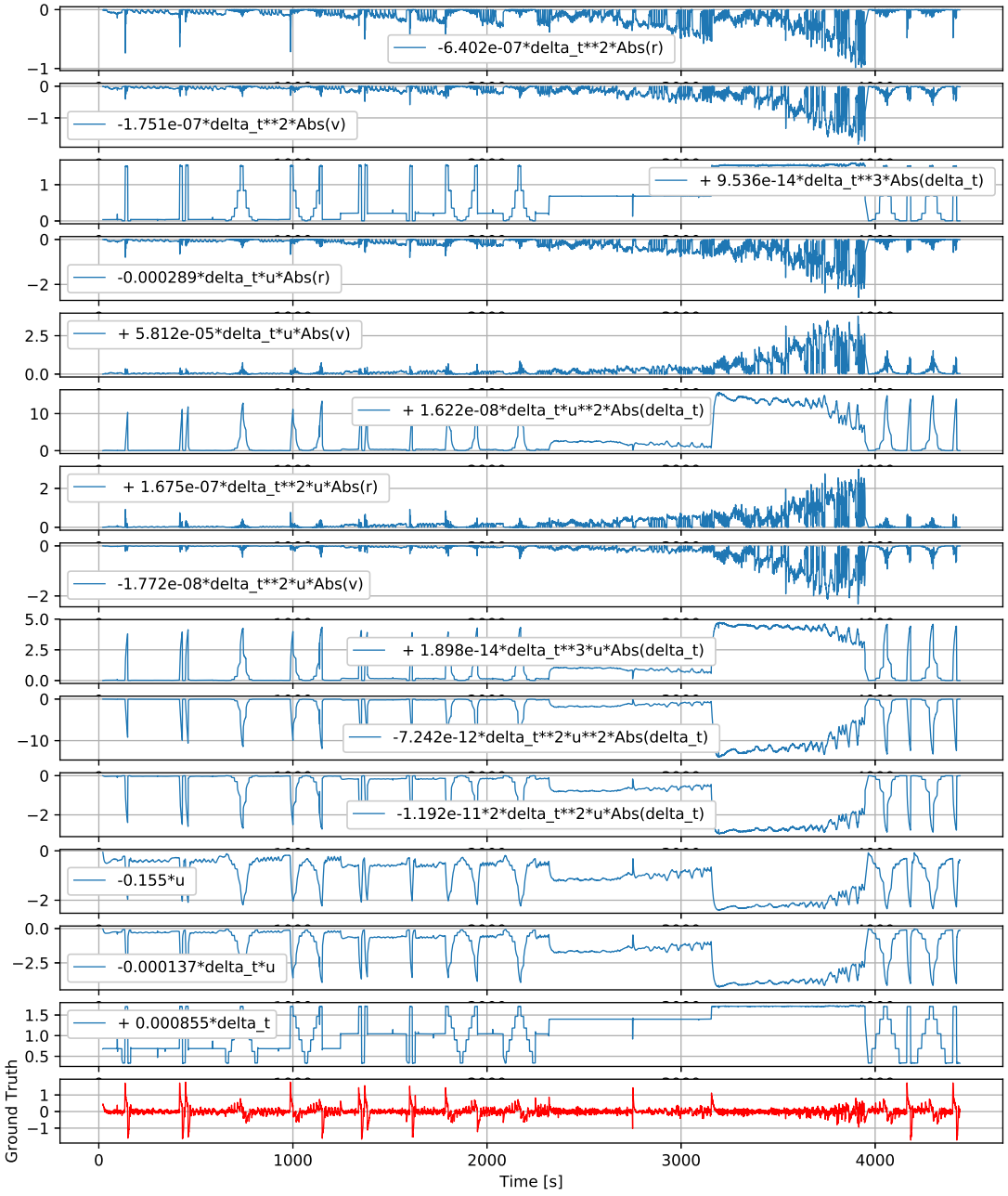


$\dot{v}$ plotted part-wise, for the model with force as inputs to the system

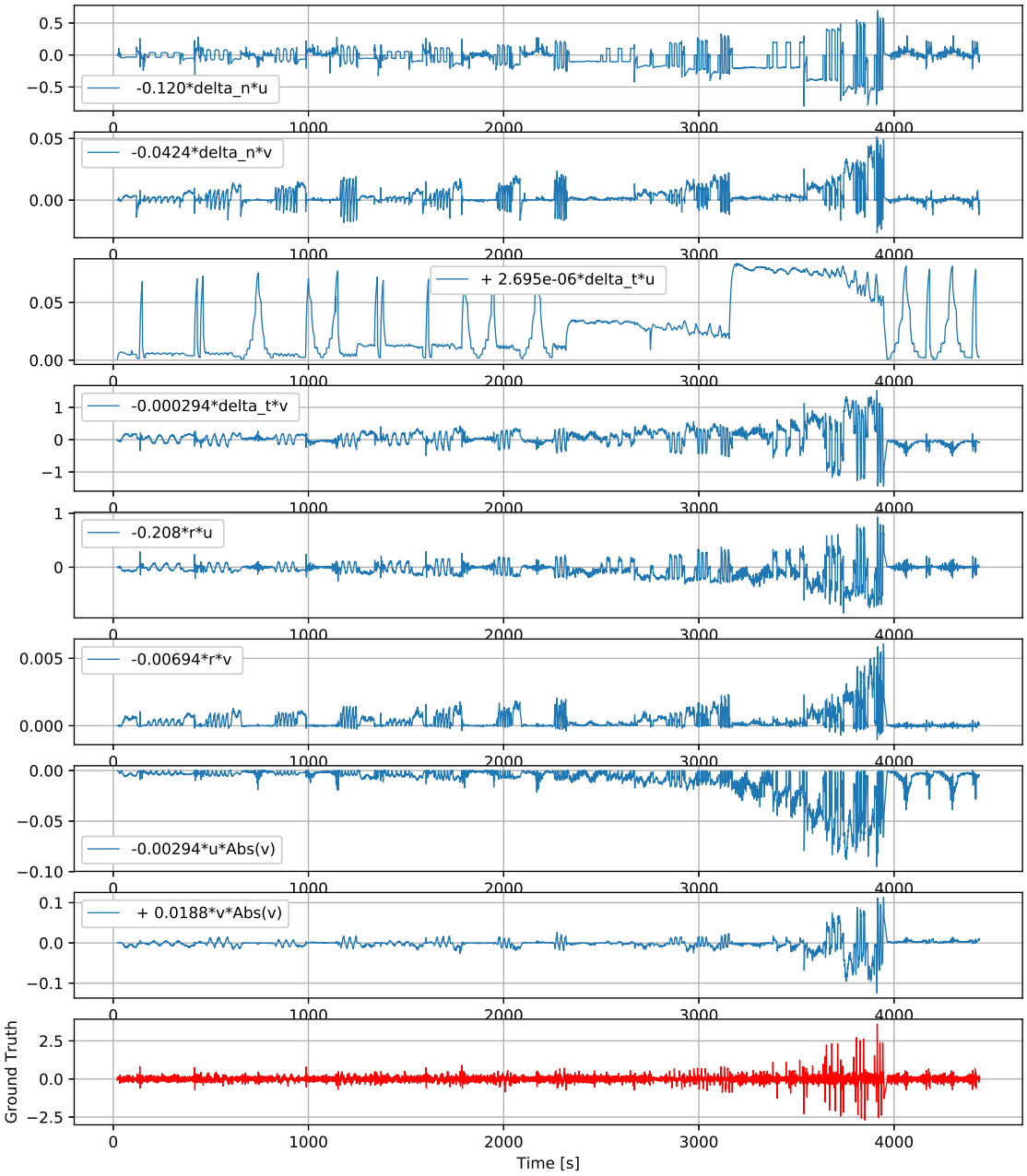


