Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Bøe, Mikael

Numerical Modelling of Sailing Hydrofoil Boats

Master's thesis in Marin Teknikk
Supervisor: Steen, Sverre
June 2019

NTNU
Norwegian University of
Science and Technology

Bøe, Mikael

Numerical Modelling of Sailing Hydrofoil Boats

Master's thesis in Marin Teknikk
Supervisor: Steen, Sverre
June 2019

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU
Kunnskap for en bedre verden

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MASTER THESIS IN MARINE TECHNOLOGY

SPRING 2019

FOR

Mikael Bøe

Numerical modelling of sailing hydrofoil boats

High-performance sailing boats are increasingly using hydrofoils to lift the hull out of the water and thereby reduce the total resistance at high speed. For instance have the later America's Cup yachts been constructed in this way. When predicting the performance of a sailing yacht, it is common determine the condition that balances the aerodynamic forces (mainly on the sails and rig) and the hydrodynamic forces on the hull, keel and rudder. This involves finding the trim, heel, yaw (drift angle), speed and required rudder – often using an iterative procedure. The process is often named Velocity Prediction Process (VPP). Traditionally, hydrodynamic forces have been found by interpolation in a large, multi-dimensional table coming out of an extensive series of captive model tests (using a yacht dynamometer). In the later years, CFD is increasingly used instead of model tests. Aerodynamic forces might be determined in a similar way, using wind tunnel experiments, CFD, or analytics-based calculations. For the sail rig, a so-called polar might be constructed from these experiments or calculations, and used as input to the VPP.

The objective for the master thesis is to develop a simulation model for a sailing hydrofoil boat, aimed for later inclusion in a training simulator for deck crew. It is expected that emphasis will be put on the hydrodynamic part, with further focus on the hydrofoil performance. Any graphical user interface or logic related to the training simulator is not expected, but the aim of creating a mathematical model suitable for use in a training simulator means that the work shall aim for a time-domain simulation model fast enough for real-time simulation. It is acknowledged that this again means that fairly drastic simplifications in the mathematical modelling of the complicated physics might be necessary. The thesis shall include a description of the theory implemented in the simulation model, clarifying the simplifications used, and then document the verification and validation of the simulation model, before the performance and potential of the developed simulation model is discussed.

In the thesis the candidate shall present his personal contribution to the resolution of problem within the scope of the thesis work.

Theories and conclusions shall be based on mathematical derivations and/or logic reasoning identifying the various steps in the deduction.

The thesis work shall be based on the current state of knowledge in the field of study. The current state of knowledge shall be established through a thorough literature study, the results of this study shall be written into the thesis. The candidate should utilize the existing possibilities for obtaining relevant literature.

The thesis shall be organized in a rational manner to give a clear exposition of results, assessments, and conclusions. The text should be brief and to the point, with a clear language. Telegraphic language should be avoided.

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

The thesis shall contain the following elements: A text defining the scope, preface, list of contents, summary, main body of thesis, conclusions with recommendations for further work, list of symbols and acronyms, reference and (optional) appendices. All figures, tables and equations shall be numerated.

The supervisor may require that the candidate, in an early stage of the work, present a written plan for the completion of the work. The plan shall include a budget for the use of laboratory or other resources that will be charged to the department. Overruns shall be reported to the supervisor.

The original contribution of the candidate and material taken from other sources shall be clearly defined. Work from other sources shall be properly referenced using an acknowledged referencing system.

The thesis shall be submitted electronically (pdf) in Inspera:

- Signed by the candidate
- The text defining the scope (this text) (signed by the supervisor) included

The candidate will receive a printed copy of the thesis.

Supervisor : Professor Sverre Steen
Start : 15.01.2019
Deadline : 11.06.2019

Trondheim, 15.01.2019

Sverre Steen
Supervisor

Preface

I would like to thank my supervisor Professor Sverre Steen for guidance and for keeping me grounded at times I was to ambitious. In addition, I would like to point out that the project would not have been possible without his knowledge, enthusiasm in the subject and clear guidance.

Although I have much experience in sailing, the implementation of theory to a numerical model has challenged me greatly. Therefore, I have benefited greatly from the project and has increased my knowledge and challenged both my theoretical and practical skills.

It is also important to mention my office colleagues as they have always shown support during these trying times.

Lastly, I would like to thank my family for continued support throughout my time as a student.

Mikael Bøe June 10, 2019

Summary

The motivation of this thesis stems from the desire to create a training simulator for experienced sailors not familiar with hydrofoils. As the approach to such a simulator can be based on several types of numerical models, a review is made where potential based methods are deemed the most efficient. Therefore, the goal of the thesis is to view the feasibility of using a potential flow approach to numerical assessment of sailing hydrofoil vessels. As such, a model is implemented to analyse the dynamic behaviour of a sailing hydrofoil, this is investigated to asses whether the method has any merit. In addition, as a training simulator is the background for the thesis, attempts will be made to approach real-time computational speeds.

The chosen numerical method behind the simulation is the potential flow based lifting surface, or vortex lattice method as it is also known as. Briefly explained the lifting surface model solves Laplace's equation within the parameters of linear foil theory. It accomplishes this by applying boundary conditions directly onto the camber line of the foil thus disregarding the effect of thickness and solving the condition with a distribution of vortex singularities. Where the condition to be solved is zero normal flow across the surface. Based on the investigations carried out it is believed the created lifting surface model sufficiently models the effects of camber and aspect ratio during attached conditions. Furthermore, the modelling of dynamic lift is also believed to be sufficiently accurate during attached conditions, which was investigated against Theodorsen's function.

The overall simulation model uses the above mentioned lifting surface to determine the forces from the foils and sail. Thereafter the equation of motion is solved, where a generic model of the mass matrix has been used as well as an analytical approach to added mass, where aerodynamic added mass has been neglected. An attempt was made to simulate the crew of vessel by adjusting their position as well as a de-powering scheme of the sail, which were both implemented to compensate for roll motion. It is believed the current model is able to simulate the dynamic behaviour of a sailing hydrofoil. Although, the model is far from perfect and experienced an abnormal behaviour during specific conditions. In addition, the current implementation of crew behaviour is believed to have made the vessel unstable. With regards to a potential based simulator, the method is adequate but suffers predictably during conditions where viscous effects are dominant. This means, the simulator will either have to be made for certain scenarios or switch to adequate models when potential flow suffers.

Sammendrag

Motivasjonen til denne avhandlingen kommer fra ønsket om å lage en treningssimulator for erfarne seilere som ikke er kjent med hydrofoilfartøy. Siden en simulator kan være basert på flere typer numeriske metoder, blir en undersøkelse gjort hvor potensial strøm methoder sett som mest egnet. Målet til oppgaven er å se på gjennoførbarheten ved å bruke metoder fra potensialteori for å simulere hydrofoilseilbåter. Basert på at en treningssimulator er bakgrunn for oppgaven, vil det tilstrebes å nå sanntidsberegninger for farkosten.

Den valgte numeriske metoden bak simulaotren er en potensialteorimentode kalt løfteflate, kjent som vortex lattice method. Kort fortalt løser løftflaten Laplace's lingning ved bruk av lineær foilteori. Dette gjøres ved å plassere grensebetingelsene direkte på kurvaturflaten til foilen. Dermed neglisjeres tykkelsen av foilen og ligningen løses med en fordeling av virvler på kurvaturflaten. Hvor betingelsen som skal bli løst er null normal strømmning over flaten. Simuleringene tilsier at modellen tar hensin til både krumning og apsektforholdet av tilstrekkelig grad, under Kutta kondisjon. I tillegg, er det dynamiske løftet funnet å være tilstrekkelig nøyaktig, sammenlignet med Theodorsen's funksjon.

Simuleringsmodellen bruker den nevnte løftflaten til å finne kreftene fra foilene og seilet. Deretter blir bevegelsligningene løst, hvor en generisk model av massematrisen blir anvdent. En analytisk løsning til tilleggsmassen blir brukt, hvor den aerodynamiske tilleggsmassen neglisjeres. Et forsøk ble gjort ved å simulere mannskapet til båten ved å justere posisjonen deres og trimming av seilet, for å kompensere for rullebevegelse. Basert på resultatene kan det antas at modellen klarer å simulere den dynamiske oppføreselen til en hydrofoilseilbåt til tross for at modellen opplevde unormale bevegelser under spesifikke forhold. Den nåværende implementeringen av mannskapets manøvre kan gjøre båten ustabil. Med hensyn til en potensialtoeri basert simulator er metoden tillstrekkelig. Men signifikante usikkerheter oppstår når viskøse krefter dominerer. Dette indikerer at simulatoren bør bygges men hensyn til spesifikke scenarioer eller endres til mer passende metoder når potensialtoeri feiler.

Contents

List of Figures

List of Tables

Nomenclature

List of Abbreviations

DOF	Degree of Freedom
FPS	Frames Per Second
GPU	Graphics Processing Unit
HPC	High Performance Computing
ITTC	International Towing Tank Conference
LBM	Lattice-Boltzmann Method
NACA	National Advisory For Aeronautics
NS	Navier-Stokes
VLM	Vortex Lattice Model

List of Symbols

α	True Wind Angle
β	Apparent Wind Angle
$\boldsymbol{n}$	Normal Vector
$\ddot{\eta}$	Acceleration
ΔF	Force on element with directions x,y,z
Δp	Pressure on element
Δt	Time Step
Δx	Element length in x direction
Δy	Element length in y direction
$\dot{\eta}$	Velocity
η	Displacement
Γ	Vortex Strength

μ	Kinematic Viscosity
Φ	Perturbation Velocity Potential
ϕ, θ, ψ	Angle about x,y,z axis
Φ_∞	Free Stream Velocity Potential
ρ	Density
$\tau_{x,y}$	Tangential vector in x,y direction
Θ	Angle Vector about x,y,z axis
C_B	Block Coefficient
C_D	Drag Coefficient
C_F	Frictional Coefficient
C_L	Lift Coefficient
C_M	Midship Coefficient
C_P	Pristmatic Coefficient
C_{Da}	Drag Coefficient Slope
C_{La}	Lift Coefficient Slope
F_D	Driving Force
F_R	Heeling Force
F_T	Total Aerodynamic Force
M_a	Added Mass Matrix
Q_∞	Total Incoming Flow Velocity
Re	Reynolds Number
S_b	Surface Area
$U_\infty, V_\infty, W_\infty$	Velocity of Incoming Flow in x,y,z direction
V_A	Apparent Wind Speed
V_T	True Wind Speed
w_{ind}	Induced Downwash
a	Influence Matrix
AR	Aspect Ratio
b	Downwash Influence Matrix

C	Centripetal Matrix
c	Chord
D	Drag Force
F	Force vector with directions x,y,z
I	Mass Moment of Intertia
K,M,N	Torque about x,y,z axis
L	Lift Force
M	Mass Matrix
m	Mass
N	Total Number of Elements
n,m	Number of Elements in Chord, Span
p,q,r	Rotational velocity about x,y,z axis
R	Rotational Matrix
S	Skew Matrix
s	Span
t	Time Step Counter
u,v,w	Velocity in x,y,z Direction
X,Y,Z	Force in x,y,z direction

Chapter 1

Introduction

The utilization of hydrofoils on a sail powered vessel is nearly as old as the usage of hydrofoils, yet academic and industrial interest is not as abundant as conventional hydrofoil vessels. However, the last few Americas Cup's has invigorated a broader interest in the subject, resulting in both acamedic works such as tacking simulations (Lidtke et al., 2013) and commercial vessels such as the WASPZ (WASZP, 2016).

However as the resources required to participate are large and the time in which the crew may train with the vessel before the race is small. Therefore it becomes difficult for new participants to compete. To reduce this gap in experience a training simulator can be used to familiarise experienced sailors in how to handle such a vastly different vessel. The motivation of this thesis stems thus from the desire to create a training simulator for experienced sailors not familiar with hydrofoils. Given the time-frame and resources available to simulate a sailing hydrofoil vessel, the goal of the thesis is to view the feasibility of using a potential flow approach to numerical assessment of sailing hydrofoil vessels. In addition, as a training simulator is the background for the thesis, attempts will be made to approach real-time computational speeds.

1.1 Scope of the Work

The scope of the work is summarised below.

- Development of a simulation model for a sailing hydrofoil model and review its capability.
- A suitable model for the hydrofoils will be chosen and argued for. In addition, the model will be verified and validated.
- A suitable model for the sail will be chosen, however an emphasis is placed on the hydrodynamical part.

1.2 Real time interactive simulation challenges

The three main tasks an interactive model has to overcome are and not necessarily in this order; Simulation, visualisation and steering. Steering is the users interaction with the simulation, and is required to work in tandem with visualisation and simulation at the same time (Wenisch et al., 2005). A common attempt at this is to allow some delay, this is done by updating the geometry based on user interaction while processing the previous settings (Linxweiler et al., 2010). The speed at which these three tasks have to be done is measured in frames per second, usually this speed should not be lower than 20 FPS (Ou et al., 2008). Meaning the model has to solve 20 simulations per second, thus for a single simulation the computational time should not exceed 42 milliseconds.

The challenges described here revolve around computer architecture and mainly deal with how one efficiently processes and updates data in parallel. The solution to these challenges often arise from the computer game industry, where all of the above are tackled, although the industry understandably prioritises visualisation and not physical accuracy (NVIDIA, 2019). Even though these challenges are not physical in nature, they are important to mention when assessing a numerical method the simulation solver is to be based on.

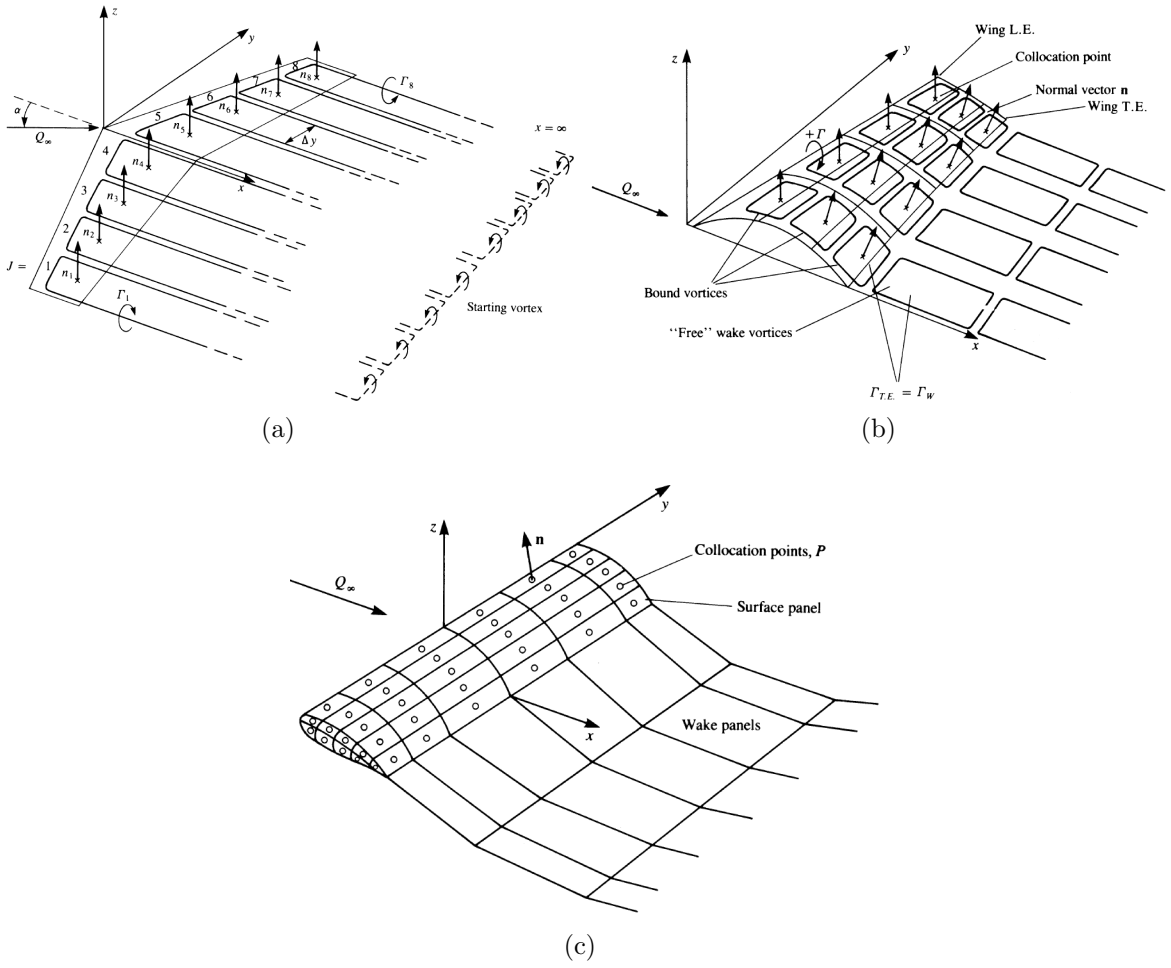
1.3 Numerical Methods

When deciding upon a numerical method for a training simulator the computational cost can be a deciding factor. However, with reduced computational cost comes negligence and simplicity which reduce the capabilities of the simulator. Thus a cost-benefit analysis is required to assess the most optimum methodology based on today's methods and technology. It is assumed going forward that the hardware utilized in the simulator is not a supercomputer but at least several times stronger then the average computer. We can split the basis of the simulator into the four different categories; Empirical methods which use results from experimental and full scale trials in combination with linearised equations of motion, potential based methods which neglect the effects of viscosity, Navier-Stokes based methods and Lattice-Boltzmann methods.

Empirical methods would be the most optimal to base a simulator on, as the computational cost and development complexity would be smallest of the three. The simulator would be based on the manoeuvring equations to control the simulator, which would require the vessels specific non-dimensional manoeuvring coefficients (HochBaum, 2019). This however would require extensive experimental testing, quickly increasing the time and effort required to run the simulator. In addition, for each specific vessel, brand new experiments would have to be performed. Not only is it time consuming to perform model tests for regular vessels, with sailboats the hull and sail would have to be experimented on separately, due to how different physical properties are scaled. Although this complication can be removed entirely by performing full scale trials, which could be used to develop a simulator. Subsequently, this is the reason numerical methods are utilised and the reason this thesis will not consider such an approach any further.

Potential Flow based methods

A potential based approach has the opposite benefits of empirical methods. They are fast, flexible and have been widely developed and validated throughout their inception in the 1940's. However, due to the fluids approximation of inviscid, incompressible and irrotational the approach suffers when viscous effects are dominant. Thus the application of an potential approach can only consider the case of attached flowfields. Where this is not the case, the pressure distribution will be faulty, but the solution will however point towards where there might exist separation (Chattot and Hafez, 2015). The potential methods considered here are boundary element methods, namely the Prandtl lifting line, the vortex lattice method and panel methods, these are visualised in Figures 1.1a, 1.1b and 1.1c respectively.



(a) (b) (c)

Figure 1.1: Potential Flow methods, a - Prandtl's lifting line, b - Vortex Lattice Method, c - Panel method.

Potential solvers mainly follow the same procedure, although differing in either boundary condition, complexity of the geometry or choice of singularity element. The procedure is as follows; Discretisation of the geometry, calculate influence coefficients, solve a linear system of equations and calculate forces. As the methods are very similar and the largest difference being how the geometry is handled, it can be important to assess what these simplifications imply.

Prandtl's classical lifting line involves representing the body as a series of horseshoe elements along the span. Even though the method can account for the effects of aspect ratio, wing sweep, dihedral and side slip the model becomes inaccurate for lower aspect ratios. In addition, the method does not consider the camber or thickness, although thickness has an effect the camber is more important (Abbot et al., 1945). As the camber is not considered, it would be difficult to justify its use when modelling the sail of vessel. This is due to the sail can be viewed as a very thin foil with a large camber (Marchaj, 2003). Although the model would be sufficient for the hydrofoils if uncambered foils are utilised.

The lifting surface or vortex lattice method (VLM), distributes a series of vortex ring elements along both the span and chord on the camber surface. Both the lifting line and lifting surface are based on linear foil theory as such they both revolve around thin foils, with small angles of attack and small disturbances of the free stream (Chattot and Hafez, 2015). Therefore the methods accounts for the same geometrical factors but the VLM considers camber effects and lower aspect ratios, whereas the thickness is still not considered. In terms of computational resources the increase is noticeable, but not large.

Panel methods is an extension of the lifting surface from the camber surface to an enclosed geometry. The method is optimum when any arbitrary geometry is to be handled, although the method is more prevalent for aerodynamics as the entire aircraft can be considered as a lifting body. Being able to handle any arbitrary geometry however is a lucrative capability as many vessels utilise abnormal wing shapes.