$\dot{r}$ plotted part-wise, for the model with force as inputs to the system

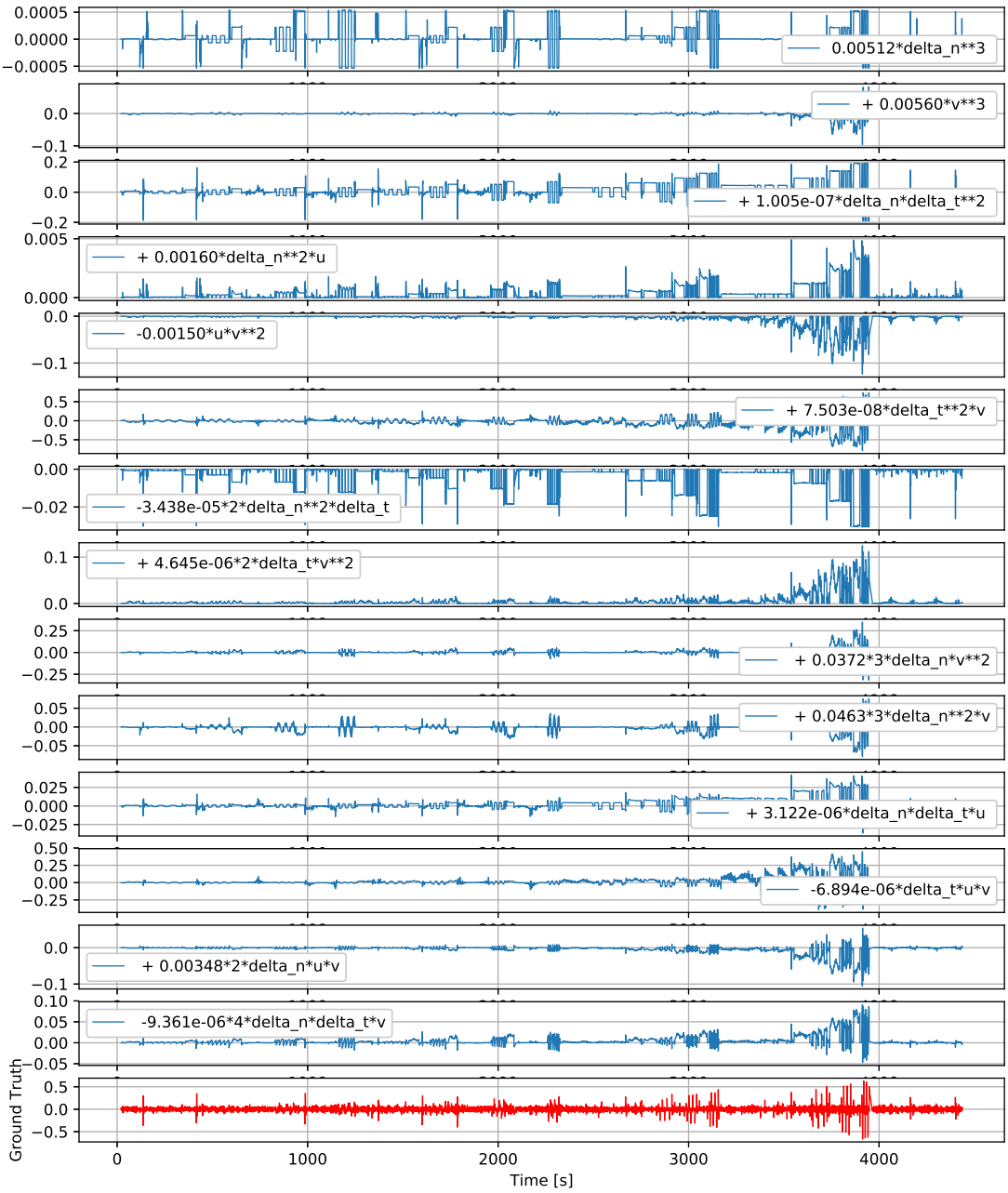
B.2 Part-Wise Model Plots with Controller Inputs as Inputs



$\dot{u}$ plotted part-wise, for the model with control inputs to the system



$\dot{v}$ plotted part-wise, for the model with control inputs to the system



$\dot{r}$ plotted part-wise, for the model with control inputs to the system

C Standard Model

C.1 Original Matrices

Listed here are the original matrices for the standard model, used as the complex system to test the GP algorithm. M is the inertia matrix, C is the sum of Coriolis and centripetal matrices, and D is the sum of the nonlinear and linear damping matrices.

$$M = \begin{bmatrix} 4935.14 & 0 & 0 \\ 0 & 4935.14 & 0 \\ 0 & 0 & 20928 \end{bmatrix} \tag{9.1}$$

$$C = \begin{bmatrix} 0 & 0 & -4935.14v \\ 0 & 0 & 4935.14u \\ -4935.14v & 4935.14u & 0 \end{bmatrix} \tag{9.2}$$

$$D = \begin{bmatrix} -50 - 243|u| & 0 & 0 \\ 0 & -200 - 2000|v| & 0 \\ 0 & 1281 & -1281 - 3594.38|r| \end{bmatrix} \tag{9.3}$$

C.2 Standard Model Parameters and Structure

Parameter	Original Value	New Value
m	4935.14	-
I_z	$2.09 \cdot 10^4$	$1.57 \cdot 10^6$
$X_{\dot{u}}$	0	$-4.47 \cdot 10^4$
$Y_{\dot{v}}$	0	$-7.18 \cdot 10^4$
$Y_{\dot{r}}$	0	-0.256
$N_{\dot{v}}$	0	$-1.28 \cdot 10^6$
$N_{\dot{r}}$	0	$-2.67 \cdot 10^6$
X_u	-50	-239.71
Y_v	-200	$-1.32 \cdot 10^4$
Y_r	0	$-2.38 \cdot 10^5$
N_r	-1281	-194.30
$X_{\|u\|u}$	-243	-9.00
$Y_{\|v\|v}$	-2000	$-3.45 \cdot 10^4$
$N_{\|r\|r}$	-3594.38	$-1.58 \cdot 10^5$

The original parameters and the optimized parameters for the standard model.

The matrices of the standard model:

$$M_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix} \quad M_A = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & N_{\dot{r}} \end{bmatrix} \tag{9.4}$$

$$M = M_{RB} + M_A \tag{9.5}$$

$$C_{RB} = \begin{bmatrix} 0 & 0 & -m(x_g r + v) \\ 0 & 0 & mu \\ m(x_g r + v) & -mu & 0 \end{bmatrix} \quad C_A = \begin{bmatrix} 0 & 0 & Y_{\dot{v}} v + Y_{\dot{r}} r \\ 0 & 0 & -X_{\dot{u}} u \\ -Y_{\dot{v}} v + Y_{\dot{r}} r & X_{\dot{u}} u & 0 \end{bmatrix} \tag{9.6}$$

$$C = C_{RB} + C_A \tag{9.7}$$

$$D_l = \begin{bmatrix} -X_u & 0 & 0 \\ 0 & -Y_v & -Y_r \\ 0 & -N_v & -N_r \end{bmatrix} \quad D_{nl} = \begin{bmatrix} -X_{|u|u}|u| & 0 & 0 \\ 0 & -Y_{|v|v}|v| & 0 \\ 0 & 0 & -N_{|r|r}|r| \end{bmatrix} \tag{9.8}$$

$$D = D_l + D_{nl} \tag{9.9}$$

D Froude Number and the Phases of a Marine Surface vehicle

Froude number is a way of classifying which phase the marine craft is in. How to calculate the number is shown in (9.10), and the table is showing which phase this number corresponds to [9].

$$F_n = \frac{U}{\sqrt{g l_{pp}}} \tag{9.10}$$

where $U = \sqrt{u^2 + v^2}$, l_{pp} is the length of the craft and g is the gravitational acceleration

Froude number	Phase
Fn <0.4	Displacement
0.4-0.5 <Fn <1.0 - 1.2	Semi-displacement
Fn >1.0 - 1.2	Planing

Different phases of the vehicle

E Goodness of Fit

How the calculations for the goodness if fit were calculated, and confidence interval with bootstrapping, is shown in this section.

$$RMSE = \sqrt{\frac{1}{m}\sum(y - \hat{y})^2} \tag{9.11}$$

$$C_p = \frac{1}{m}(MSE + 2d\sigma^2) \tag{9.12}$$

$$AIC = \frac{1}{m\sigma^2}(MSE + 2d\sigma^2) \tag{9.13}$$

$$BIC = \frac{1}{m\sigma^2}(MSE + ln(m)d\sigma^2) \tag{9.14}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{p}(y_i - f(x_i))^2}{\sum(y - \hat{y})^2} \tag{9.15}$$