As the lifting surface method incorporates the most important geometrical factors affecting the lift of a foil, the method is chosen and implemented further.

Navier-Stokes

Navier-Stokes (NS) methods revolve around solving newtons second law by adhering to the continuity equation, conservation of energy and kinematic viscosity (Blazek, 2015). There exist many free and commercial solvers which primarily vary in user friendliness and mesh generation schemes. The method requires intense computer power and although computational power increases, current growth rate projects half of the normal rate (Spalart and Venkatakrishnan, 2016).

(Kenny et al., 2008) created a helicopter flight simulator coupled with an CFD solver, using high performance computing (HPC) clusters. They found when utilizing such resources that the bottleneck became data messaging between the different systems.
(Roper et al., 2006) performed a similar study although did not couple the CFD tool with the pilot simulator. Their method was to simulate numerous steady state solutions and interpolate before exporting to a look-up table for the simulator to use. This type of approach is valid when training the user for specific tasks and conditions, and can be updated given time and resources.

Based on the studies regarding CFD coupled training simulators, it can be deduced that such methods are feasible but require extensive computational infrastructure both to simulate and process data. Which is outside of the scope of this thesis and as such the method will not be considered further, even though it is most optimal given enough resources.

Lattice-Boltzmann

The Lattice-Boltzmann method (LBM) is an alternative means to traditional Navier-Stokes solvers. In contrast, LBM considers the fluid on a microscopic scale while NS is on a macroscopic scale. On the microscopic scale the particles are represented statistically as distribution functions (Succi, 2001). The method seems more prevalent in aeronautical academics with limited studies in the maritime academic community.

The main advantage is LBM's large parallelisation potential which when working in tandem with graphics processing units (GPU) allow for very fast computational speeds. However, due to the meshing scheme in LBM, which is a uniform Cartesian grid, the size becomes substantial leading to high computational requirements. Thus the method is most optimal when utilising HPC, compared to traditional CFD which has the option, although slow, of being run on a laptop.

Although the use of Lattice-Boltzmann methods in training simulators seems non-existent, several studies have been performed to test its real-time capabilities. (Delbosc, 2015) performed a study testing the real-time simulation of fluids, which suggest the method is up to the challenge given enough HPC. Other studies such as NASA's (NASA, 2017) comparison between NS and LBM found LBM to be 12-15 times faster and equally accurate. However, their lowest grid size of a single landing gear had 56 million cells while their highest 1.6 billion. Which would require a tremendous amount of HPC to compute.

Chapter 2

Lifting Surface Model

This section describes the numerical approximation of a lifting surface and is completely based on sections 10.4.6, 12.3 and 13.12 in the book Low-Speed aerodynamics (Katz and Plotkin, 2001). The method is potential based in nature and aims to find the strength of singularity elements on the body's surface. The model is split into three aspects, grid generation, a steady solution and an unsteady solution. The steady solution refers to static conditions, while the unsteady solution is time-dependent.

Shortly explained the model solves Laplace's equation within the parameters of linear foil theory. It accomplishes this by applying boundary conditions directly onto the camber line of the foil, rather than solving for the flowfield in the whole fluid volume. The model disregards the effects of thickness and solves the condition with a distribution of vortex singularities. Where the condition to be solved is zero normal flow across the surface, which is expressed in equation 2.1. Where Φ is the perturbation potential, Φ_∞ is the free stream potential and n is the normal vector of the surface.

$$\nabla(\Phi + \Phi_\infty) \cdot \boldsymbol{n} = 0 \tag{2.1}$$

Going forward N is the total number of elements distributed across the geometry and is of size $i \times j$, where i and j are the number of elements along the chord and span respectively.

2.1 Constant Strength Vortex Segment

The fundamental function behind the model is based on Biot-Savart law, which is represented in equation 2.2. The equation is comprised of a vortex strength, Γ, the distance between a vortex and an arbitrary point P in space, $\vec{r}$, the size of a vortex segment, $d\vec{l}$, and the radial direction to the evaluated point P denoted $\hat{r}$.

$$q = \frac{\Gamma}{4\pi} \int \frac{d\vec{l} \times \hat{r}}{|\vec{r}|^3} \tag{2.2}$$

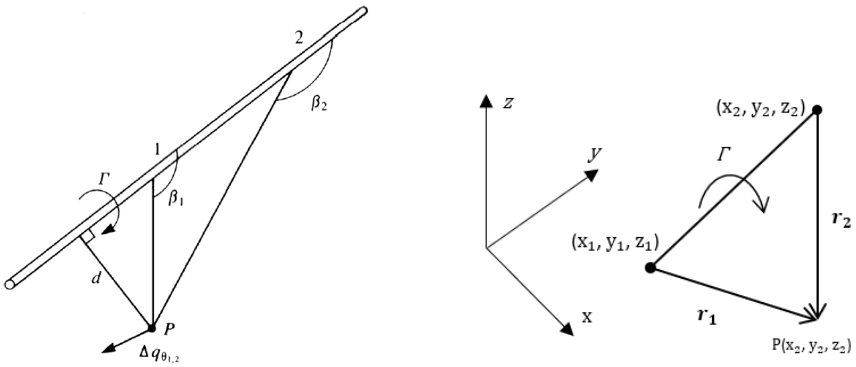


Figure 2.1: Nomenclature used by a straight vortex segment. (Katz and Plotkin, 2001)

The velocity induced by a segment $d\boldsymbol{l}$ of equation 2.2 at a point P is

$$\Delta \boldsymbol{q} = \frac{\Gamma}{4\pi}\frac{d\boldsymbol{l} \times \boldsymbol{r}}{r^3} \qquad and\ in\ scalar\ form \qquad \Delta q_\theta = \frac{\Gamma}{4\pi}\frac{\sin\beta}{r^2}dl \tag{2.3}$$

From Figure 2.1 the following relationships are given

$$l = r\sin\beta \qquad and \qquad \tan(\pi - \beta) = \frac{d}{l}$$

Which means

$$l = \frac{-d}{\tan\beta} \qquad and \qquad dl = \frac{d}{\sin^2\beta}d\beta$$

Substituting these terms into equation 2.3 and integrating over the section in Figure 2.1 leads to

$$(q_\theta)_{1,2} = \frac{\Gamma}{4\pi d}\int_{\beta_1}^{\beta_2}\sin\beta d\beta = \frac{\Gamma}{4\pi d}\left(\cos\beta_1 - \cos\beta_2\right) \tag{2.4}$$

Based on the definitions in Figure 2.1 the following relationships can be established

$$d = \frac{|\boldsymbol{r}_1 \times \boldsymbol{r}_2|}{|\boldsymbol{r}_0|} \qquad \cos\beta_1 = \frac{\boldsymbol{r}_0 \cdot \boldsymbol{r}_1}{|\boldsymbol{r}_0||\boldsymbol{r}_1|} \qquad \cos\beta_2 = \frac{\boldsymbol{r}_0 \cdot \boldsymbol{r}_2}{|\boldsymbol{r}_0||\boldsymbol{r}_2|}$$

In addition, the vector connecting the edges is defined as

$$\boldsymbol{r}_0 = \boldsymbol{r}_1 - \boldsymbol{r}_2$$

while the directional vector of the induced velocity is normal to the plane created between the point P and vortex segment edges and is given by

$$\frac{\boldsymbol{r}_1 \times \boldsymbol{r}_2}{|\boldsymbol{r}_1 \times \boldsymbol{r}_2|}$$

Substituting the new definitions into 2.4 and multiplying with the directional vector leads to a Cartesian interpretation of Biot-Savart law. The following procedure is thus applied to calculate the induced velocity at a point P by a line segment illustrated in Figure 2.1.

$$\boldsymbol{q}_{1,2} = \frac{\Gamma}{4\pi} \frac{\boldsymbol{r}_1 \times \boldsymbol{r}_2}{|\boldsymbol{r}_1 \times \boldsymbol{r}_2|^2} \boldsymbol{r}_0 \left(\frac{\boldsymbol{r}_1}{r_1} - \frac{\boldsymbol{r}_2}{r_2} \right) \tag{2.5}$$

In cartesian coordinates the above is solved by the following procedure. (u, v, w) are the induced velocities in (x, y, z) directions respectively. Where K is an influence coefficient and r_1 and r_2 are the distances towards the evaluated point.

$$(\boldsymbol{r}_1 \times \boldsymbol{r}_2)_x = (y_p - y_1)(z_p - z_2) - (z_p - z_1)(y_p - y_2) \tag{2.6}$$

$$(\boldsymbol{r}_1 \times \boldsymbol{r}_2)_y = (z_p - z_1)(x_p - x_2) - (x_p - z_1)(z_p - z_2) \tag{2.7}$$

$$(\boldsymbol{r}_1 \times \boldsymbol{r}_2)_z = (x_p - x_1)(y_p - y_2) - (y_p - y_1)(x_p - x_2) \tag{2.8}$$

$$|\boldsymbol{r}_1 \times \boldsymbol{r}_2|^2 = (\boldsymbol{r}_1 \times \boldsymbol{r}_2)_x^2 + (\boldsymbol{r}_1 \times \boldsymbol{r}_2)_y^2 + (\boldsymbol{r}_1 \times \boldsymbol{r}_2)_z^2 \tag{2.9}$$

$$\boldsymbol{r}_0 \cdot \boldsymbol{r}_1 = (x_2 - x_1)(x_p - x_1) + (y_2 - y_1)(y_p - y_1) + (z_2 - z_1)(z_p - z_1) \tag{2.10}$$

$$\boldsymbol{r}_0 \cdot \boldsymbol{r}_2 = (x_2 - x_1)(x_p - x_2) + (y_2 - y_1)(y_p - y_2) + (z_2 - z_1)(z_p - z_2) \tag{2.11}$$

$$K = \frac{\Gamma}{4\pi |\boldsymbol{r}_1 \times \boldsymbol{r}_2|^2} \left(\frac{\boldsymbol{r}_0 \cdot \boldsymbol{r}_1}{r_1} - \frac{\boldsymbol{r}_0 \cdot \boldsymbol{r}_2}{r_2} \right) \tag{2.12}$$

$$u = K \cdot (\boldsymbol{r}_1 \times \boldsymbol{r}_2)_x \qquad v = K \cdot (\boldsymbol{r}_1 \times \boldsymbol{r}_2)_y \qquad w = K \cdot (\boldsymbol{r}_1 \times \boldsymbol{r}_2)_z \tag{2.13}$$

2.2 Grid Generation

Grid generation is based on a cosine distribution in both chord wise and span wise direction, this is to minimize the required number of elements. As tests performed by (Fiddes and Gaydon, 1996) point out the accuracy of the solution increases. In addition the required number of elements for convergence decreases. The cosine distribution follows the form of equation 2.14 in both chord wise, x axis, and span wise, y axis, directions. Where c and s are respectively chord length, span length, while n and m are the number of elements in chord and span direction respectively. This section also determines the size and normal vector of each element.

$$x, y = c, s \cdot \left(1 - \cos\left(\frac{i \cdot \pi}{n+2}\right)\right) \tag{2.14}$$

The configuration of each element follows Weissinger's quarter-three-quarter-chord approximation. As in the leading bound vortex is placed on the elements quarter chord line, while the boundary condition is placed on the elements three-quarter chord line. This is due to the configurations ability to accurately solve boundary conditions (Faltinsen, 2005). The geometry created is a product of 5 parameters, span length, chord length, NACA profile and number of elements in chord and span wise directions and is presented in Figure 2.2.

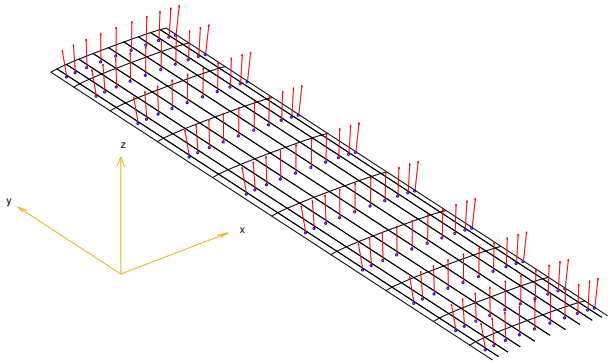


Figure 2.2: Result of the pre-processing detailing the elements, control points and normal vector.

2.3 Steady Solution

The model aims to solve a series of singularity elements distributed across the camber surface. The solution to the lifting problem is now found by finding the strength of each singularity element. The condition to be satisfied is the zero normal flow across the surface and is expressed on the right hand side of equation 2.15, which is equation 2.1 except neglecting the perturbation potential. Where $(U_\infty, V_\infty, W_\infty)$ are the free stream velocity components in (x, y, z) directions while n_N is the normal vector for element N.

$$a_N \cdot \Gamma_N = -[U_\infty, V_\infty, W_\infty] \cdot n_N \tag{2.15}$$

The left hand side of the equation consists of the singularity elements meant to solve the boundary condition, where Γ_N is the unknown vorticity at element N. The influence coefficient a_N is a matrix describing the influence each element had on the evaluated control point. The influence of a single element on a single control point is determined by equation 2.16, where $(u, v, w)_1$ is the induced velocity by segment $1-2$ in Figure 2.3 and so on going clockwise. Where the induced velocity follows Biot-Savarts Law as presented in section 2.1.

$$a_N = ((u, v, w)_1 + (u, v, w)_2 + (u, v, w)_3 + (u, v, w)_4) \cdot n_N \tag{2.16}$$

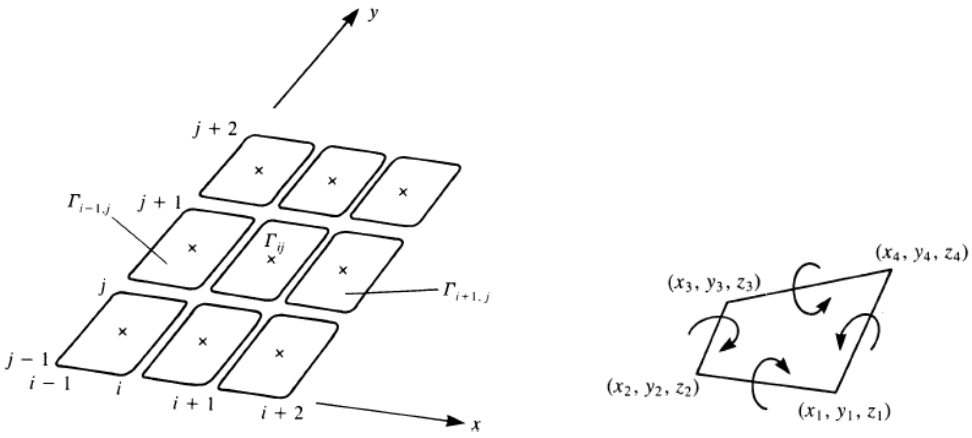


Figure 2.3: Arrangement of Vortex Rings and orientation (Katz and Plotkin, 2001).

Before the boundary condition can be solved the effect of the wake has to be considered, as well as any fluid boundary effects. Fluid boundaries in this case is the sea surface and if desired the sea floor. Both fluid boundaries are handled in a similar manner where the so called mirror image method is used, where the geometry is translated above the sea surface and the induced velocity from this mirror image is added to the already computed induced velocity as shown in Figure 2.4 and equation 2.17.

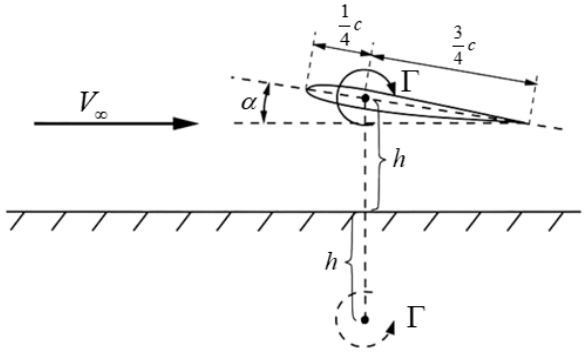


Figure 2.4: Modelling of fluid boundary by using the mirror image technique (Faltinsen, 2005).

$$(u, v, w) = (u, v, w)_{real} + (u, v, -w)_{imaginary} \tag{2.17}$$

The wake is modelled as force free and only effects the trailing edge as shown in Figure 2.5. Where a vortex ring with equal strength has been added to cancel the spanwise starting vortex. The induced velocity for the trailing edge is thus modified by equation 2.18. Where notation TE and W are respectively trailing edge and wake.

$$(u, v, w)_{TE} = (u, v, w)_{TE} + (u, v, w)_W \tag{2.18}$$

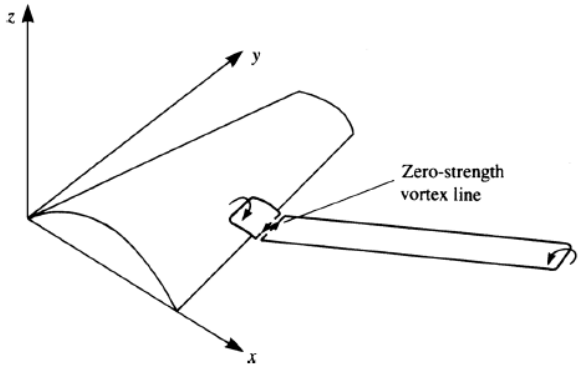


Figure 2.5: Method of attaching a vortex wake ring to fulfull the Kutta condition (Katz and Plotkin, 2001).

Once both the right hand side of equation 2.15 and the influence coefficients have been established only the strength of each element is unknown. As the problem has N number of unknowns and N number of equations the solution is found by standard matrix techniques as shown in equation 2.19. With the strength determined, the lift can be determined through Koutta-Joukowski theorem presented in 2.20. Where ρ is the fluid density, Q_∞ is the total free stream velocity and Δy is the width of each element.

$$\begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \vdots \\ \Gamma_N \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{11} & a_{12} & \cdots & a_{1N} \\ \vdots & & \ddots & \vdots \\ a_{1N} & a_{2N} & \cdots & a_{NN} \end{bmatrix}^{-1} \begin{bmatrix} [U_\infty, V_\infty, W_\infty] \cdot n_1 \\ [U_\infty, V_\infty, W_\infty] \cdot n_2 \\ \vdots \\ [U_\infty, V_\infty, W_\infty] \cdot n_N \end{bmatrix} \tag{2.19}$$

$$\Delta L_{i,j} = \rho Q_\infty \left(\Gamma_{i,j} - \Gamma_{i-1,j}\right) \Delta y_{i,j} \tag{2.20}$$

$$L = \sum_{i=1}^{M} \sum_{j=1}^{N} \Delta L_{i,j} \tag{2.21}$$

Drag forces follow a similar procedure except here influence coefficient is established by omitting the leading and trailing segment from equation 2.16. These vortex segments correspond to segments $1-2$ and $3-4$ in Figure 2.3. As the circulation is already known at this point, the induced downwash can be determined by the product of circulation distribution and influence coefficient. It is important to mention that this drag component is not the total drag, but only the induced drag due to downwash.

$$b_N = ((u, v, w)_2 + (u, v, w)_4) \cdot n_N \tag{2.22}$$

$$\begin{bmatrix} w_{ind_1} \\ w_{ind_2} \\ \vdots \\ w_{ind_N} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1N} \\ b_{11} & b_{12} & \cdots & b_{1N} \\ \vdots & & \ddots & \vdots \\ b_{1N} & b_{2N} & \cdots & b_{NN} \end{bmatrix} \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \vdots \\ \Gamma_N \end{bmatrix} \tag{2.23}$$

$$\Delta D_{i,j} = -\rho w_{ind_{i,j}} \left(\Gamma_{i,j} - \Gamma_{i-1,j}\right) \Delta y_{i,j} \tag{2.24}$$

$$D = \sum_{i=1}^{I} \sum_{j=1}^{J} \Delta D_{i,j} \tag{2.25}$$

2.4 Unsteady Solution

The unsteady solution requires changes to how the wake and resulting forces are modelled. The wake in the unsteady solution is made up of shed vortices from the trailing edge. Where the strength of said vortex is equal the vorticity at the trailing edge. Furthermore, the strength of these shed vortices are constant through time. The initial position of the shed vortices follows Kutta's condition, where it is stated that the flow leaves the foil tangentially (Anderson Jr, 2010). How these vortices move in the wake field is controlled by equation 2.26, where the velocity at every vortex is the total induced velocity from the wake field in addition to the induced velocity from the body bound vortices on the foil. Here the notation w is a vortex within the wake field, while b is a body bound vortex. The new position of vortices within the wake field is then determined by multiplying with a time step.