$$R^2_{adjusted} = 1 - \frac{(1 - R^2)(m - 1)}{m - k - 1} \tag{9.16}$$

$$\sigma^2 = \frac{MSE}{m - d - 1} \tag{9.17}$$

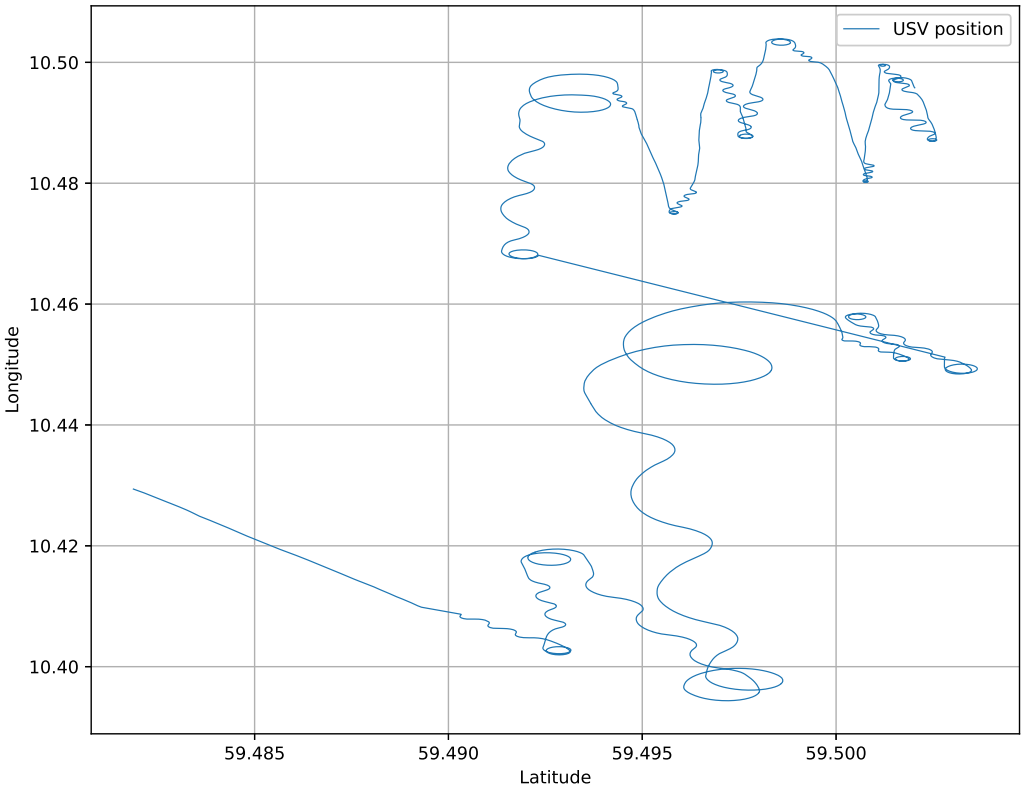
Where m is the number of samples, y is the data, $\hat{y}$ is the prediction, x_i is the ith variable, MSE is the mean squared error, ln is the natural logarithm and d is the number of features.

```
def boot_strap(model, y, B = 10000):
    #residuals
    abs_error = np.abs(y-model)

    #bootstrap
    x_boot = np.random.choice(abs_error, size =(B, int(len(abs_error)/2)),
                              replace = True)

    #mean
    mu = np.mean(x_boot, axis = 1)

    #confidence interval
    mu_avr = np.average(mu)
    n = len(abs_error)
    se = scipy.stats.sem(abs_error)
    h = se * scipy.stats.t.ppf((1+ 0.95) / 2, n-1)

    return mu_avr, mu_avr-h, mu_avr+h
```

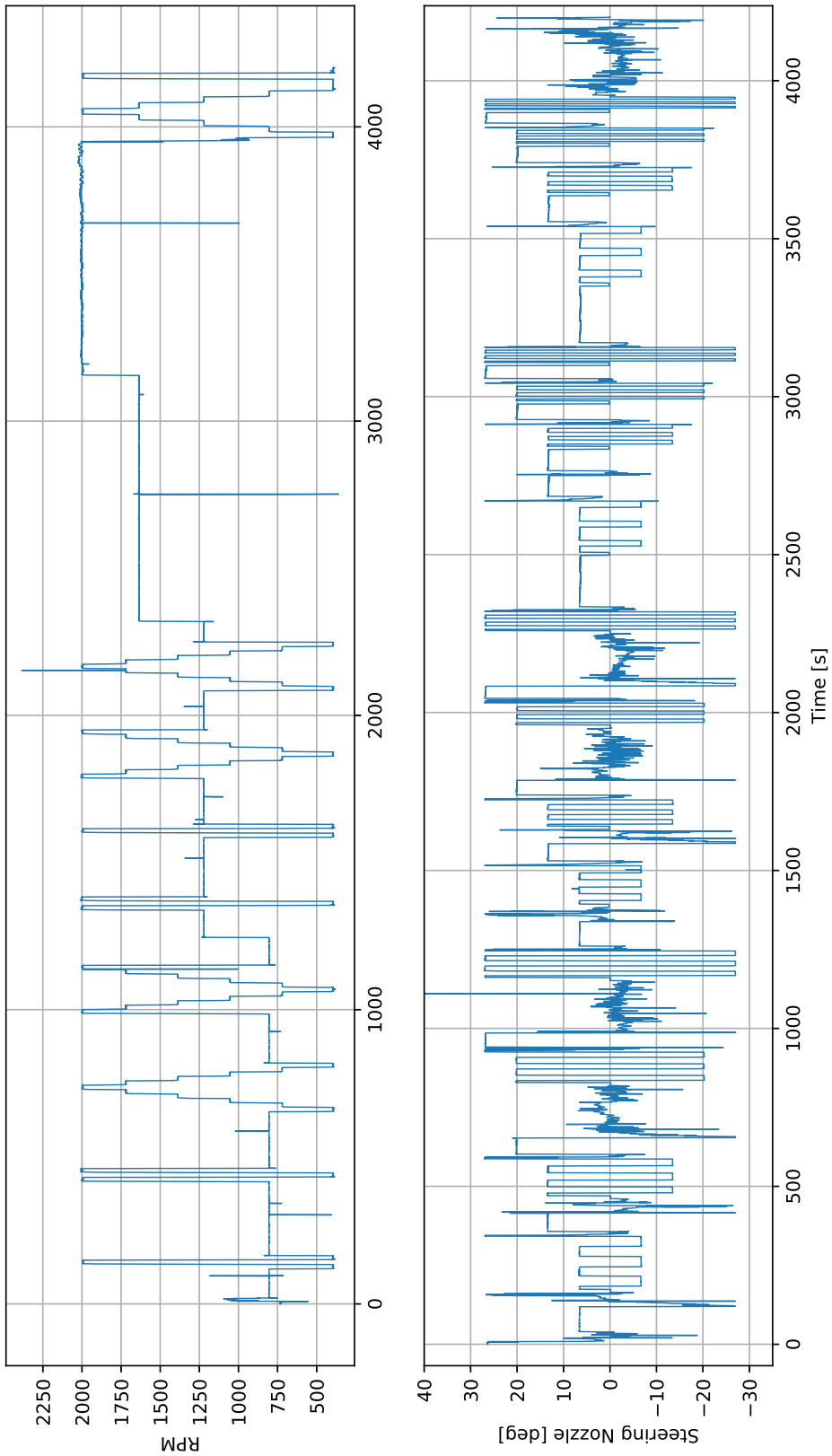
Listing 9.1: Bootstrapping and 95% confidence interval, mu_avr-h, mu_avr+h is the confidence interval

F Data

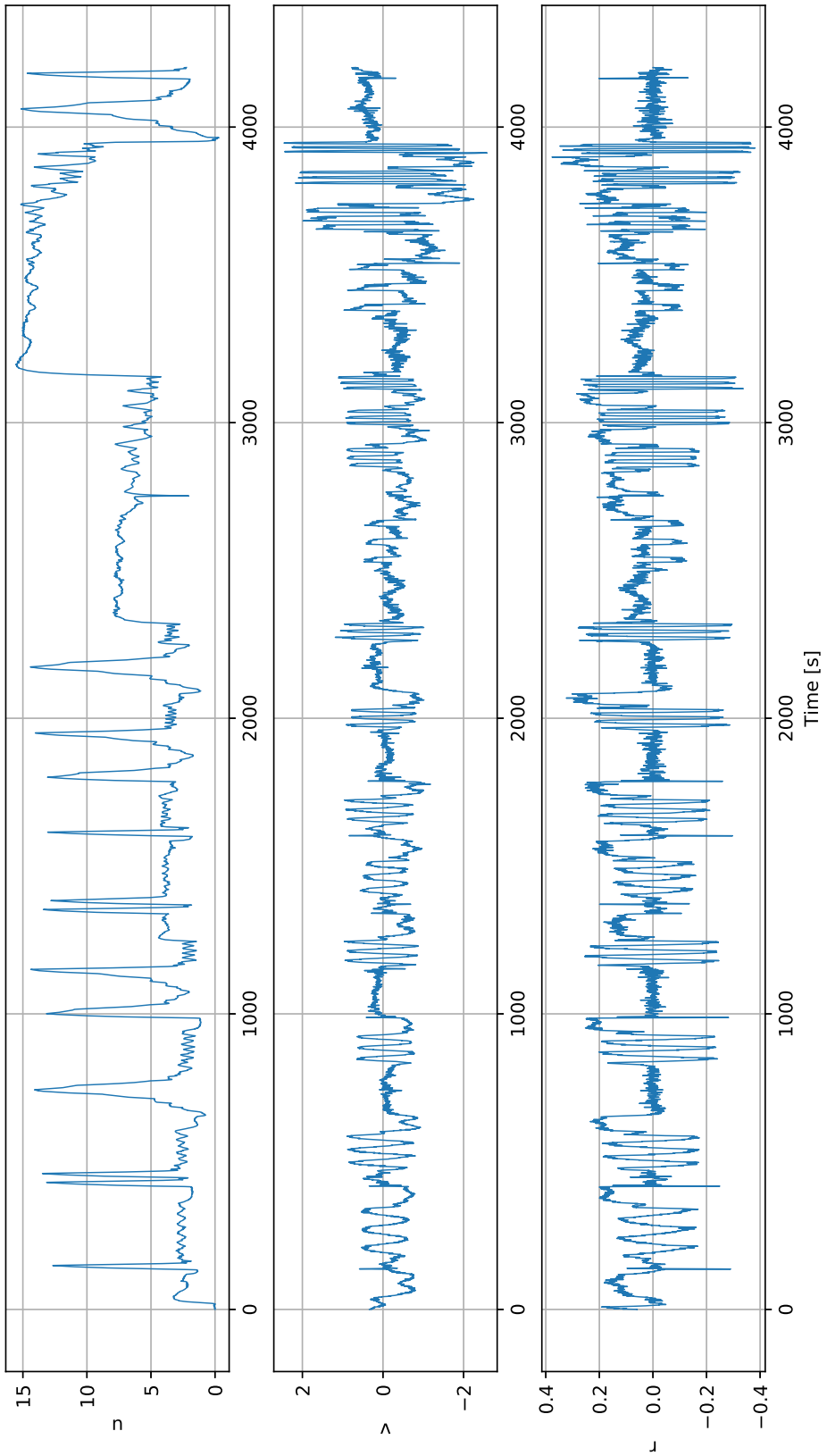
This section presents the data used in the thesis.