$$(u, v, w)_w = \sum_{b=1}^{B} (u, v, w)_b + \sum_{w=1}^{W} (u, v, w)_w \tag{2.26}$$

The wake now considers the perturbation potential in the boundary condition presented in 2.1 which was neglected in the steady solution. The numerical approximation of this is represented in equation 2.27. As the influence coefficient, a_N, is geometry based it is only evaluated once, thus for each time step the only new addition is determining the influence of the wake on the boundary condition. Solving for the strength of the singularity elements, Γ, leads to the vorticity of each element and subsequently the loads can now be determined.

$$a_N \cdot \Gamma_N = \left(\underbrace{\sum_{w=1}^{W} (u, v, w)_w}_{Perturbation} + \underbrace{(U_\infty, V_\infty, W_\infty)}_{Freestream} \right) \cdot n_N \tag{2.27}$$

The loads calculated still follow Kutta-Joukowskie's theorem, however now the change in circulation in time, $\frac{d\Gamma}{dt}$, is added. In addition, as for the boundary condition, the perturbation potential is added. Where Δx and Δy is an elements chordwise and spanwise length. In addition τ_x and τ_y represents a panels tangential vector in chord and span direction respectively. What follows is translating the pressure into a force and summing up all elements. Where S is a panels area, and $\Delta\vec{F}$ is a panels force vector in (x, y, z) directions.

$$\Delta p_{i,j} = \rho \left([U_\infty + u_w, V_\infty + v_w, W_\infty + w_w]_{i,j} \cdot \left(\tau_{x_{i,j}} \frac{\Gamma_{i,j} - \Gamma_{i-1,j}}{\Delta x_{i,j}} + \tau_{y_{i,j}} \frac{\Gamma_{i,j} - \Gamma_{i,j-1}}{\Delta y_{i,j}} \right) + \frac{d\Gamma_{i,j}}{dt} \right) \tag{2.28}$$

$$\Delta\vec{F} = S_{i,j} \cdot \Delta p \cdot n_{i,j} \qquad \vec{F} = \sum_{i=1}^{I} \sum_{j=1}^{J} \Delta\vec{F} \tag{2.29}$$

The induced drag has the same additional changes to the calculation. Furthermore, calculating the induced downwash at each element is unchanged from the steady solver. Hence the induced downwash follows equation 2.23 before calculating the induced drag in equation 2.30. Where α is the elements angle relative to the free stream.

$$\Delta D_{i,j} = \rho \left((w_{ind} + w_w)_{i,j} (\Gamma_{i,j} - \Gamma_{i-1,j}) \Delta x + \frac{d\Gamma_{i,j}}{dt} \Delta S_{i,j} \sin \alpha_{i,j} \right) \tag{2.30}$$

Chapter 3

Modelling Sailing Hydrofoils

The difference between hydrofoil and displacement vessels are the increased number of equilibrium states which have to be considered. For a displacement vessel, motions such as heave and pitch will naturally balance itself due to how the wetted surface of the vessel changes. Although this does not mean it is never a problem, as resonant oscillations can provoke large outbursts. For a foiling vessel the force balance is more similar to an airplane than displacement vessels. This is due to the forces in both aircraft and foiling vessels originate from the same physical principles, the mechanics behind foils.

The complexity of the problem is increased when considering wind powered hydrofoils, as now the vessel has a fluctuating force attempting to heel the vessel.

3.1 Kinematics

Reference frame notation for vessels moving in 6 degrees of freedom is summarised in Table 3.1 and visualised in Figure 3.1 (Fossen, 2011). Furthermore, there exist 2 coordinate systems for a marine vessel, an inertial reference frame and a body-fixed reference frame. As a general rule, velocities and forces are kept in the body-fixed reference frame while the position and orientation is kept in the inertial reference frame. A transformation between these two follows Euler's Theorem on Rotation and is described in equation 3.1. Where the body fixed velocity denoted with b transformed to the inertial frame denoted s. R is the rotation from body-fixed to the inertial frame expressed in equation 3.2, where Θ is the angular vector containing the roll, pitch and yaw angle. For simplicity c and s signify cosine and sine respectively.

DOF	Description	Force/Moment	Velocity	Position/angle
1	motions in x direction (surge)	X	u	x
2	motions in y direction (sway)	Y	v	y
3	motions in z direction (heave)	Z	w	z
4	rotation about the x axis (roll)	K	p	ϕ
5	rotation about the y axis (pitch)	M	q	θ
6	rotation about the z axis (yaw)	N	r	ψ

Table 3.1: Reference frame notation for marine vehicles.

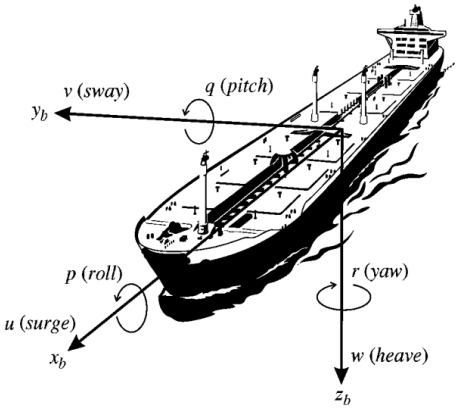


Figure 3.1: The 6 DOF velocities u, v, w, p, q and r in the body-fixed reference frame (Fossen, 2011).

$$\begin{bmatrix} u_s \\ v_s \\ w_s \end{bmatrix} = R_b^a(\Theta) \begin{bmatrix} u_b \\ v_b \\ w_b \end{bmatrix} \tag{3.1}$$

$$R_b^a(\Theta) = \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta \\ s\psi c\theta & c\psi c\phi + s\phi s\theta s\psi & -c\psi s\phi + s\theta s\psi c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{3.2}$$

3.2 Sailing

The total aerodynamic force produced by the sail, is regarded as a combination of lift and drag, and is denoted F_T. This force can be decomposed into a force in the direction of the course sailed defined as driving force, and a force normal to the course sailed defined as the heeling force. These are visualised in Figure 3.2 (Marchaj, 1979). For this project the centre of these forces will be constant and determined as the centroid of the sail area. However, this is an approximation and will depend on the trim of the sail, apparent wind speed and rotation of the vessel.

It is also important to mention the various terms used in sailing. Upwind sailing refers to when the wind direction is the opposite direction of the vessel direction. Downwind sailing is when the vessel moving with the direction of the wind. A jib is the foremost sail of a vessel while the mainsail is attached to the mast and boom of the vessel. A Tacking manoeuvre refers to a change in vessel direction such that the wind changes from port side to starboard side, or vice versa.

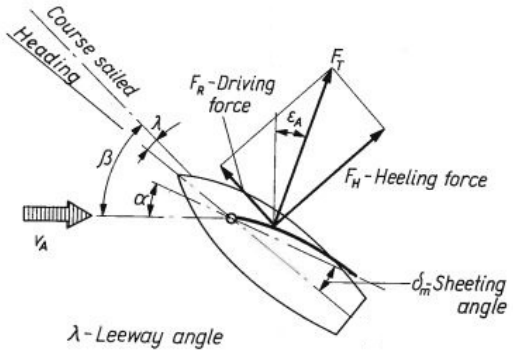


Figure 3.2: Relationship between driving force and heeling force (Marchaj, 1979).

$$F_R = L\sin(\beta) - D\cos(\beta) \qquad (3.3) \qquad F_H = L\cos(\beta) + D\sin(\beta) \qquad (3.4)$$

The force in direction of the course sailed and the force normal to this can be defined as in equation 3.3 and 3.4 respectively (Marchaj, 1979). Where β is the apparent wind angle, which is the wind felt by the vessel due to relativity, the relative wind strength and its angle are determined through equations 3.5 and 3.6. Where the apparent wind speed V_A is determined by the relativity between the direction of the vessel speed V_S, true wind speed V_T and its direction a as shown in Figure 3.3.

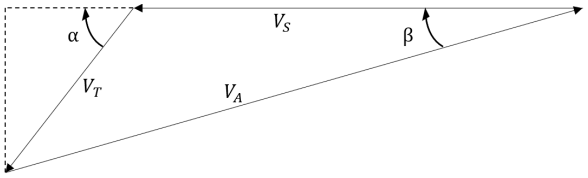


Figure 3.3: Velocity triangle and related angles (Marchaj, 1979)

$$V_A = \sqrt{V_T^2 + V_S^2 + 2V_T V_S \cos\alpha} \qquad (3.5)$$

$$\beta = \cos^{-1}\left(\frac{V_T\cos\alpha + V_S}{V_A}\right) \qquad (3.6)$$

The aerodynamic resistance of the hull can be regarded as 2 contributions, viscous effects and kinetic energy. The viscous effects is mostly friction along the hull and can be determined as shown in equation 3.7. where S is the surface of the hull and C_F is the frictional correlation line calculated using the ITTC-1978 prediction method. Where V_∞ is the relative velocity over the characteristic length L_{char} with viscosity υ. While the resistance due to the kinetic energy can be regarded as dynamic pressure on a flat plate. The formula in equation 3.8 is derived from Newtons Principa, where α is the angle of of the flow (Marchaj, 1979).

$$C_F = \frac{0.075}{(\log(R_N) - 2)^2} \qquad R_N = \frac{V_\infty L_{char}}{\upsilon} \qquad R_{air} = \frac{1}{2}\rho V_A^2 S C_F \qquad (3.7)$$

$$F_{AR} = \rho V_A^2 S \sin^2(\beta) \qquad (3.8)$$

3.3 Equation of Motion

The rigid body kinetics of a general marine craft is expressed in equation 3.9. Where η is a vector representing the motion of the vessel in all degreese of freedom with the $\dot{\eta}$ and $\ddot{\eta}$ notation signifies velocity and acceleration respectively. While M and C are respectively the mass and centripetal matrices of the system and are represented in equation 3.10 and 3.11. Where S is a skew matrix and the mass matrix is separated into four equal parts of size 3×3.

$$\tau = M\ddot{\eta}_b + C\dot{\eta}_b \tag{3.9}$$

$$M = \begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & I_x & -I_{xy} & -I_{xz} \\ mz_G & 0 & -mx_G & -I_{yx} & I_y & -I_{yz} \\ -my_G & mx_G & 0 & -I_{zx} & -I_{zy} & I_z \end{bmatrix} \tag{3.10}$$

$$C = \begin{bmatrix} 0_{3\times3} & -S(M_{11}v_b + M_{12}w_b) \\ -S(M_{11}v_b + M_{12}w_b) & -S(M_{21}v_b + M_{22}w_b) \end{bmatrix} \tag{3.11}$$

To determine the velocities of the next time step, equation 3.9 is solved with respect to acceleration directly before using Euler's method to determine the new velocities. This is shown in equations 3.12 and 3.13 respectively. Where t is a time step counter and Δt is the time step.

$$\ddot{\eta}_b = M^{-1}(\tau - C\dot{\eta}_b) \tag{3.12}$$

$$\dot{\eta}_b(t+1) = \dot{\eta}_b(t) + \ddot{\eta}_b \Delta t \tag{3.13}$$

Chapter 4

The Simulation Model

This section details the implemented model. First an explanation of the assumptions and neglections are presented, then an overview of the model as a whole is presented before going into the different routines which make up the model.

4.1 Physical Assumptions and Neglections

The jib of the vessel, i.e foremost sail, is neglected, thus only the mainsail of the vessel will be considered. Superstructure of the vessel will not be considered, the deck of the vessel is assumed to be a rigid flat plate. This also holds true for the mast and boom of the vessel. The mainsail of the vessel is assumed to be rigid and stiffened as to simulate a non-aero-elastic sail thus luffing will not occur, i.e flapping back and forth.

It is assumed that the current in the ocean is zero thus the incoming flow is equal and opposite the direction of the motion of the vessel. Furthermore, the sea state is assumed to be calm-water, thus the effect of waves will be neglected. The wind will be assumed to be of a constant value with no gusts.

Only foil-borne condition will be evaluated, meaning the simulation terminates if the hull is in the water. The hydrofoils will not suffer from cavitation and ventilation. In addition foil interaction effects are neglected, thus all hydrofoils will be evaluated independently from each other. The centre of the lift and drag produced by the hydrofoils is assumed to be at the midpoint of the span.

4.2 Overview of the Model

Figure 4.1 is a schematic of how the model operates, the required input which are simulation, vessel, foil and sail settings are summarised in Table 4.1.

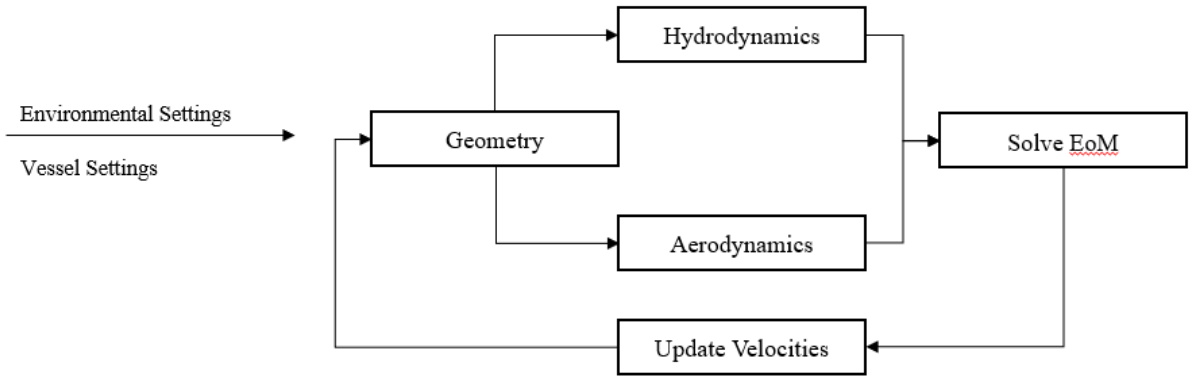


Figure 4.1: Schematic of the overall model.

Simulation	Vessel	Foil	Sail
True Wind Speed	LOA	Chord	Min - Max Chord
True Wind Angle	Beam	Span	Span
Initial Vessel Speed	Initial Airgap	NACA	NACA
Initial Vessel Direction	Depth	Material Density	Material Density
Initial Vessel Orientation	C_B	Angling	
	C_M	Position For / aft	
	C_P		
	Weight		
	Nr of Crew		
	Crew Weight		
	Crew Position		

Table 4.1: Necessary parameters to run the model.

First of the geometry is modelled based on section 2.2 and 4.3 and translated between the body-fixed and inertial reference frame as shown in section 3.1. This section also determines the moment arms of the different contributors and how much of each hydrofoil is submerged. This is done by modelling the foil as a straight line and finding the point where the line pierces the sea surface, which is modelled as a flat plane.

With the geometry modelled the forces can be determined which are split into forces resulting from hydrodynamics and aerodynamics. Both the hydrofoils and sail use the same lifting surface presented in section 2 to determine the lift and drag. However, the aerodynamical component uses the principles behind section 3.2 to determine the final force derived from a sail. Resistance components on the hull also follow the expressions presented in section 3.2. Before the total force and torque in each direction is calculated the forces are transformed to account for the vessel orientation.

To simulate the crew of the vessel, the helmsman is assumed fixed while the other is free to move across the deck. This is to simulate the crew adjusting their position to account for the heeling and pitching of the vessel. This is done by finding the most optimum position, which is not exactly feasible but justifiable.

Now that the total force and torque acting on the vessel is determined the equations of motion can be solved as presented in section 3.3. Eulers method is then used to determine the velocities of the next time step. Now the geometry is updated based on translational and angular velocities and the loop continues. The simulation continues until either a desired time has been reached or if the vessel is no longer foil-borne.

4.3 Geometry

The vessel is, geometrically speaking, represented as a series of points. For the hull, these points are placed equally along the length and in the middle of it's cross-section. The surface area of the hull, required for aerodynamical resistance is determined from equation 4.1. Where L is length, B is beam, D is depth while C_B, C_P and C_M is the block , prismatic and midships coefficient respectively. It is important to mention that this is not an accurate method and will overestimate the area, as such it was decided to modify it by 0.6.

$$S_A = 2 \cdot (LBC_P + BDC_M + LDC_B) \cdot 0.6 \tag{4.1}$$

The vessel modelled can be described as several independent elements with their own mass distribution. Where each element is modelled as a point mass distribution. For the hydrofoils, mass distribution is centred and equally spaced along the quarter chord line across the camber line. Where the mass is determined by the NACA profile along with the specified material density, chord and span. The sail follows the same principle as the hydrofoils. The hull is assumed to have a mass distribution along its longitudinal axis and proportional to its cross sectional volume. The end product of such an arrangement can be viewed in Figure 4.2, where x are the foils, $\triangle$ is the hull and $\triangledown$ is the sail.

The mass moment of inertia is determined as shown in equation 4.2, where x, y and z is the distance from the centre of gravity to the various point masses and m is the mass of the point mass.

$$\vec{I} = \sum_{i=1}^{I} \begin{bmatrix} m_i * (y_i^2 + z_i^2) & -m_i \cdot x_i \cdot y_i & -m_i \cdot x_i \cdot z_i \\ -m_i \cdot x_i \cdot y_i & m_i \cdot (x_i^2 + z_i^2) & -m_i \cdot y_i \cdot z_i \\ -m_i \cdot x_i \cdot z_i & -m_i \cdot y_i \cdot z_i & m_i \cdot (y_i^2 + x_i^2) \end{bmatrix} \tag{4.2}$$

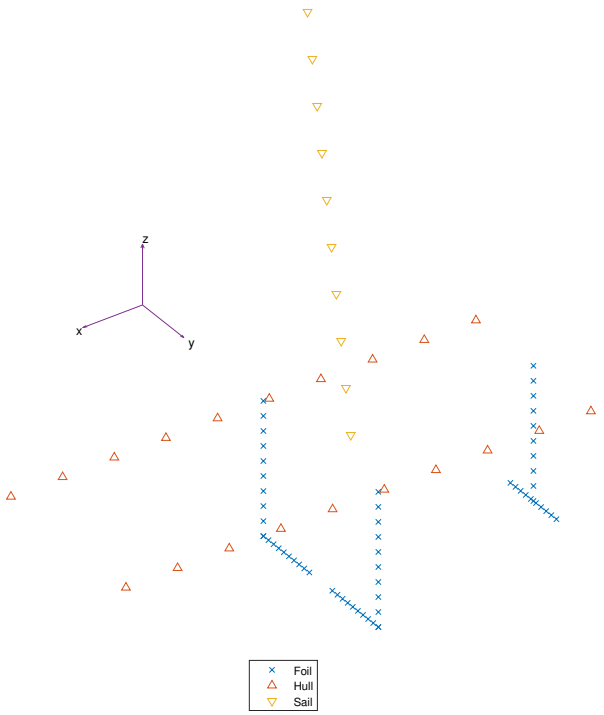


Figure 4.2: Mass distribution of the sail boat.

To determine the submerged section of a hydrofoil an intersection point between a line and plane is found. The foil is represented by two points (x_1, y_1, z_1) and (x_2, y_2, z_2) where point 1 is above the free surface. This creates a line through the free surface modelled as a plane with normal vector $n = [0, 0, 1]$ and the point $V = [1, 1, 0]$ on it, the intersection is then found as performed in equation 4.3. The zero in V makes it so the free surface is modelled at $z = 0$. An example of this is shown in Figure 4.3. The force originating from these hydrofoils are assumed to be at the midpoint of the foil, and as such the moment arm is determined from this point. This will of course not be valid when entire foil is below or above the free surface. However as the simulation terminates once this occurs it is not of consequence.

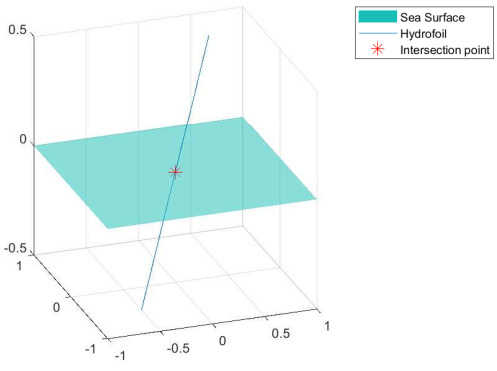


Figure 4.3: Plane-line intersection method.

$$(x, y, z) = (x_1, y_1, z_1) - \frac{n \cdot ((x_1, y_1, z_1) - V)}{n \left((x_2, y_2, z_2) - (x_1, y_1, z_1)\right)} \cdot ((x_2, y_2, z_2) - (x_1, y_1, z_1)) \quad (4.3)$$

4.4 Hydrodynamics

Due to the scope of the work, added mass has been included however the estimation is not a numerical method and based on an elliptical shape. Therefore the equation of motion presented in equation 3.9 changes to equation 4.4. The added mass matrix is presented in equation 4.5 and due the symmetrical shape most components can be neglected, while $m_{22} = m_{33}$ and $m_{55} = m_{66}$. The estimation of the different components is listed beside. The estimation is based on slender body theory and is taken from (Lewis, 1989), where the 2 dimensional cross section is integrated over the length where a and b are characteristical lengths of an ellipsoid, which in this case are the span and chord respectively.

$$\tau = (M + M_a)\ddot{\eta} + C\dot{\eta} \tag{4.4}$$

$$M_a = \begin{bmatrix} m_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & m_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & m_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & m_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & m_{55} \end{bmatrix} \tag{4.5}$$

$$m_{11} = \int_L \rho\pi b^2$$
$$m_{22} = \int_L \rho\pi a^2$$
$$m_{55} = \int_L \rho(a^2 - b^2)$$

The lifting surface model works as explained in section 2. Ideally, the steady state solution would only be determined once. Due to the influence matrix is dependent on the geometry and not the wake. However, as the vessel is not fixed the geometry and proximity to the sea surface will change, thus the routine presented in section 2.3 has to be completed for every time step. Even though it is not dependent on time, the parameters which are used change. This is most important for surface piercing foils as they will have a varying length.

Furthermore, the incoming flow felt by the hydrofoils is equal and opposite direction of the vessel motion. This is illustrated in Figure 4.4 where heave and surge velocity is shown and the incoming velocity is in the opposite direction. This is valid as there is assumed to be no current or waves, it is valid as the incoming flow of the hydrofoils is the relative velocity between the fluid and vessel motion.

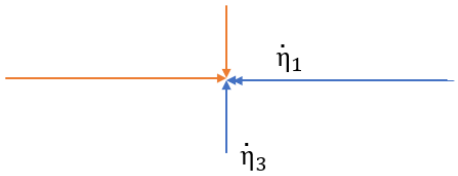


Figure 4.4: Incoming flow felt by the hydrofoils due to the vessels motion.

4.5 Aerodynamics

Aerodynamical forces are determined as presented in section 3.2 where the lift and drag is based on the lifting surface model presented in section 2. The aerodynamical added mass is neglected due the density of air is substantially lower than water. It is known that the air velocity increases farther from the ground, having the same gradual increase as in a boundary layer. As such the incoming flow follows such a gradual increase in altitude reaching a maximum at the top of the mast.

- *Sail Trim - Roll Control*

During the implementation of the model, it was noticeable that the simulation failed most often due to the roll angle. As such a primitive control device was created to simulate the crew trimming the sail to compensate for this increasing angle. The control module works by adjusting the boom angle when a certain roll angle has been reached. The boom angle was chosen as a change in angle of attack gives the most predictable outcome and the implementation requires the least amount of programming effort. Furthermore, as these changes occur instantly it is easier to justify the change in angle of attack then the effect of a sudden decrease in sail area.

The reason roll angle was chosen as a criteria and not its total torque, velocity or acceleration, is that the vessel requires time to slow down the roll motion. This means the total torque needs to continuously have changed sign, which would lead to a continuous change in sign of acceleration thus slowing the velocity and ultimately bringing the vessel back up.

4.6 Matlab

The model has been implemented in Mathworks Matlab which is a proprietary programming language. The reason is its advantages in matrix manipulation, implementation of algorithms and user-friendliness. An effort was made to matrices most of the calculations as most operations are either of a "loop" type nature or matrix manipulation.

Chapter 5

Validation and Verification

This section entails an investigation into the lfitng model. First a convergence test is performed before verifying whether different geometrical aspects are considered by the model. The results are validated against empirical and analytical means.

Before delving into the validation is it important to understand the term NACA. NACA and its digits are a way of describing the sectional shape of a foil. The designation of the digits follows as such: NACA MPXX, NACA LPQXX, where M is the maximum camber, P is the position of the maximum camber along the chord and XX is the thickness. The values are represented as percentages of the chord length.

Figure 5.1 is a convergence test to determine the optimum amount of elements as well as most optimum arrangement. The convergence is also plotted against experimental data from (Abbot et al., 1945) and checked against a theoretical value based on Helmbold's lift slope (Chattot and Hafez, 2015). In Figure 5.1 n and m refer to the number of elements along the chord and span respectively. It is noticeable that the solution does not change by much after 1000 elements, where the difference between the 1000 mark and final value is 0.09%. As the difference is negligible, the only tangible effect of increasing the number of elements is increasing the computational effort.

The discrepancy between the numerical model and experimental is most likely due viscous effects. As the numerical model is potential based it does not consider the effects of boundary layer or separation. With regard to the boundary layer, (Melnik et al., 1977) suggested that the boundary layer accounts for approximately 50% of the viscous effect on lift. This is more noticeable in the coming sections.

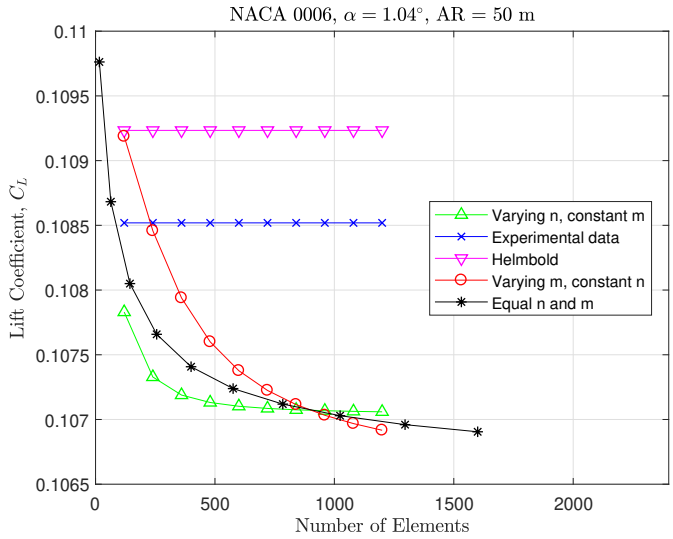


Figure 5.1: Convergence test, difference between constant number of cells in chord and span direction.

5.1 Effect of Aspect Ratio

An analytical lift slope is tested against the model to quantify the models ability to capture the effects of aspect ratio. The theoretical formulation used is Helmboldz formulae for any aspect ratio, it is used due to its simplicity and accuracy. The formulae is expressed in equation 5.1.

$$C_{L\alpha} = \frac{\pi AR}{\left(1 + \sqrt{1 + \frac{AR}{2}}\right)^2} \tag{5.1}$$

Figure 5.2 depicts the effect of aspect ratio on the lift curve slope, here it is quite noticeable that the aspect ratio has a clear effect. The effect is largest until an aspect ratio of 20 which coincides with the studies by (Abbot et al., 1945). The difference between the theoretical and numerical lift slope is plotted alongside and it is reassuring to see the difference is within the region of $1 - 2\%$.

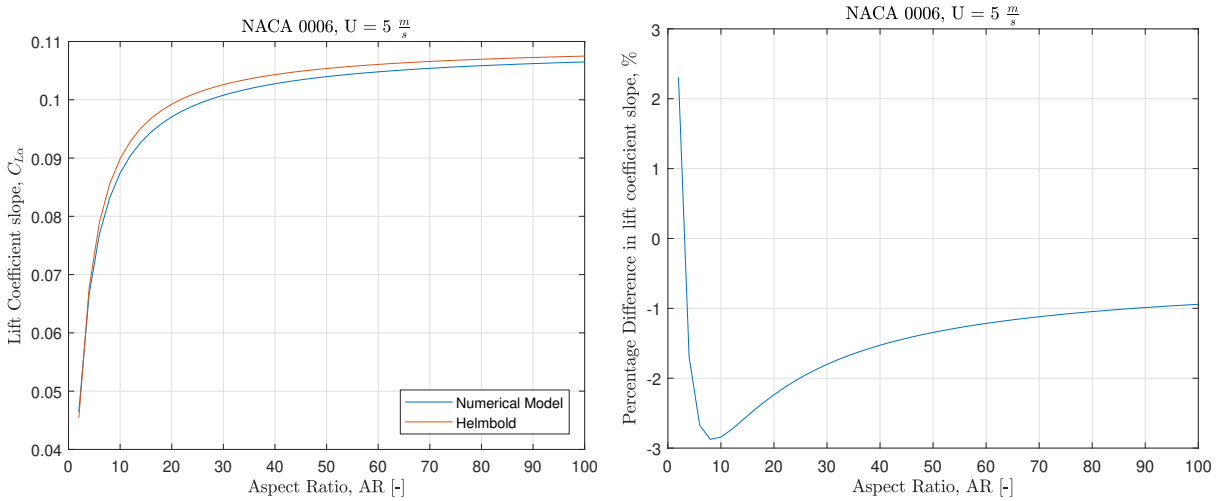


Figure 5.2: Effect of the aspect ratio on the lift coefficient.

Figure 5.3 is the effect on drag coefficient, here it is quite visible that the drag modelled is far from correct. The largest contributor to the difference is most likely viscous effects. The model does not consider the no-slip condition, and can be said to use half the boundary condition superimposed on the camber surface. As the boundary condition to be solved is in actuality "that the normal component of the relative velocity between the fluid and the solid surface is zero on the boundary" (Anderson Jr, 2010). Thickness effects would also contribute to the drag in the form of Form drag. In addition, it is important to remember the model only considers the induced drag. However, even though the magnitude is not correct, the form of the curve is correct as well as the effect aspect ratio has on the drag coefficient.

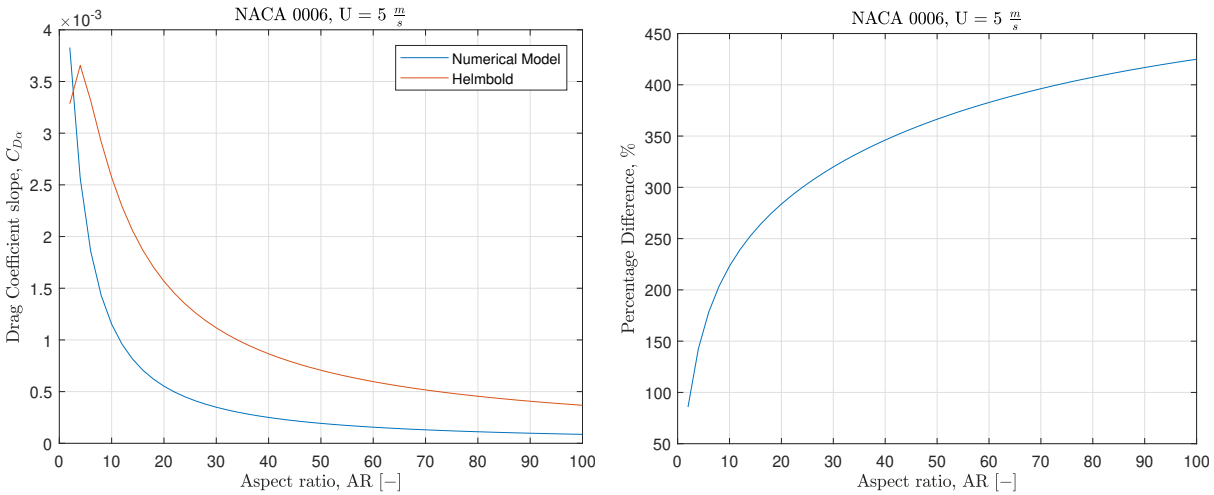


Figure 5.3: Effect of the aspect ratio on the drag coefficient.

5.2 Effect of Camber

To verify whether the model accounts for camber accurately several different profiles have been selected. The profiles selected are based on the availability of experimental data, the data presented is taken from (Abbot et al., 1945). To reduce the uncertainty between the experimental data and the model, the aspect ratio used is of infinite nature which corresponds to sectional lift. Based on the figures below it can be concluded that the numerical model sufficiently accounts for the effects of camber. Even though it is not perfect, the model is only overestimating within a range of $2 - 10\%$ during attached flow conditions. However, expectedly the model grossly overestimates once stalling occurs. In addition, for increasing camber the model would seem to be overestimating the lift, this is based on Figures 5.5 through 5.7.

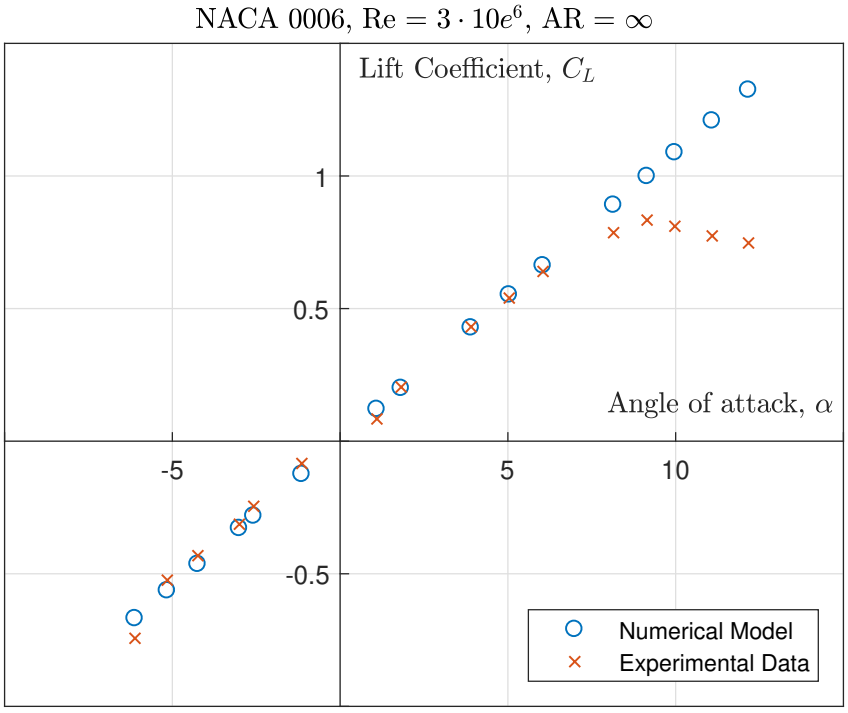


Figure 5.4: Comparison of Lift Coefficient data for NACA 0006 (Abbot et al., 1945).

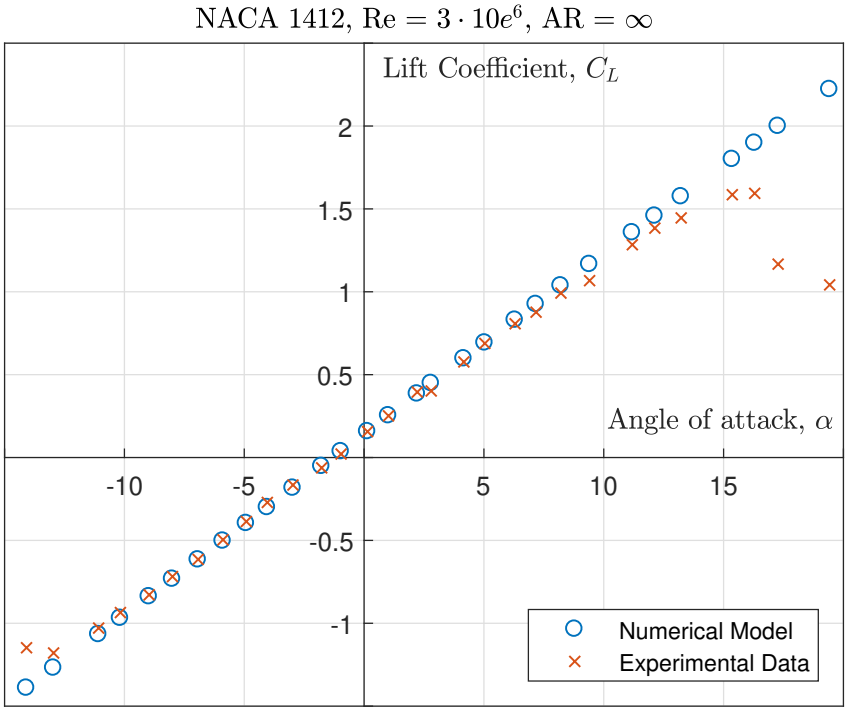


Figure 5.5: Comparison of Lift Coefficient data for NACA 1412 (Abbot et al., 1945).

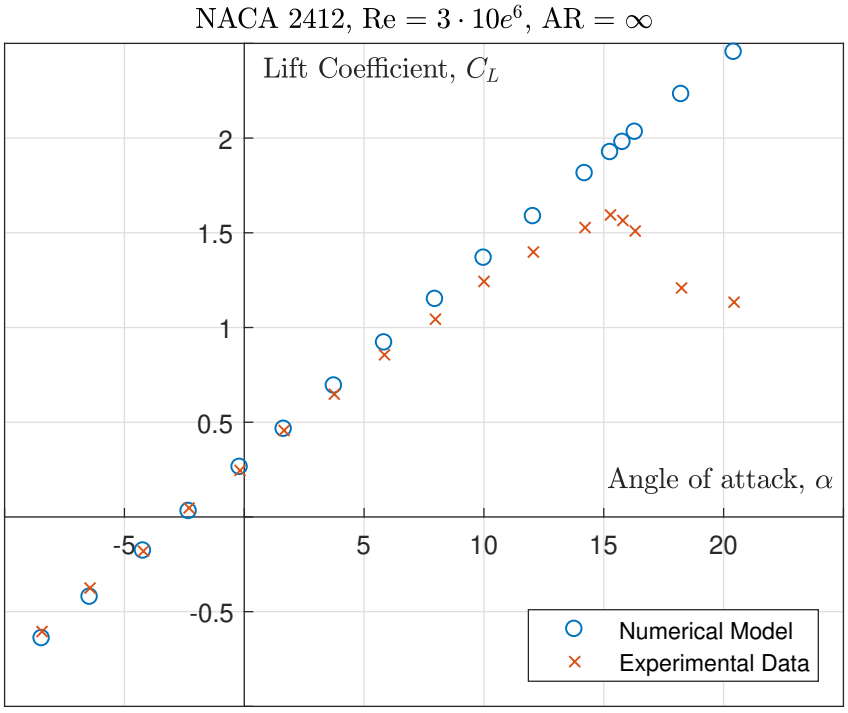


Figure 5.6: Comparison of Lift Coefficient data for NACA 2412 (Abbot et al., 1945).

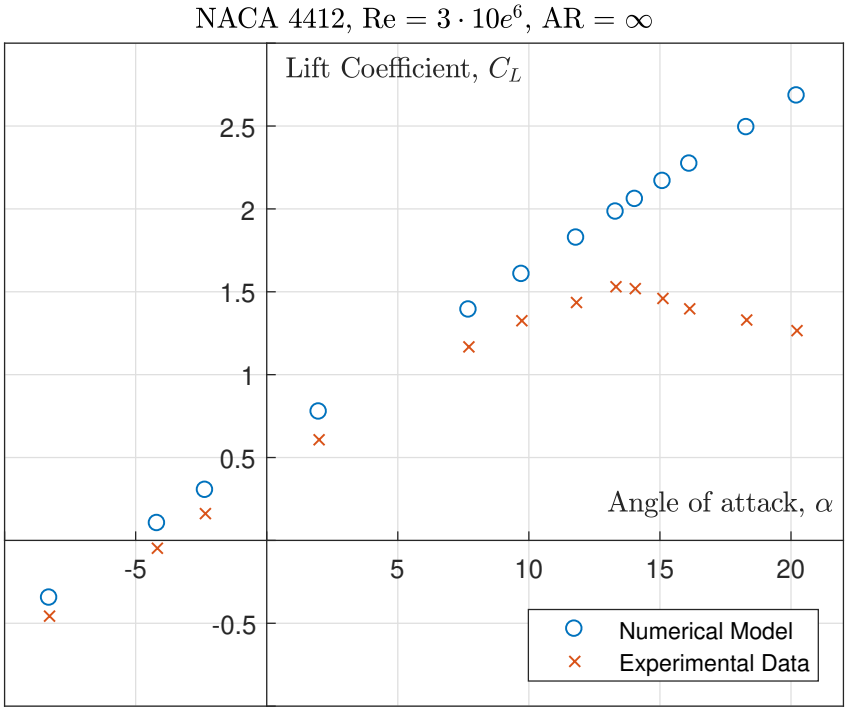


Figure 5.7: Comparison of Lift Coefficient data for NACA 4421 (Abbot et al., 1945).