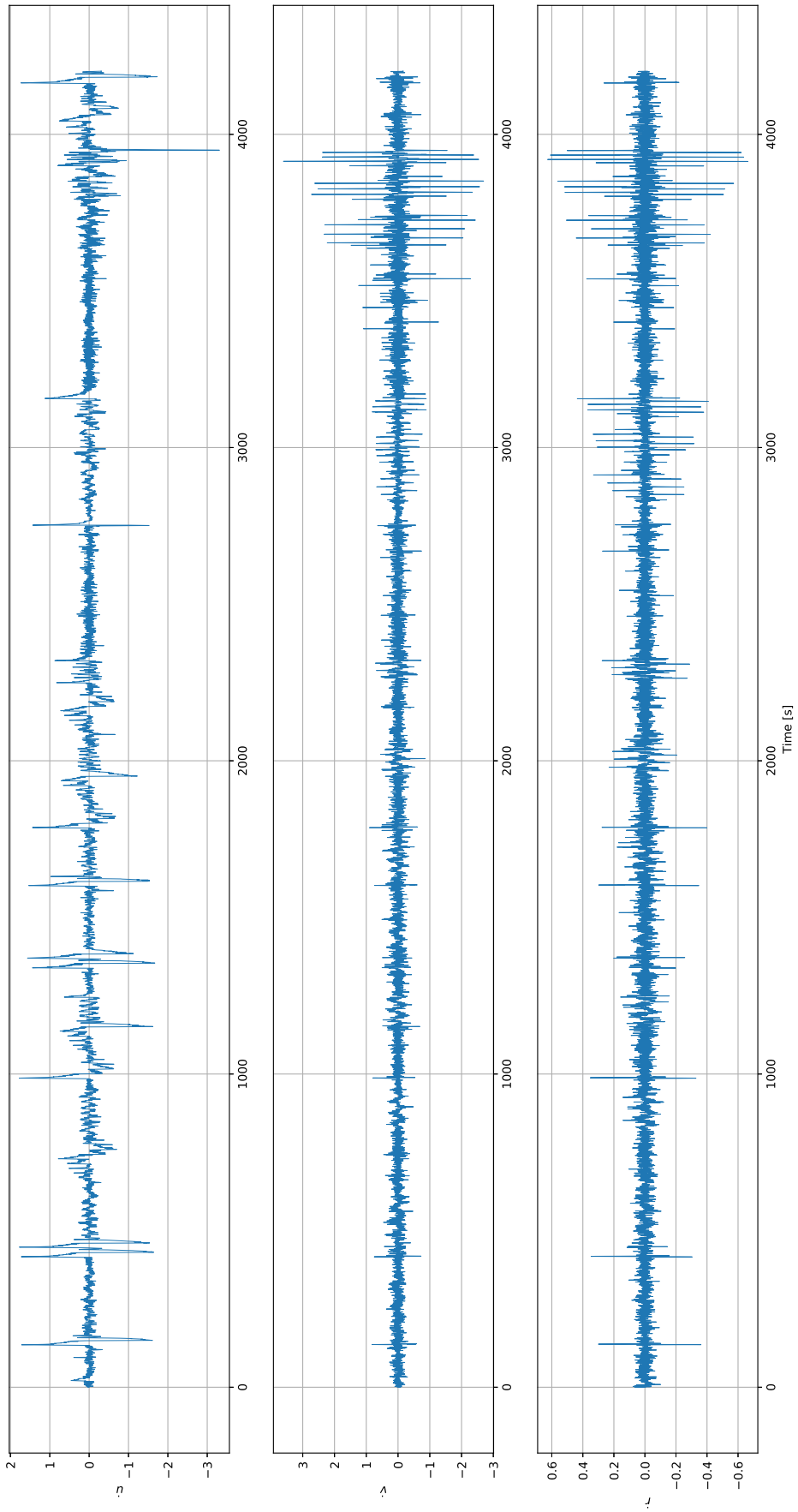
Latitude and Longitude



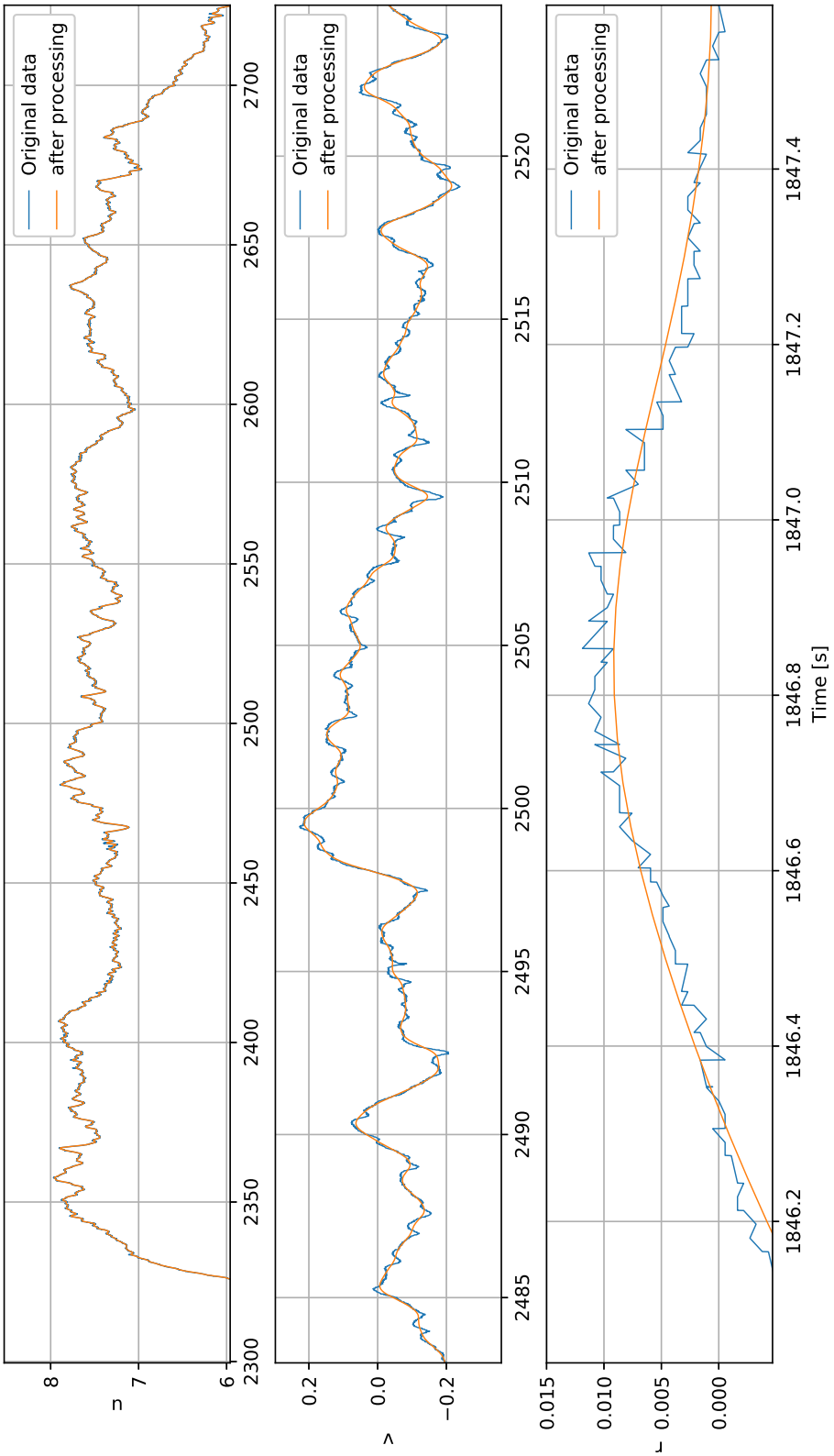
RPM and Nozzle angle



Surge, sway and yaw

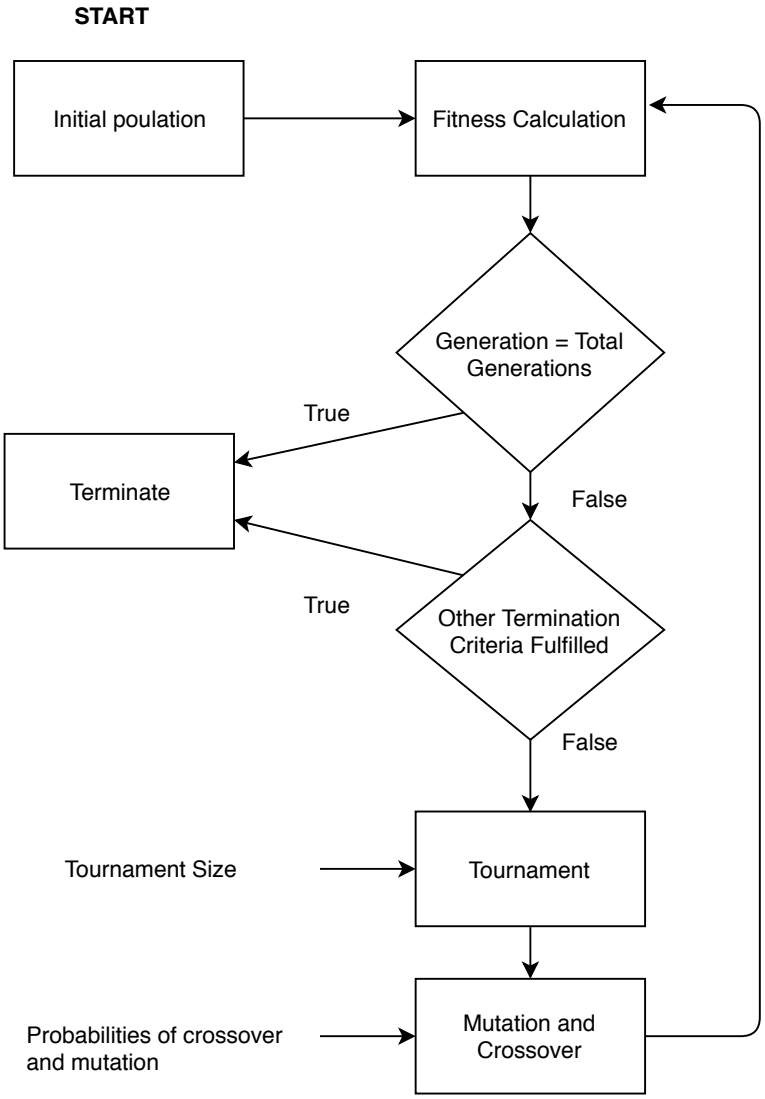


Acceleration

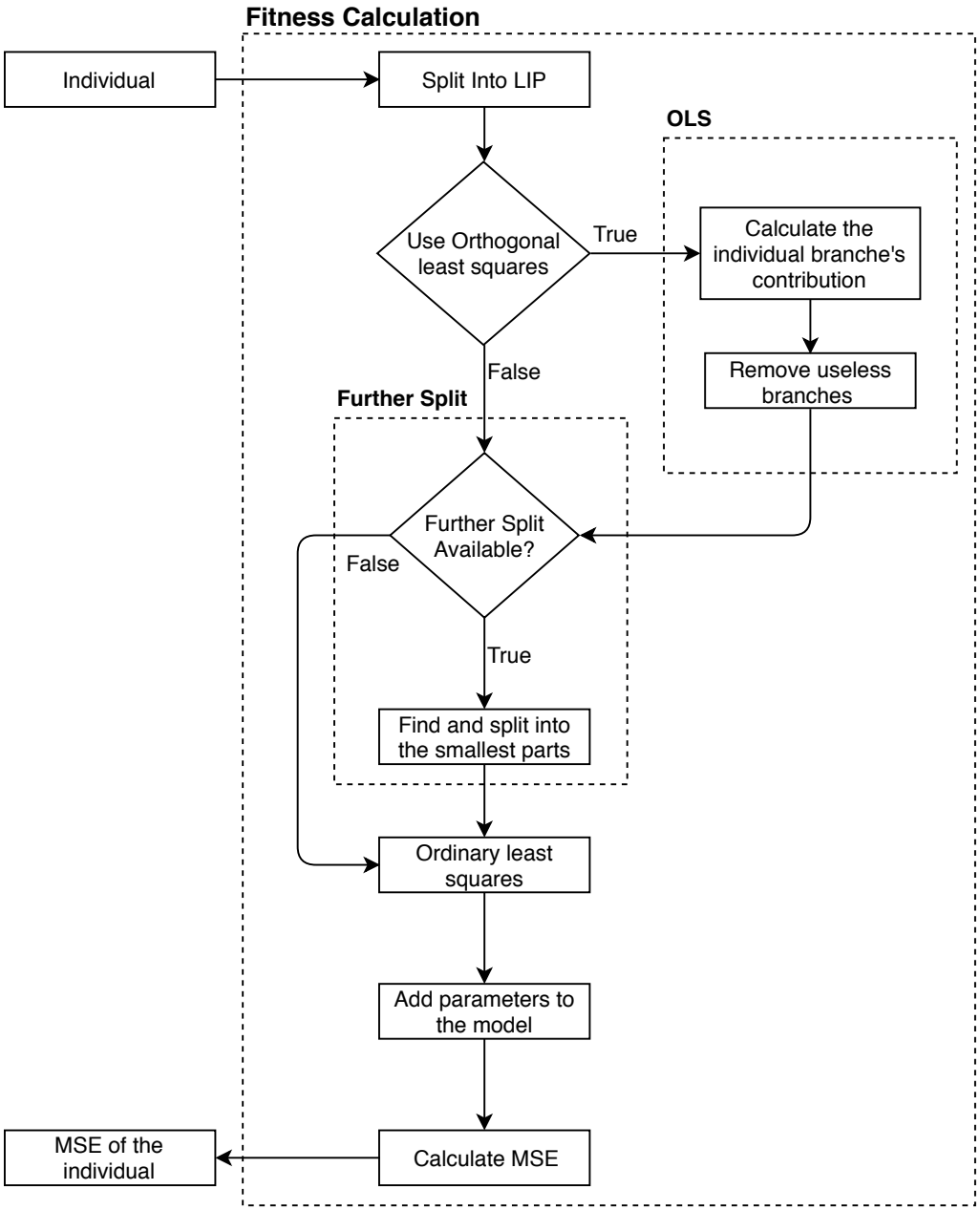


Raw and filtered data

G GP Flowcharts



Overview of GP



Overview of the fitness calculation in GP

References

[1] Abonyi, J. Genetic Programming MATLAB Toolbox, 2014.

[2] Aranda, J., de la Cruz, J., Diaz, J., de Andrés, B., Ruipérez, P., Esteban, S., and Girón, J. Modelling of a High Speed Craft by a Non-Linear Least Squares Method with Constraints. *IFAC Proceedings Volumes 33*, 21 (2017), 221–226.

[3] Billings, S. A., Koenberg, M. J., and Chen, S. Identification of non-linear output-affine systems using Orthogonal Least Squares Algorithm.

[4] De Jong, K., Fogel, D. B., and Schwefel, H.-P. A history of evolutionary computation. *Handbook of Evolutionary Computation*, January (2004).

[5] Eiben, A. E., and Smith, J. E. *Introduction to Evolutionary Computing Genetic Algorithms*. 2012.

[6] Eriksen, B.-o. H., and Breivik, M. Modeling, Identification and Control of High-Speed ASVs: Theory and Experiments.

[7] Forsyth, R. A Darwinian Approach to Pattern Recognition. *Kybernetes* (1981).

[8] Fossen, T. I. *Marine Control Systems: Guidance Navigation, and Control of Ships, Rigs and Underwater Vheicles*. 2002.

[9] Fossen, T. I. *Handbook of Marine Craft Hydrodynamics and Motion Control*. 2011.