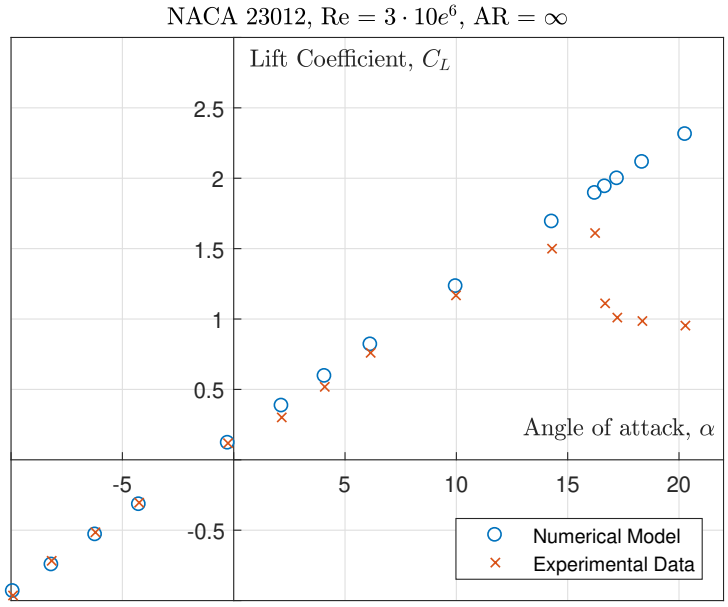


Figure 5.8: Comparison of Lift Coefficient data for NACA 23012 (Abbot et al., 1945).

5.3 Wake

To check whether the behaviour of the wake has been implemented correctly pictures from real life situations has been used. Figure 5.9 presents an aircraft moving through died gas to visualise the wake of an aircraft (NASA, 1990) and the numerical models estimation of the wake. The model shows a good approximation of the wake shape compared to the real life situation. However, the model would be more accurate if it considered what type of fluid the foil is enveloped in. At the moment the wake only considers the strength and position of a vortex.

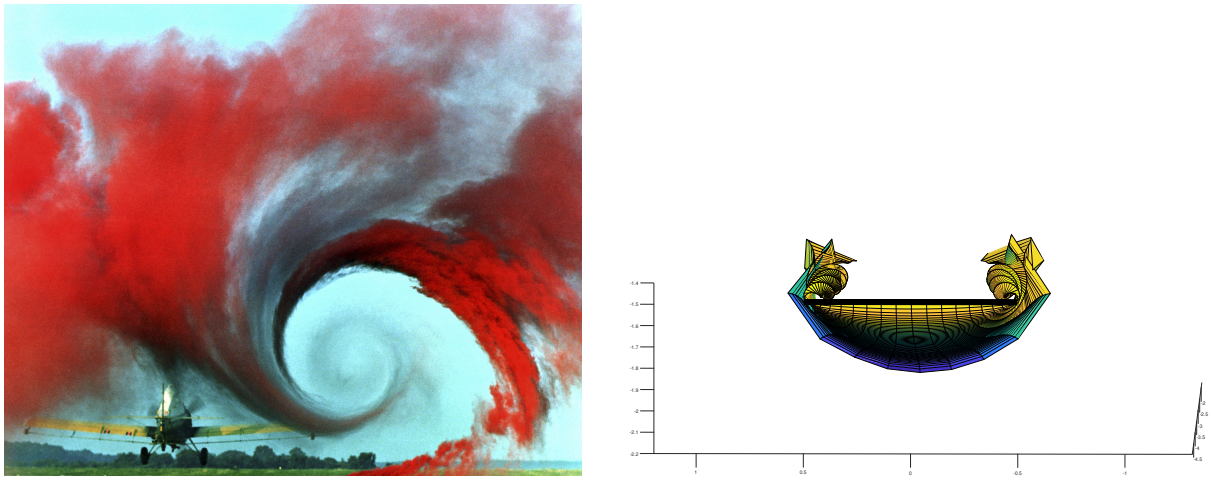


Figure 5.9: Comparison between a wake turbulence study by NASA and the wake created by the numerical model (NASA, 1990).

While implementing the model, it was noticeable that the computational cost of the wake increased with each shed vortex. It is known that a shed vortex has a decreasing impact on the foil with increasing distance from said foil. As such it was decided to test whether removing the wake at a certain distance away from the foil would impact the final results.

Figure 5.10 is a comparison between modifying the wake with a roll up and no roll up. It is noticeable that converged solution is the same, however the onset of the simulation is different. This is most likely due to the decreasing importance of the wake with increasing distance travelled. In addition, the peak behaviour is contributed towards faulty initial conditions. In reality, the wake would be built up as the foil is accelerating until a constant velocity. Where as the simulation disregards what lead to the constant velocity.

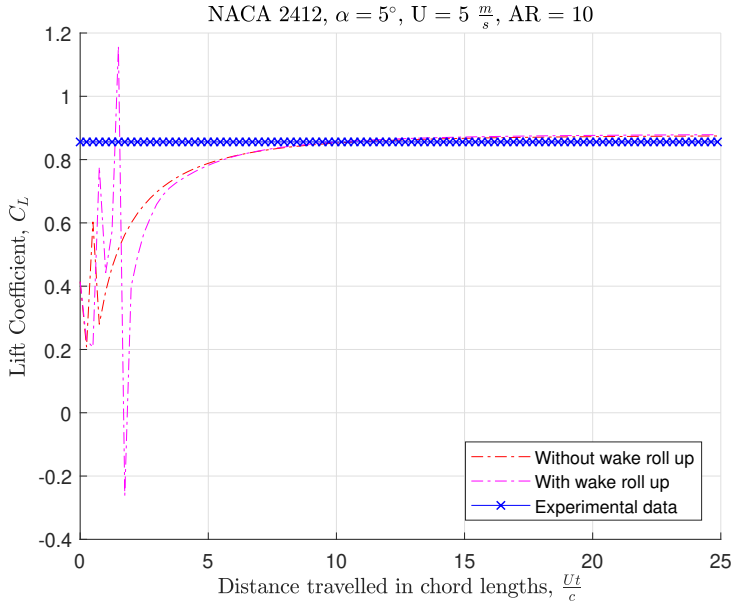


Figure 5.10: Effect of the wake roll up on the dynamic lift.

Based on the convergence of the unsteady solution after 10-15 chord lengths in Figure 5.10 it was decided to test whether removing parts of the wake would have a substantial effect. The motivation for this is to reduce the computational time, as the wake is an exponential contributor with increasing time step. Figure 5.11 depicts this test with no wake roll up. Here its noticeable that the effect of the far field wake is very small after 10 chord lengths and negligible after 15.

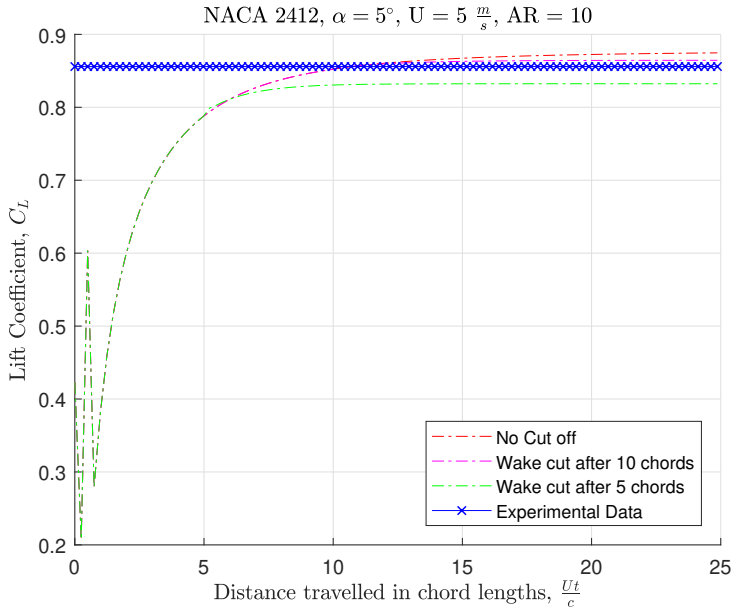


Figure 5.11: Comparison between different wake lengths.

The fluctuating lift coefficient at the onset of the simulation in Figure 5.10 and 5.11 is most likely the result of poor initial conditions. As the foil is suddenly put into constant forward speed acting as if the acceleration of the foil was near infinite from $t = 0$ to the first time step.

5.4 Forced Oscillations in heave.

To validate whether the unsteady solver is correct or not, the foil is forced to oscillate in heave. Theodorsen's function is applied due to its robust and reliable analytical formulation to the unsteady problem (Faltinsen, 2005). Figure 5.12 shows the difference between the unsteady solver and Theodorsen's analytical approach.

Neglecting pitch, Theodorsen's formulation is expressed in equation 5.2, where Theodorsen's function is $C(k_f)$ and expressed in equation 5.3, where F and G are the respectively the real and imaginary parts of the function. As the numerical model is potential flow based, it was decided to keep the oscillation as smooth as possible and at a low amplitude. The main reason is to keep the angle of attack relatively low as to avoid regions were viscous effects become important. The two solutions coincide both in shape and magnitude, however peak difference is at 20%, with a root mean squared difference of 0.14. Different frequencies were tested and the difference seen in Figure 5.12 is consistent for all tests which comply with potential parameters.

$$L = -\rho 0.25\pi c^2 \ddot{h} + \rho\pi U c\omega G(k_f)h - \rho\pi U c F(k)\dot{h} \tag{5.2}$$

$$C(k_f) = F(k_f) + iG(k_f) = \frac{H_1^{(2)}(k_f)}{H_1^{(2)}(k_f) + iH_0^{(2)}(k_f)} \tag{5.3}$$

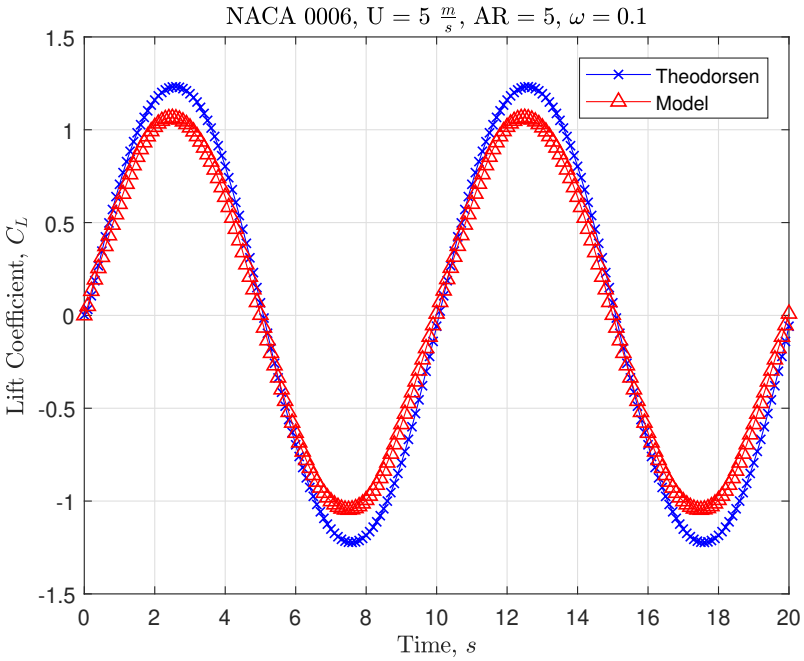


Figure 5.12: Theodorsen's theoretical solution and the unsteady solvers solution to an oscillating foil.

Chapter 6

Results

The results presented in this section is based on a vessel as described in Tables 6.1, 6.2 and 6.3, which has drawn inspiration from Figure 6.1. It is important to mention that the 4 DOF simulations in this section stop due to a zero tolerance of hull in water.

Figure 6.1: The L'Hydroptere vessel (Sheahan, 2009).

Initial conditions for the results below are as follows; The vessel has no acceleration, the vessel is only moving in x direction, initial vessel speed is $7\frac{m}{s}$.

Vessel	LOA	Beam	Depth	Airgap	C_B	C_M	C_P	Weight	Sail chord	Sail Span
	6 m	2 m	0.5 m	1 m	0.6	0.5	0.5	250 kg	0.2 - 0.75 m	4.5 m

Table 6.1: Vessel Parameters.

Foils	NACA	Span	Chord	Angling	Position (x,y,z)
Dihedral	2412	4 m	0.4 m	60	(0.4, 2.5, 0.0) m
Rudder	0012	2 m	0.4 m	0	(-3.0, 0.0, 0.0) m
Rudder T-foil	0012	1 m	0.4 m	0	(-3.0, 0.0, -1.0) m

Table 6.2: Foil Parameters, position is relative to the centre of gravity.

Crew	Size	Weight
	2	80 kg

Table 6.3: Crew Parameters.

6.1 1 DOF Stability Tests

The following simulations are performed to determine whether the vessel is stable. As such all degrees of freedom except the testing parameter are kept constant. The constant parameters used are presented in Table 6.4. The simulation was only performed for true wind speeds of $6\frac{m}{s}$ with a true wind angle of 25°.

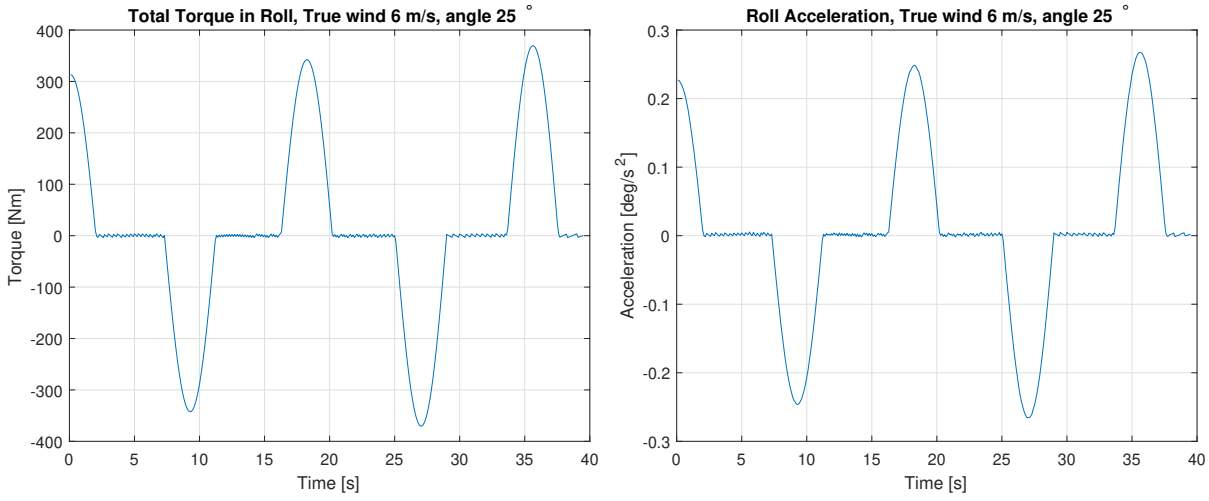
u	v	w	p	q	r
$7\ \frac{m}{s}$	$0\ \frac{m}{s}$	$0\ \frac{m}{s}$	$0\ \frac{deg}{s}$	$0\ \frac{deg}{s}$	$0\ \frac{deg}{s}$

Table 6.4: Constant Parameters used in the 1 DOF simulations.

Roll Stability

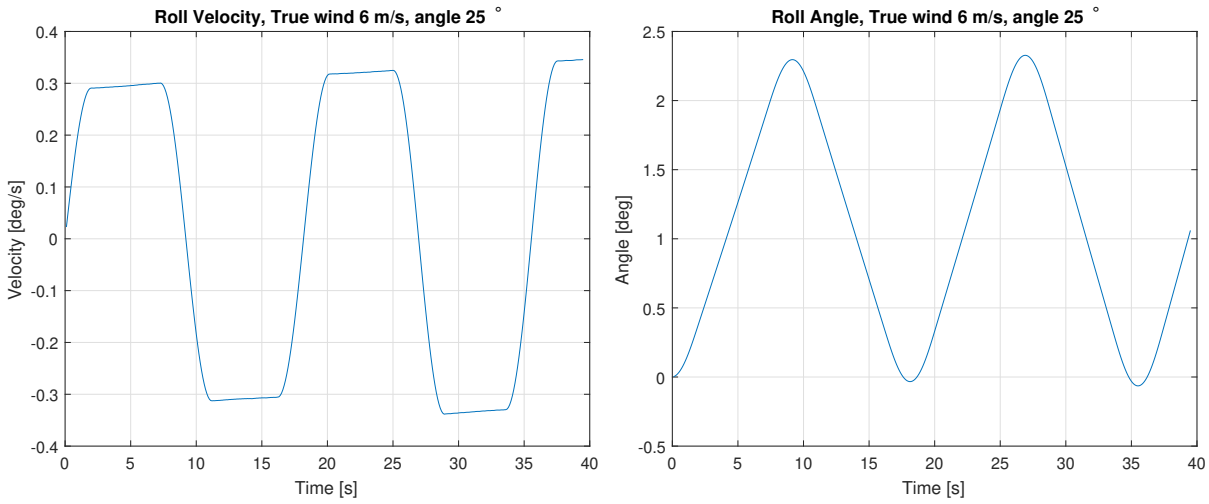
The sail trim of the vessel is allowed to be adjusted to compensate for the increasing roll angle, this is to simulate the crew applying countermeasures to the motion. The simulation was terminated after 2 oscillations, as it was deemed sufficient enough to view the roll stability.

The decline in total torque in Figure 6.2a is contributed to an increasing submerged starboard foil and a decreasing submerged port foil. Thus the force balance between these two becomes greater, allowing the starboard foil to counteract the torque from the sail until it reaches an equilibrium. However, as the acceleration in roll has only been positive seen in Figure 6.2b until 7.5 seconds there exists now a constant roll velocity as seen Figure 6.3a between 2.5 and 5 seconds heeling the vessel. This continues until the sail is de-powered at a roll angle of 2° resulting in a deceleration of the roll velocity. The de-powering works as intended, however it might be contributing to an unstable solution. This is based on the peak roll velocity in Figure 6.3a is increasing. Although the increase is small it can be seen that peak roll angle is increasing in Figure 6.3b. Therefore given enough time the results may indicate an unstable vessel.



(a) (b)

Figure 6.2: 1 DOF Roll Stability test, a - Total Torque, b - Acceleration, over 40 seconds.

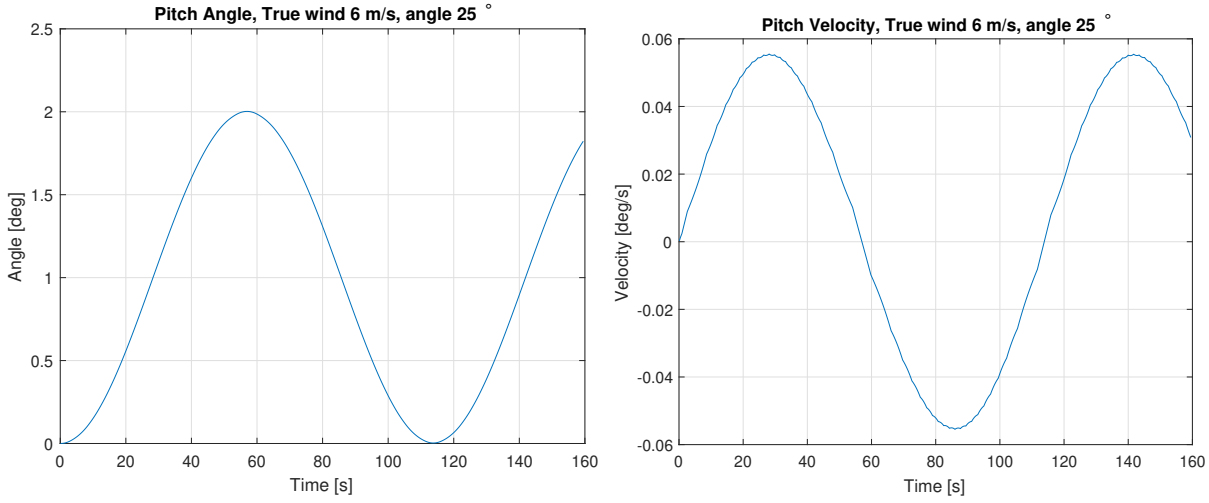


(a) (b)

Figure 6.3: 1 DOF Roll Stability test, a - Velocity, b - Angle, over 40 seconds.

Pitch Stability

As can be seen in Figure 6.4a the pitch motion is severely slower then roll motion in Figure 6.3b, thus requiring a longer simulation. Although very small the jagged behaviour in Figure 6.4b is due to the choice of allowing one crew member to move up and down the cockpit of the vessel to compensate for pitching. This is a gross simplification, however it is noticeable that the effect has had the intended effect of minimizing pitch motion.