[10] Friedberg, R. M. A Learning Machine: Part I. *IBM Journal of Research and Development 2*, 1 (1958), 2–13.

[11] Gagn'e, F.-A. F., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Christian. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research 13* (2012), 2171–2175.

[12] Gandomi, A. H., Alavi, A. H., Arjmandi, P., Aghaeifar, A., and Seyednour, R. Genetic programming and orthogonal least squares: a hybrid approach to modeling the compressive strength of CFRP-confined concrete cylinders. *Journal of Mechanics of Materials and Structures 5*, 5 (2010), 735–753.

[13] Gustafsson, F. Determining the initial states in forward-backward filtering. *IEEE Transactions on Signal Processing 44*, 4 (1996), 988–992.

[14] Halvorsen, G. System Identification of a High-Speed USV, with Genetic Programming. Tech. rep., NTNU, Trondheim, 2018.

[15] Halvorsen, G. Github repository - https://github.com/Gislefh/Master, 2019.

[16] Karvinen, A. Person overboard rescue maneuver.

[17] Khalil, H. K. *Nonlinear Systems*, third ed. 2002.

[18] Koza, J. R. *Genetic programming: On the programming of computers by means of natural selection*,, vol. 33. 1994.

[19] Laumanns, M., Thiele, L., Zitzler, E., Welzl, E., and Deb, K. *Parallel Problem Solving from Nature — PPSN VII: 7th International Conference Granada, Spain, September 7–11, 2002 Proceedings - Fighting Bloat with Nonparametric Parsimony Pressure*. 2002.