(a) (b)

Figure 6.4: 1 DOF Pitch Stability test, a - Angle, b - Velocity, over 160 seconds.

Heave Stability

The heave motion illustrated in Figure 6.5 shows a quick and clear convergence towards equilibrium. Where the reason is due to the total submerged area reduces as the vessel lifts itself upwards thus reducing the total lift upwards. In addition to the reduced submerged area, as the vessel is moving upwards the angle of attack turns negative as the relative velocity of the fluid over the hydrofoils is now negative thus reducing the lift. Although based on the heave velocity magnitude in Figure 6.5c the effect of a negative angle of attack would be significantly lower than the reduced submerged area which decreases from 0 - 12 seconds.

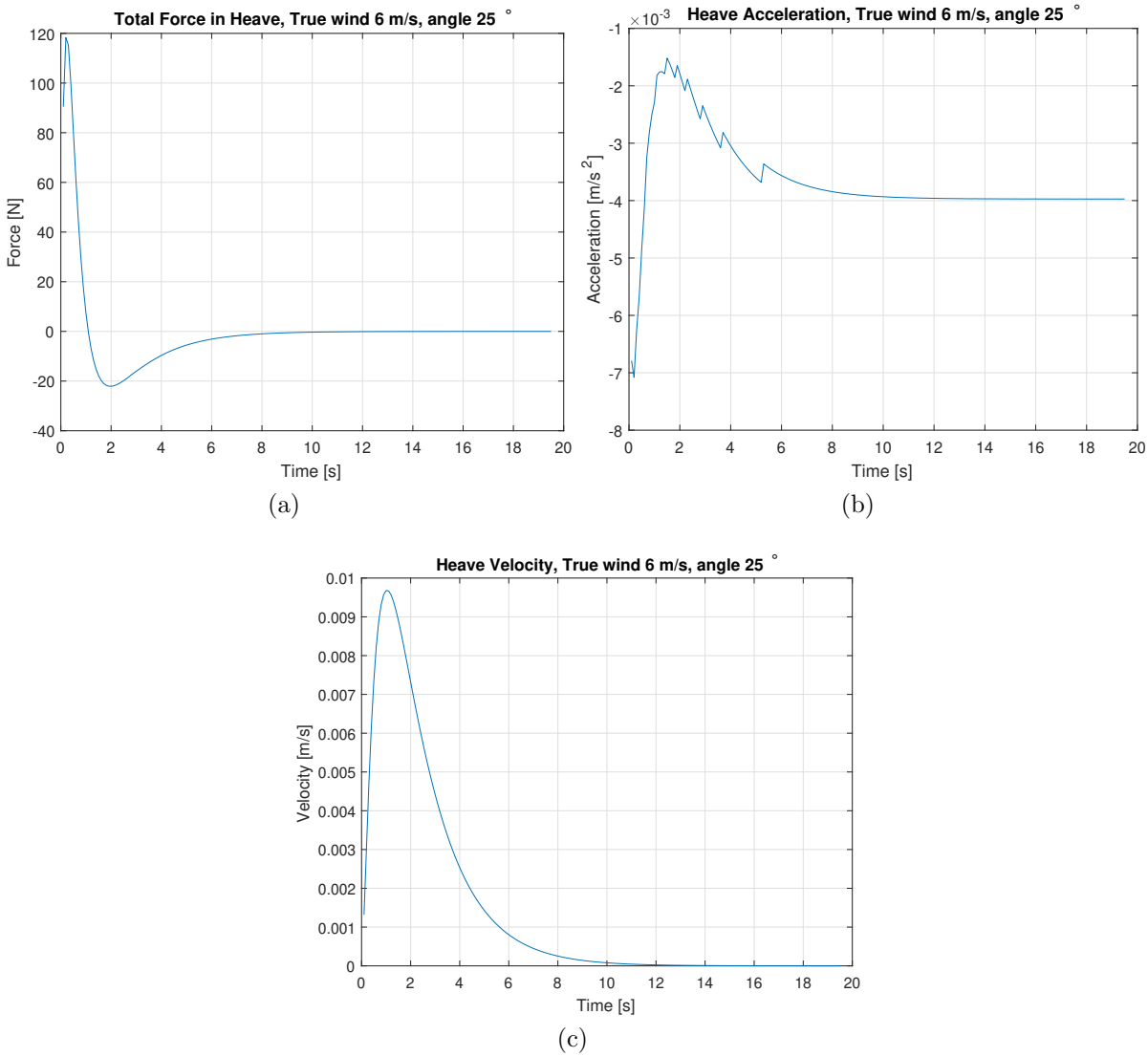


Figure 6.5: 1 DOF Heave Stability test, a - Total Force, b - Acceleration, c - Velocity, over 20 seconds.

6.2 4 DOF Simulation

The results in this section neglect sway and yaw motion. The initial conditions of the vessel are equal to the 1 DOF simulations and presented in Table 6.4. The simulation was terminated after 12 seconds as for true wind angles in the range of 25-35 the vessel no longer maintained foil-borne conditions. Figures 6.6 through 6.9 are the results for a true wind angle of 25°, while Figures 6.10 through 6.24 are the results obtained for wind angles between 25° and 75° with a step of 5.

Figure 6.6a is the total torque in roll, where the motion is steady until the second de-powering is initiated at around 8.5 seconds. Differing from the first de-powering which occurs at around 1 second, the result is a sudden peak in total torque. Subsequently, the roll velocity is decelerated extremely fast seen when plotting the acceleration and velocity in Figure 6.6b and 6.6c respectively. The sudden peak is also noticeable in terms of pitch, heave and surge which can be seen in Figure 6.7a, 6.8a and 6.9a respectively.

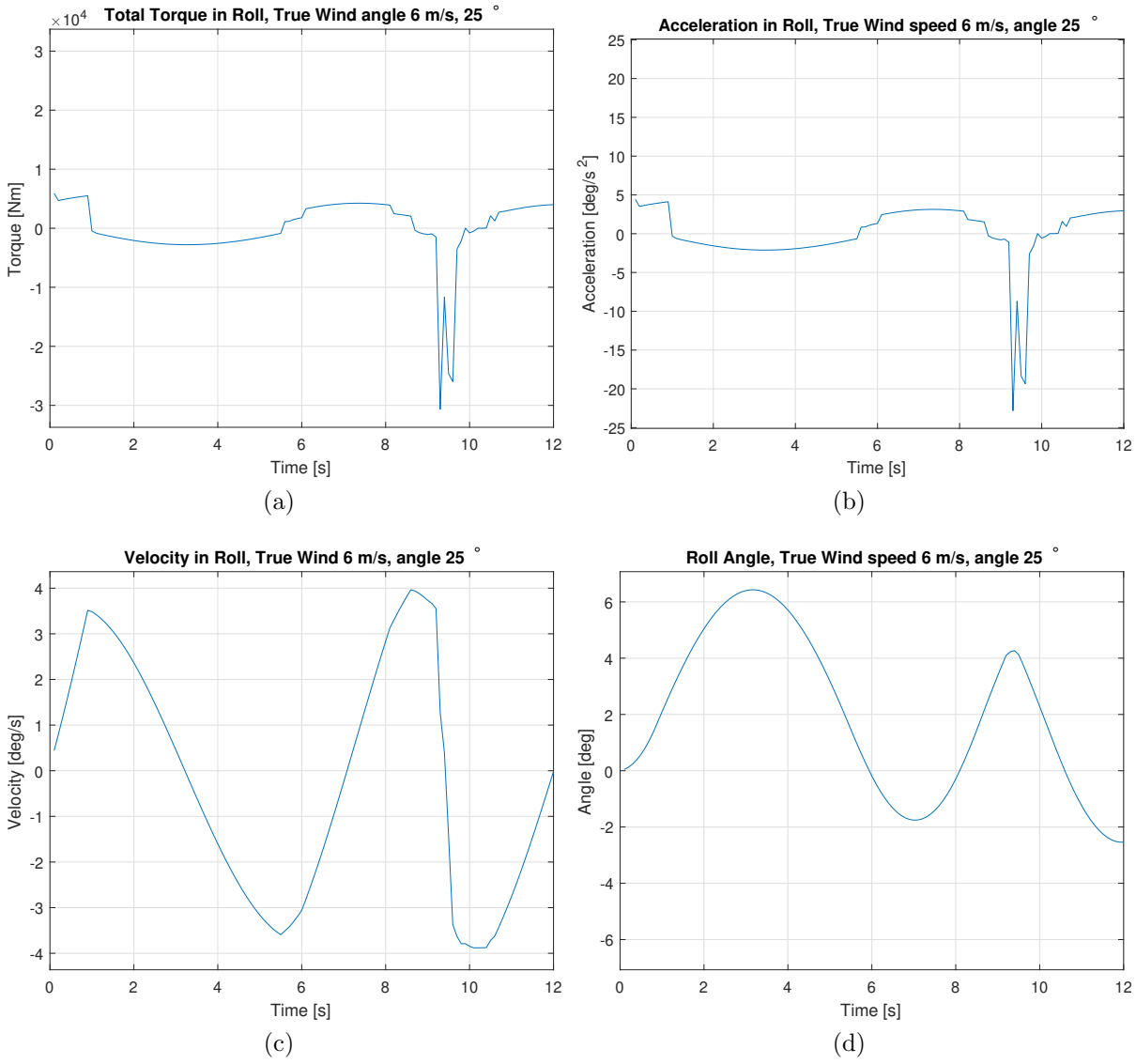


Figure 6.6: 4 DOF Roll motion, a - Total Torque, b - Acceleration, c - Velocity, d- Angle over 12 seconds.

The effect of the peak on pitch is most noticeable in terms of net change in pitch velocity seen in Figure 6.7c resulting in a steady increase of pitch angle seen in Figure 6.7d. This is due to the area under the acceleration in positive direction is larger than the area under the acceleration in negative direction seen in Figure 6.7b. How this effects the rest of the motions can be viewed as an increase in total submerged area seen in Figure 6.9d, although as the angle is very small the majority of the increased submerged area is contributed to the heave velocity.

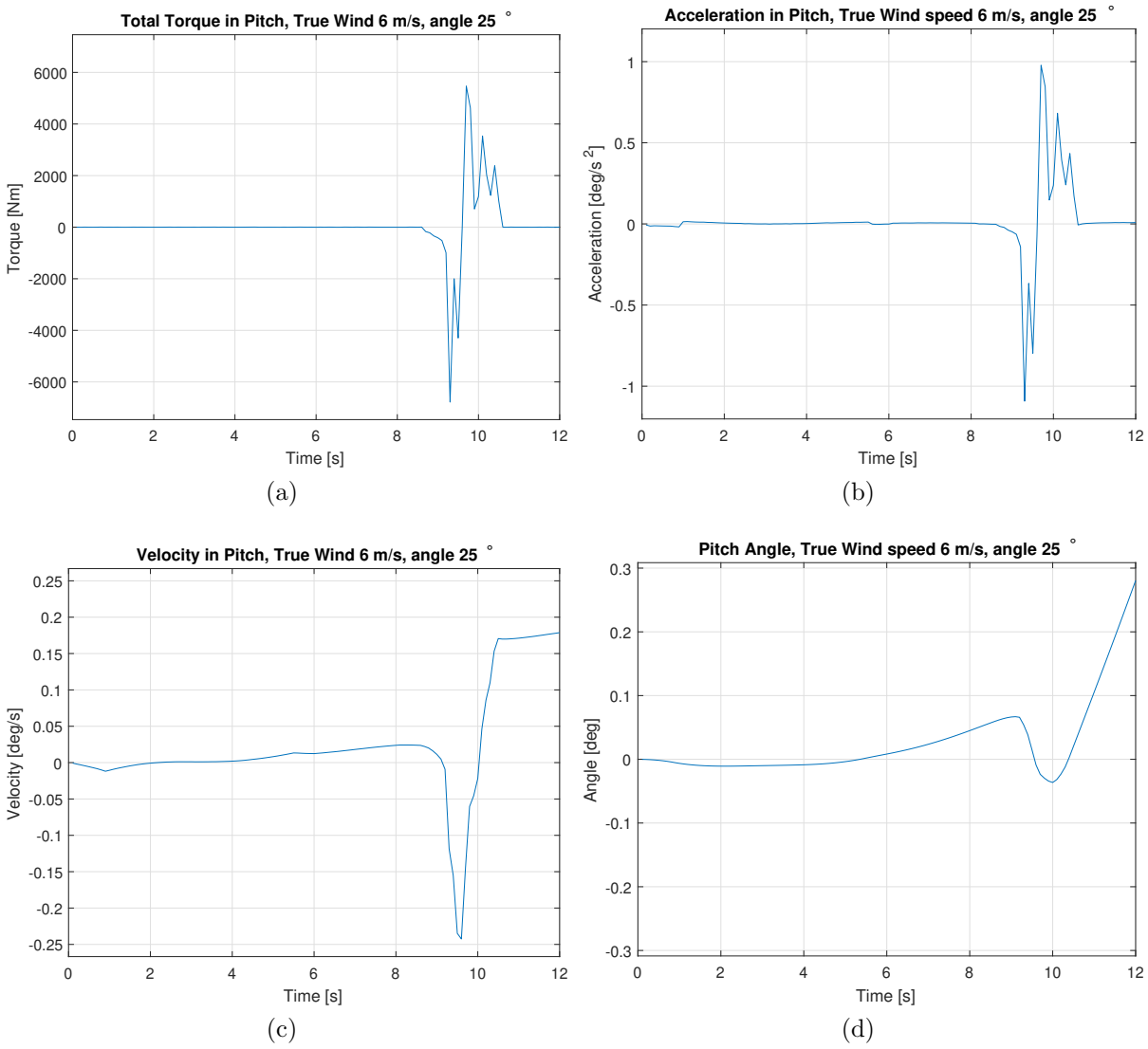


Figure 6.7: 4 DOF Pitch motion, a - Total Torque, b - Acceleration, c - Velocity, d- Angle over 12 seconds.

Due to the initial negative heave acceleration seen in Figure 6.8b the vessel gains a negative heave velocity. This increases the total submerged area seen in Figure 6.9d resulting in more lift and drag, the increased lift results in a heave force balance seen in Figure 6.8a. One would thus expect the vessel to eventually have more positive heave force than negative as the submerged area is steadily increasing. However, as the force in surge is also steadily decreasing seen in Figure 6.9a the net change in heave force is very low. The quick change between positive and negative heave force seen between 9 and 10 seconds is therefore attributed to the heave velocities magnitude and direction. First a force peak occurs which quickly accelerates the heave velocity seen in Figure 6.8c. As the relative size between surge and heave velocity is now diminished the angle of attack is increased, while the heave velocity is positive the angle becomes negative thus reducing the lift. Therefore the lift drops, decelerating the heave motion as seen in Figure 6.8c before quickly finding a new equilibrium.

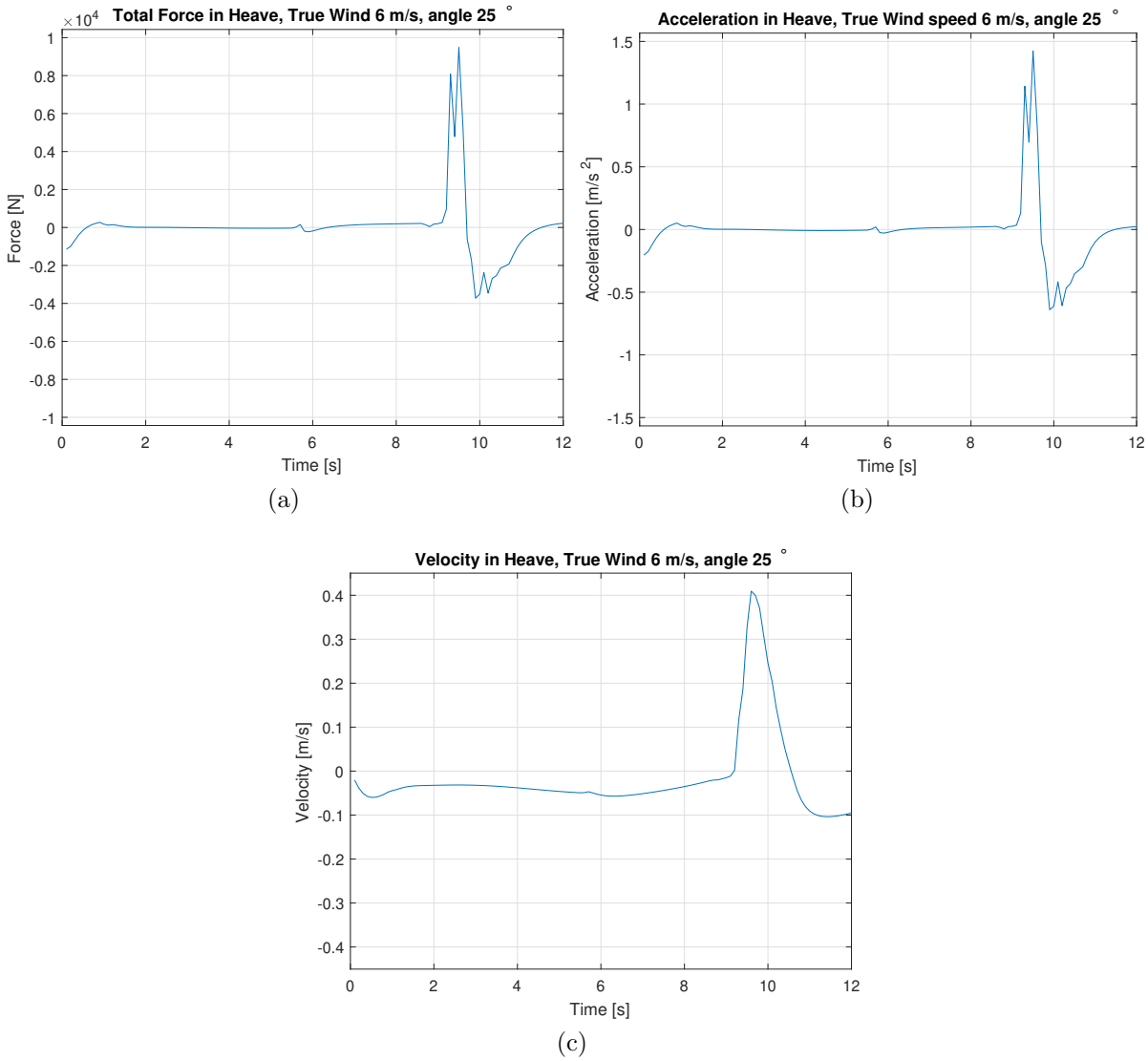


Figure 6.8: 4 DOF Heave motion, a - Total Force, b - Acceleration, c - Velocity, over 12 seconds.

The effect of the de-powering scheme is quite noticeable on the surge motion, as seen in Figure 6.9. Due to the total negative acceleration is higher than total positive acceleration seen in Figure 6.9b the resulting velocity experiences a clear downwards trend in Figure 6.9c. In addition, as the total submerged area increases the total drag increases thus decreasing the total force in surge during times when the sail is not de-powered, where a difference can be seen at 1 second and 6 seconds in Figure 6.9a. This ensures the downwards trend mentioned earlier, however the main reason for the downwards trend is the de-powering scheme is active for more of the simulation as seen when the total force in surge is negative.

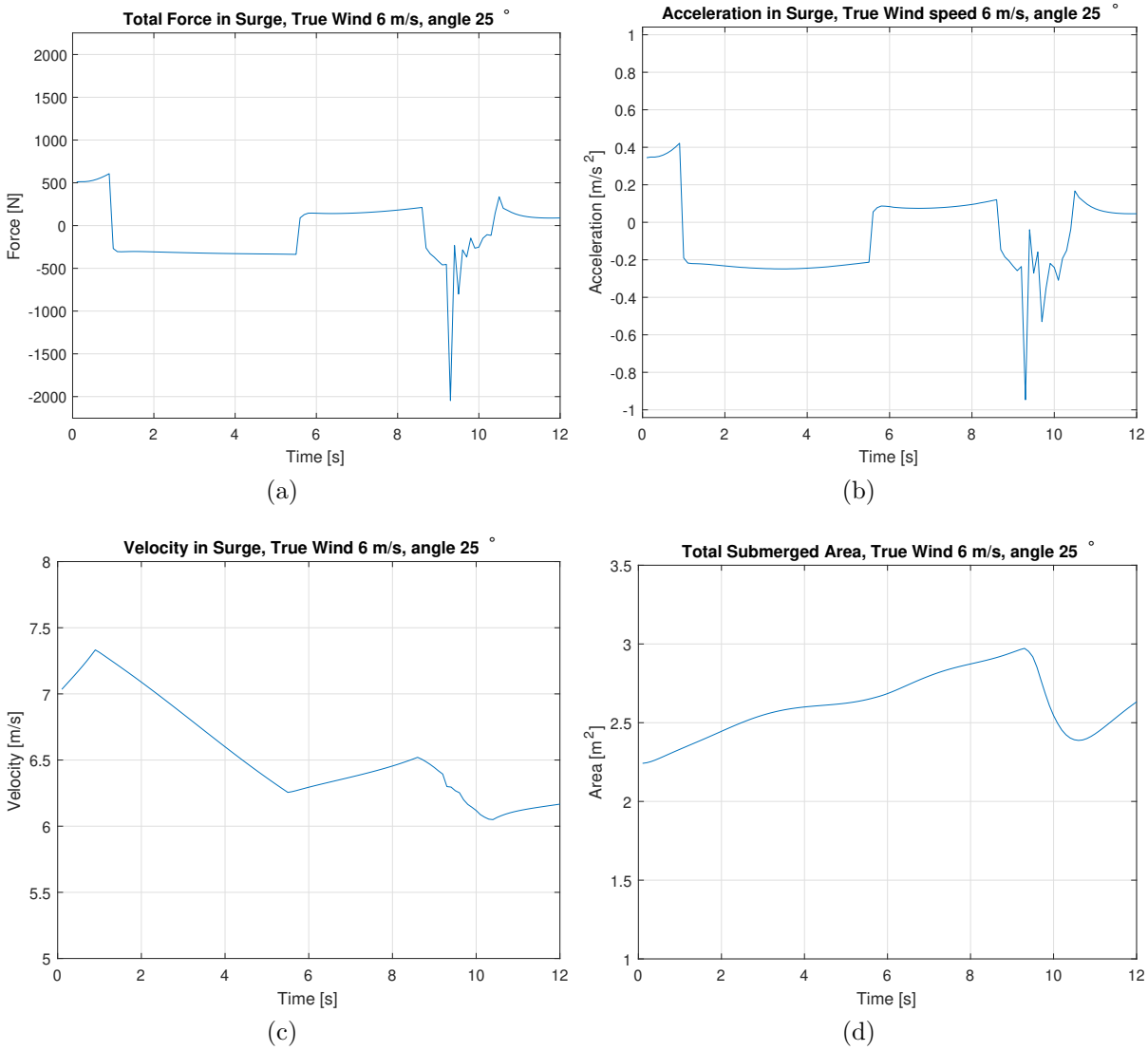


Figure 6.9: 4 DOF Surge motion, a - Total Force, b - Acceleration, c - Velocity, d- Submerged Area over 12 seconds.

It can be safely assumed that the de-powering is the reason for the sudden change, although the reason a peak occurs now and not every time the sail is de-powered, is due to the condition of the vessel just prior to the de-powering. Before the peak occurs de-powering is initiated at approximately 8.5 seconds resulting in a drop in surge force seen in Figure 6.9a this reduces the forward speed of the vessel seen in Figure 6.9c. As the forward speed is reduced the angle of attack increases due to relative size between heave and surge velocity has decreased. In addition, the total submerged area seen in Figure 6.9d has also increased since the last de-powering. Therefore it is believed the peak in total torque in roll occurs due to the peak in heave, de-powering of the sail reducing the heel moment and positive roll angle ensuring the starboard foil is more submerged than its port side counterpart.

All True wind Angles

Predictably the roll angle seen in Figure 6.13 diminishes for increasing wind angle. This is contributed to the force produced by the sail is diminishing for increasing true wind angle, due to a decreasing relative wind velocity. It also noticeable that the acceleration seen in Figure 6.12 decreases with increasing wing angle. It is also noticeable that the de-powering scheme is working as intended as the roll velocity decelerates. Although it is noticeable that the roll velocity increases for the second oscillation seen in Figure 6.11 which may indicate that current de-powering scheme implemented is destabilising the vessel. In addition, the de-powering scheme can also be seen to initiate at a later stage for increasing true wind angle.

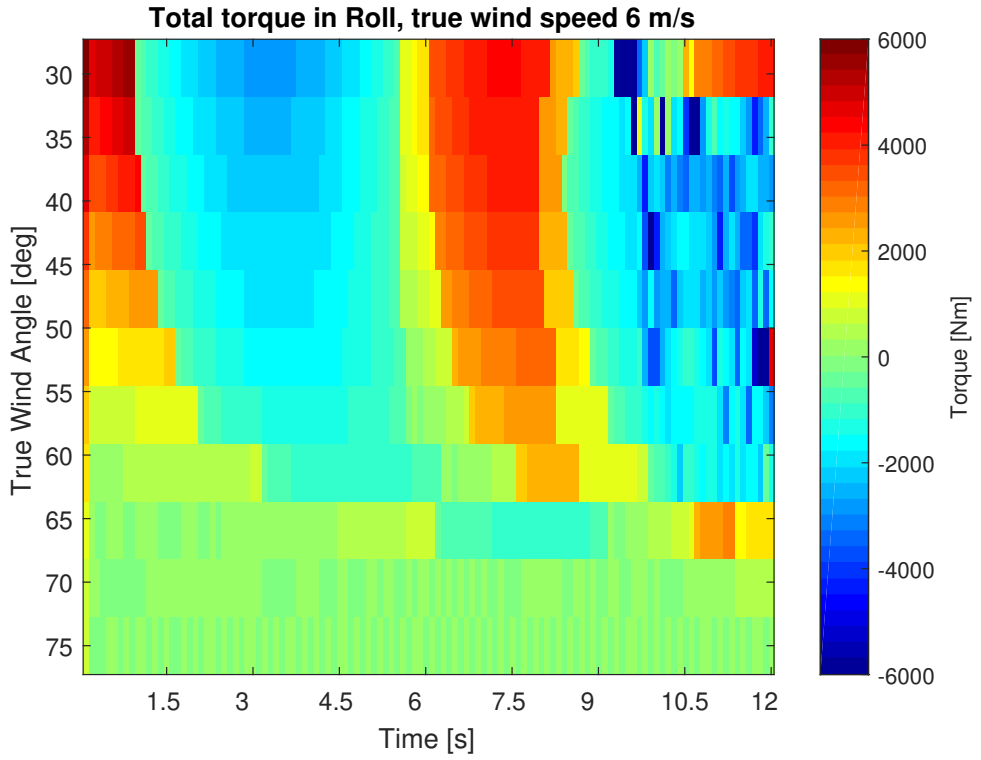


Figure 6.10: Total torque in roll, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

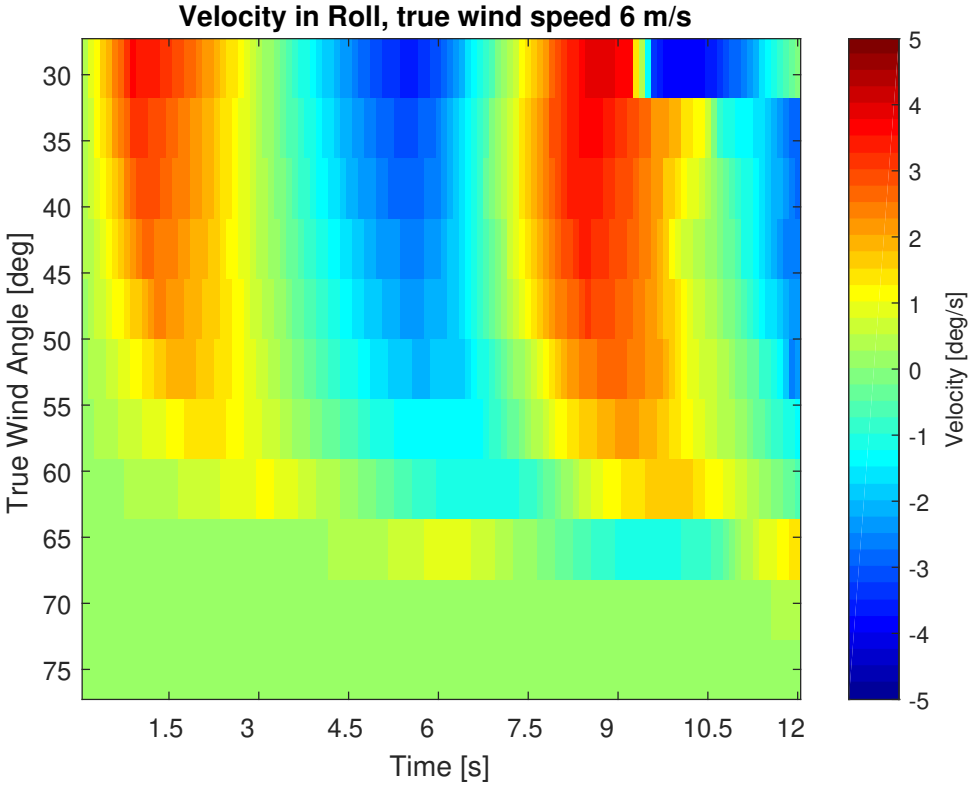


Figure 6.11: Velocity in roll, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

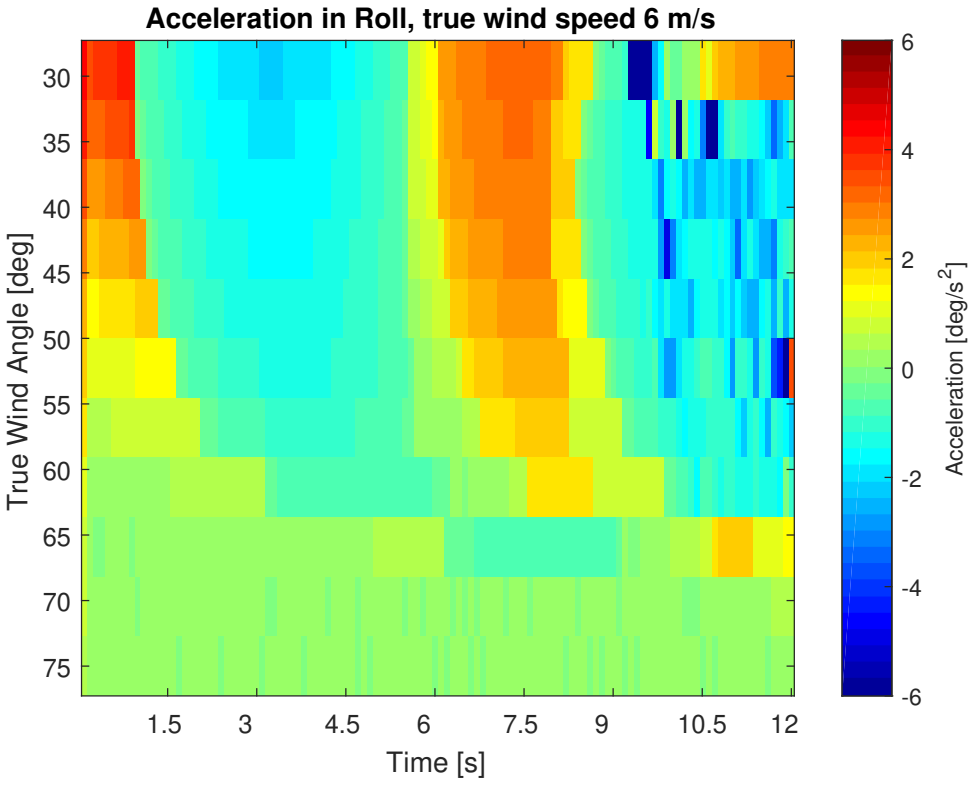


Figure 6.12: Acceleration in roll, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

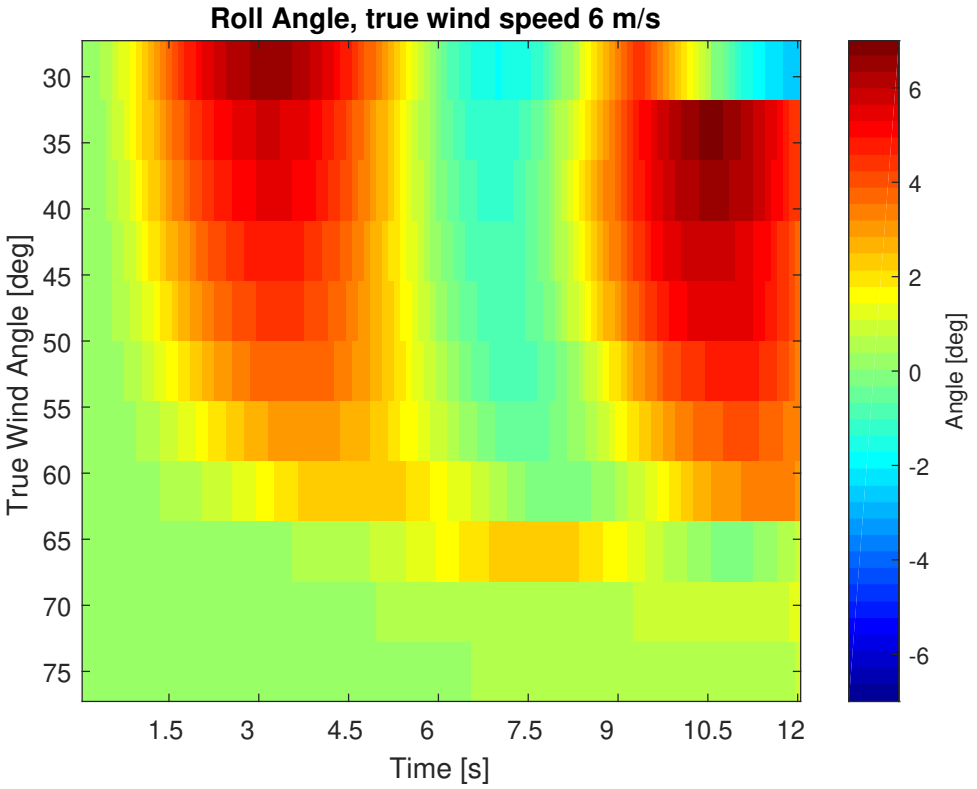


Figure 6.13: Roll angle, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

The severity of the heave peak at 8.5 seconds can be seen to diminish and occur at a later stage with increasing true wind angle seen in Figure 6.18. In addition, the effect of the peak can also be seen to diminish in Figures 6.10 and 6.14 which is the total torque in roll and pitch respectively. The reason is most likely due to a decreased submerged area seen in Figure 6.24 in the same area. In addition, the velocity in surge is greater in the same area seen in Figure 6.23 thus the relative size between surge and heave velocity is lower resulting in a lower angle of attack. One would expect the lower true wind angles to obtain larger surge velocities due to the larger relative wind speed. However, due to the de-powering scheme the vessel uses less of the sail at larger true wind angles as a countermeasure to the roll motion.

It is believed based on Figure 6.17 which visualises the pitch angle for all wind angles, that pitch motion is less of a problem then roll motion. This is also reinforced by the 1 DOF pitch test where the pitch motion is slow. However, it is noticeable that during certain wind angles, the pitch angle is steadily increasing. This is most likely the result of allowing one of the crew members to always find a position which is optimal towards the total torque in roll and pitch. Which explains the overall small jumps between positive and negative torque in Figure 6.14. Although the resulting acceleration viewed in Figure 6.16 is small it is almost in all cases the same sign. A negative consequence of this is seen when visualising the pitch velocity in Figure 6.15. Where the net gain is an increasing pitch velocity thus never stabilising.

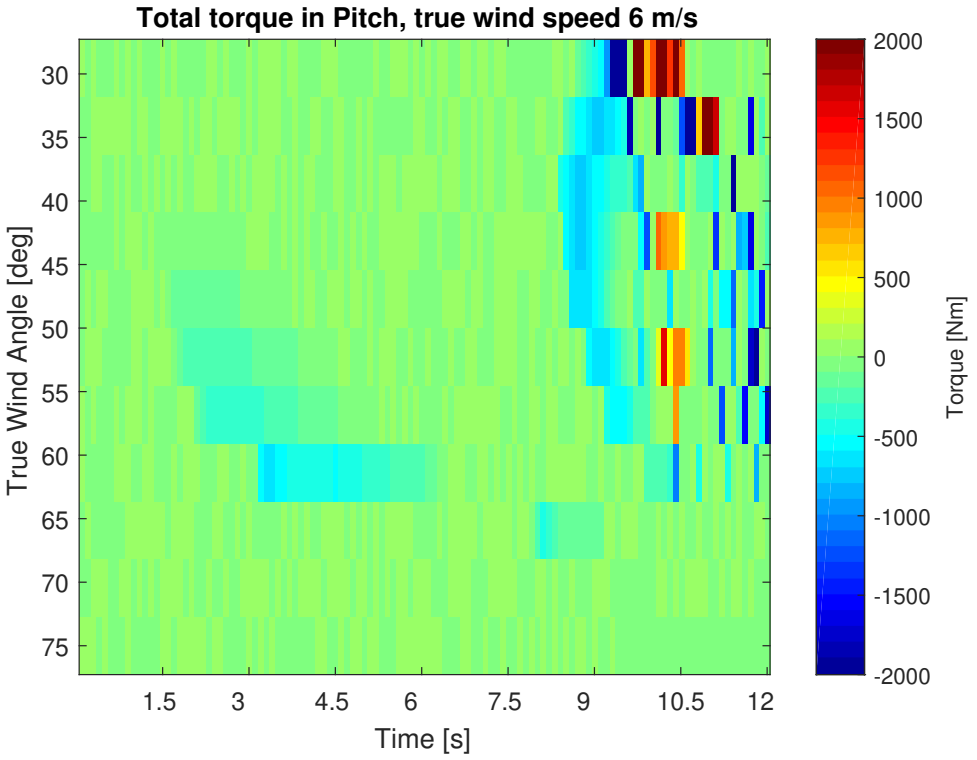


Figure 6.14: Total torque in pitch, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

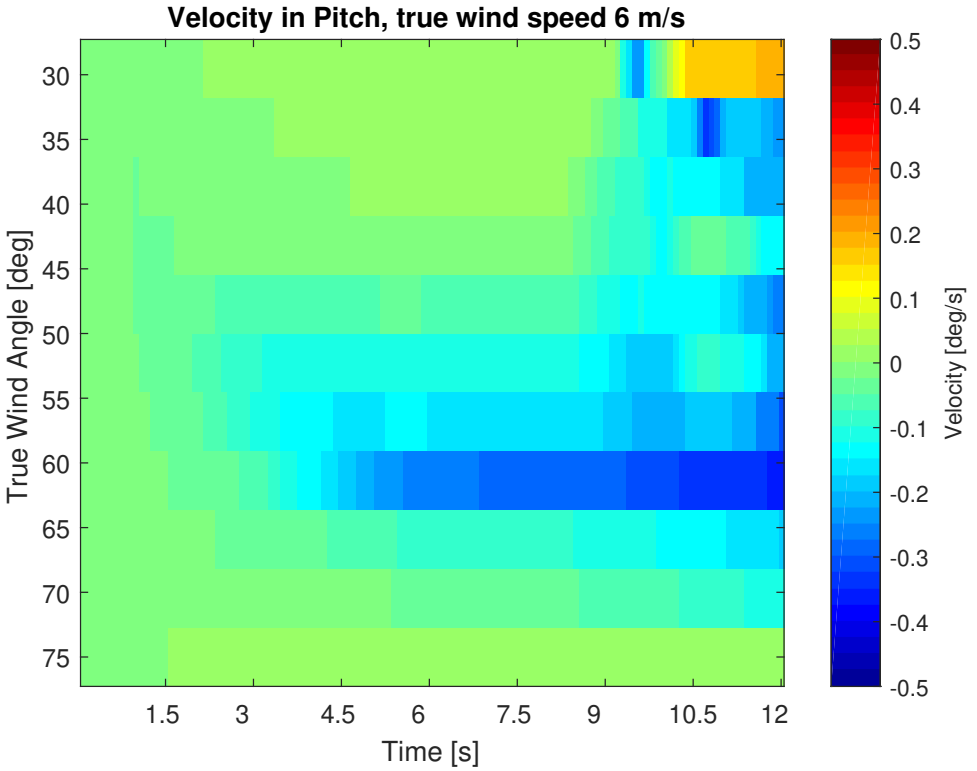


Figure 6.15: Velocity in pitch, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

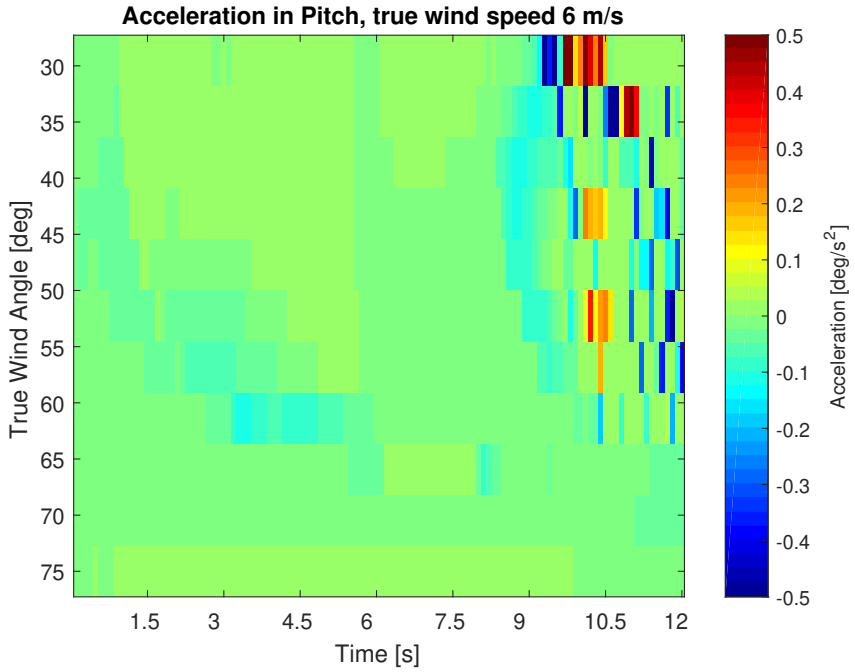


Figure 6.16: Acceleration in pitch, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

The effect of pitch on the other motions can be seen when comparing the pitch angle with the total submerged area in Figures 6.17 and 6.24. Where it seen that for decreasing pitch angle the total submerged area decreases, the reduced submerged area results in less drag and lift. The resulting total force in surge can be seen in Figure 6.21 to be larger in this area, although the difference is small. In terms of heave the effect exists although small due to the higher surge velocity seen in Figure 6.23 in the same area.