[20] Liu, Z., Zhang, Y., Yu, X., and Yuan, C. Unmanned surface vehicles: An overview of developments and challenges. *Annual Reviews in Control 41*, May (2016), 71–93.

[21] Luke, S., and Spector, L. A revised comparison of crossover and mutation in genetic programming. *Genetic Programming 98*, 208-213 (1998), 55.

[22] Madar, J., Abonyi, J., and Szeifert, F. Genetic Programming for System Identification. Tech. rep., 2002.

[23] Madár, J., Abonyi, J., and Szeifert, F. Genetic programming for the identification of nonlinear input-output models. *Industrial and Engineering Chemistry Research 44*, 9 (2005), 3178–3186.

[24] Moreno-Salinas, D., Besada-Portas, E., López-Orozco, J. A., Chaos, D., De La Cruz, J. M., and Aranda, J. Symbolic regression for marine vehicles identification. *IFAC-PapersOnLine 28*, 16 (2015), 210–216.

[25] Poli, R., and Mcphee, N. Parsimony Pressure Made Easy: Solving the Problem of Bloat in GP.

[26] Rajesh, G., and Bhattacharyya, S. K. System identification for nonlinear maneuvering of large tankers using artificial neural network. *Applied Ocean Research 30*, 4 (2008), 256–263.

[27] Stephens, T. GP Learn.

[28] Wiley, J., and Sons, . Classical Linear Regression. *Econometric theory* (1964), 156 – 212.

[29] Zhang, B.-t., and Mühlenbein, H. Balancing Accuracy and Parsimony in Genetic 1 Introduction. *Evolutionary Computation 3*, 1 (1994), 1–33.

Gisle Halvorsen

Utilizing Genetic Programming for System Identification of a High-Speed USV

NTNU
Norwegian University of
Science and Technology