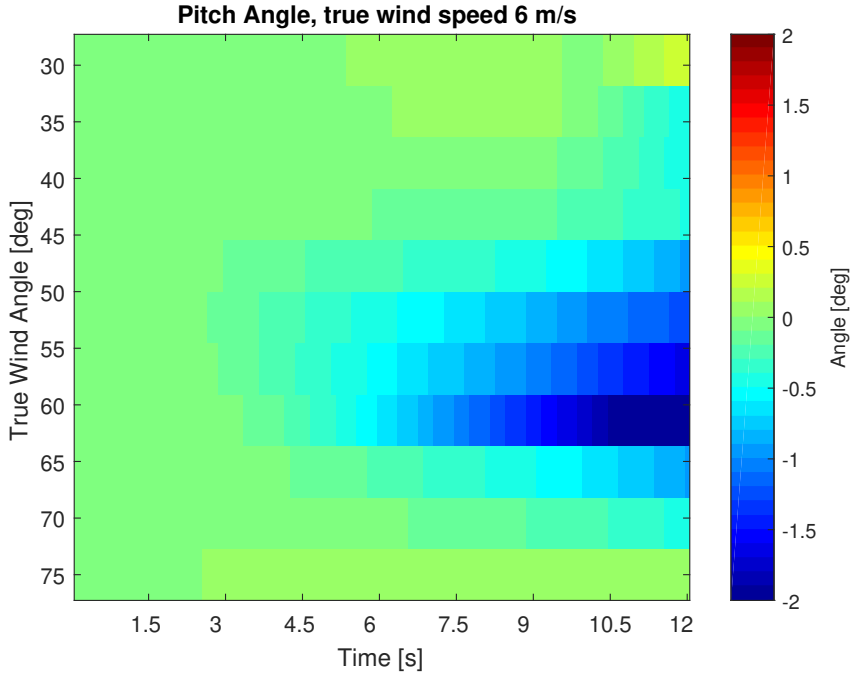


Figure 6.17: Pitch angle, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

The interaction between surge and heave is less obvious based on Figure 6.18 and 6.23 which are respectively the force in heave and velocity in surge. As one would expect the force in heave to show the same trend as the velocity in surge, meaning a reduction in heave when the surge velocity is lower. However, this can be explained by the heave velocity and total submerged area in Figures 6.20 and 6.24 respectively. Based on Figure 6.24 the total submerged area is larger in areas where the surge velocity is lowest, while lower in areas where the surge velocity is largest. In addition, the heave velocity is in general negative in areas where the surge velocity is lowest thus contributing to a larger heave force. This is due to the incoming flow is equal and opposite the direction of the vessel motion. Thus a negative heave velocity produces a positive angle of attack, while a positive heave velocity results in a negative angle of attack. Du. Thus as both are important for the lift produced it would indicate the net sum in heave force remains largely the same.

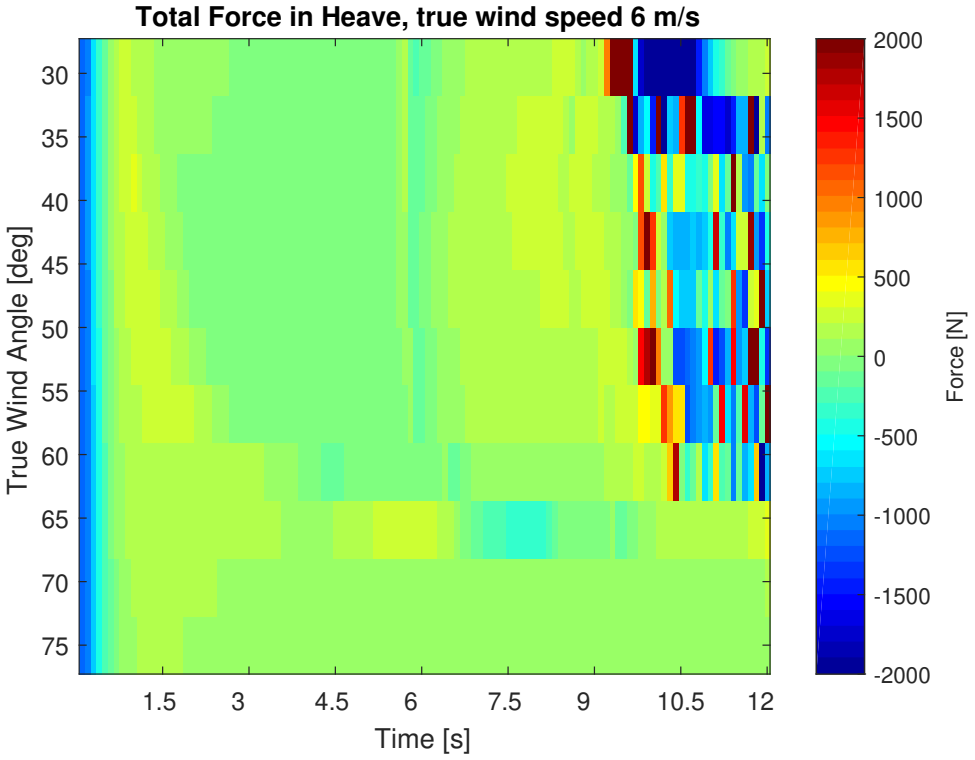


Figure 6.18: Total Force in heave, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

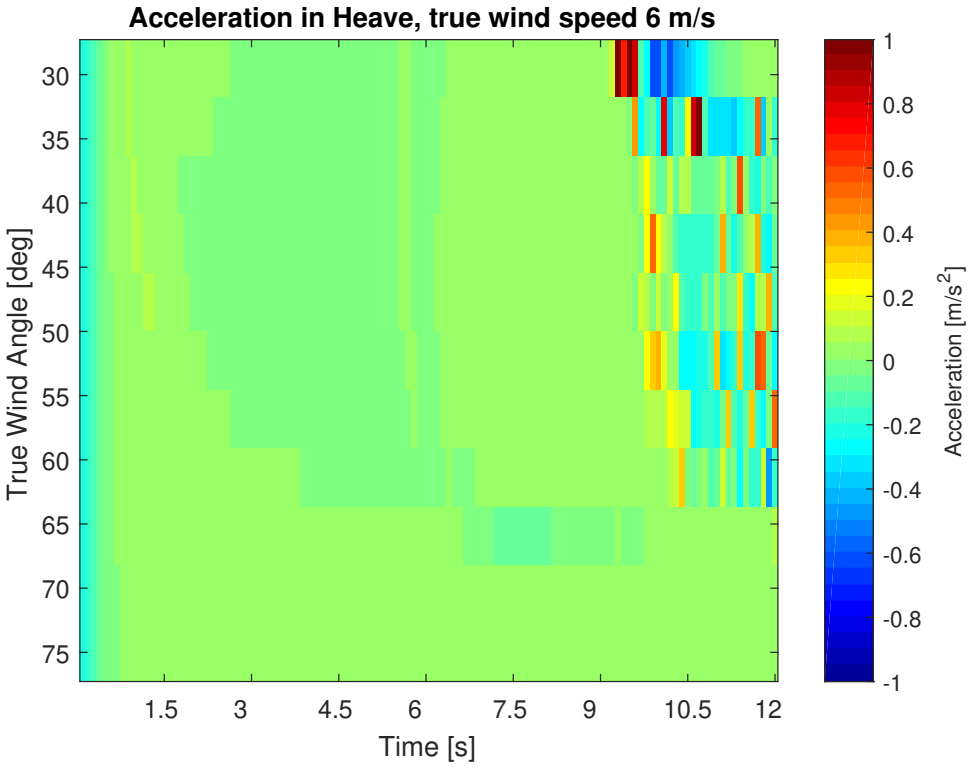


Figure 6.19: Acceleration in heave, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

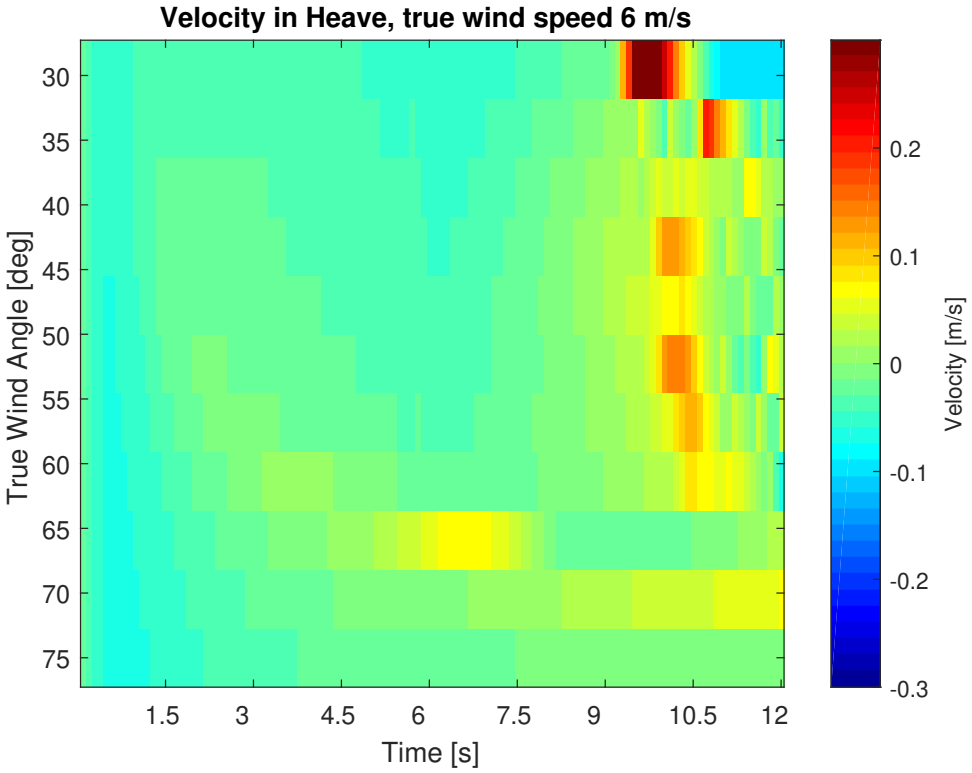


Figure 6.20: Velocity in heave, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

It is clear that the de-powering scheme of the sail has had a clear effect on the surge motion of the vessel. This is based on Figure 6.21 visualising the total surge force, where for most of the simulation is negative. Thus decelerating the forward speed of the vessel seen in Figure 6.23, however it can be seen that net change in velocity diminishes with increasing true wind angle. Although this is only valid for the cases where the de-powering scheme is most prominent. Thus further exemplifying the effect of the de-powering scheme on the performance of the vessel. The reduced forward speed affects two vital factors, the apparent wing velocity and importance of heave velocity. For decreasing apparent wind velocity the forces and torques produced by the sail diminish thus reducing the required restoring forces. However it also means that if the drag of the vessel does not decrease faster than the decreasing sail force the vessel will ultimately obtain a vessel speed where it is no longer able to maintain foil-borne conditions. Although based on Figure 6.22 visualising the acceleration in surge, the vessel is accelerating faster than it decelerates when de-powering. In terms of heave velocity, it will affect the hydrofoils more as the relative size between surge and heave velocity diminishes. This can, if the difference becomes low enough create large angles of attack. In addition the threshold for reaching the zero lift angle diminishes. In addition, the increased angle of attack would also increase the drag coefficient more than an increase in forward speed.

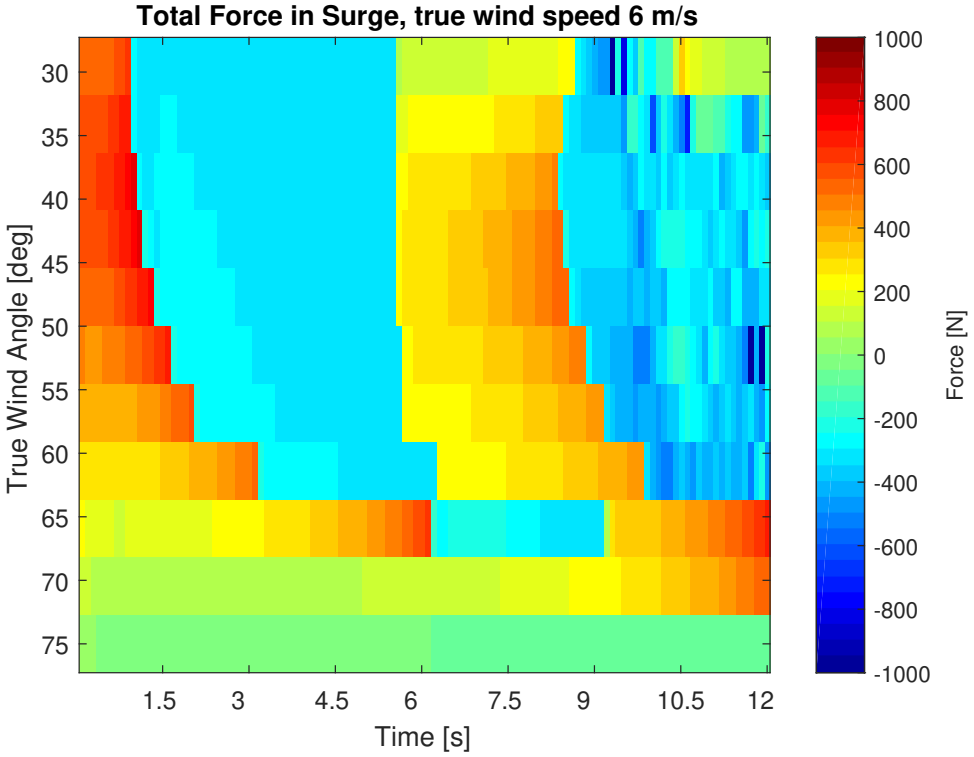


Figure 6.21: Total Force in surge, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

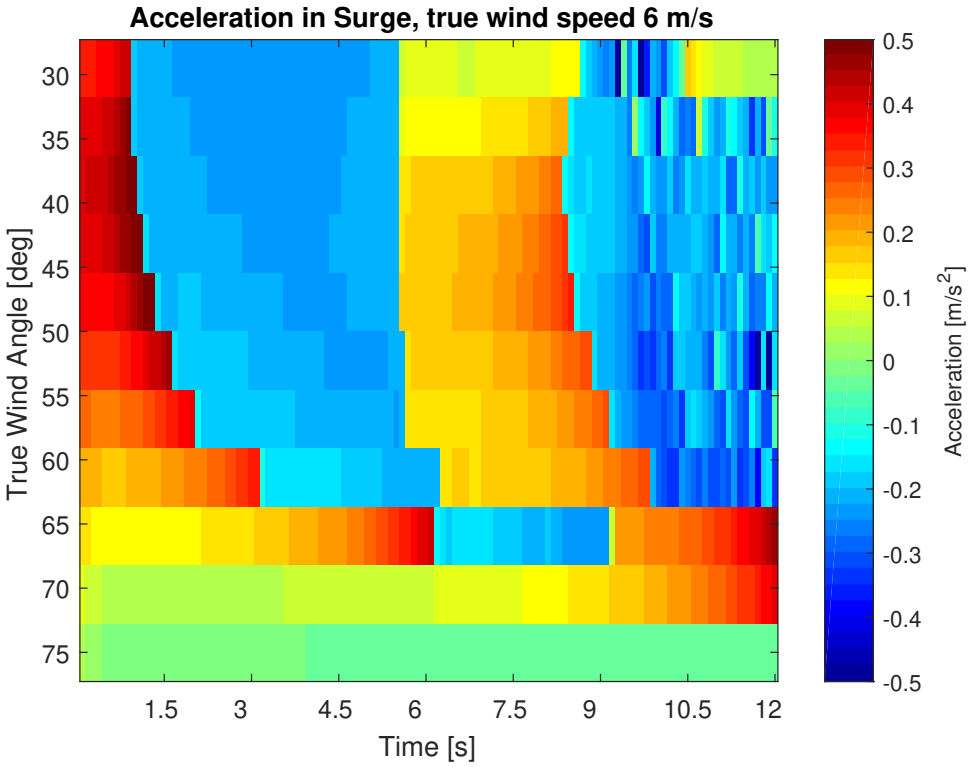


Figure 6.22: Acceleration in surge, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

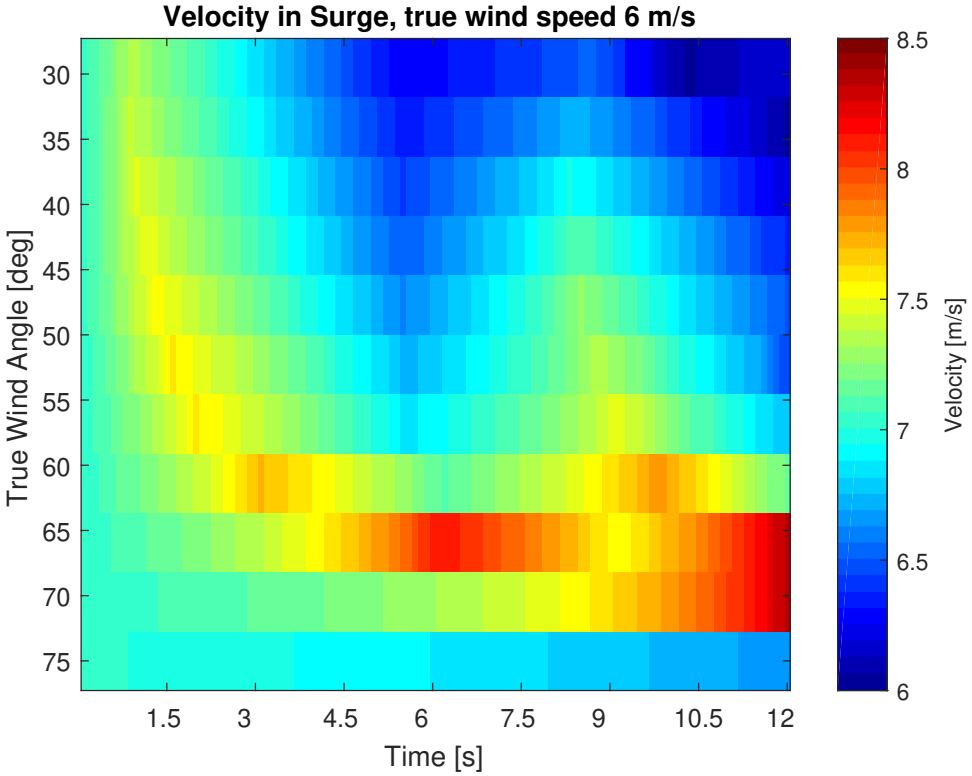


Figure 6.23: Velocity in surge, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

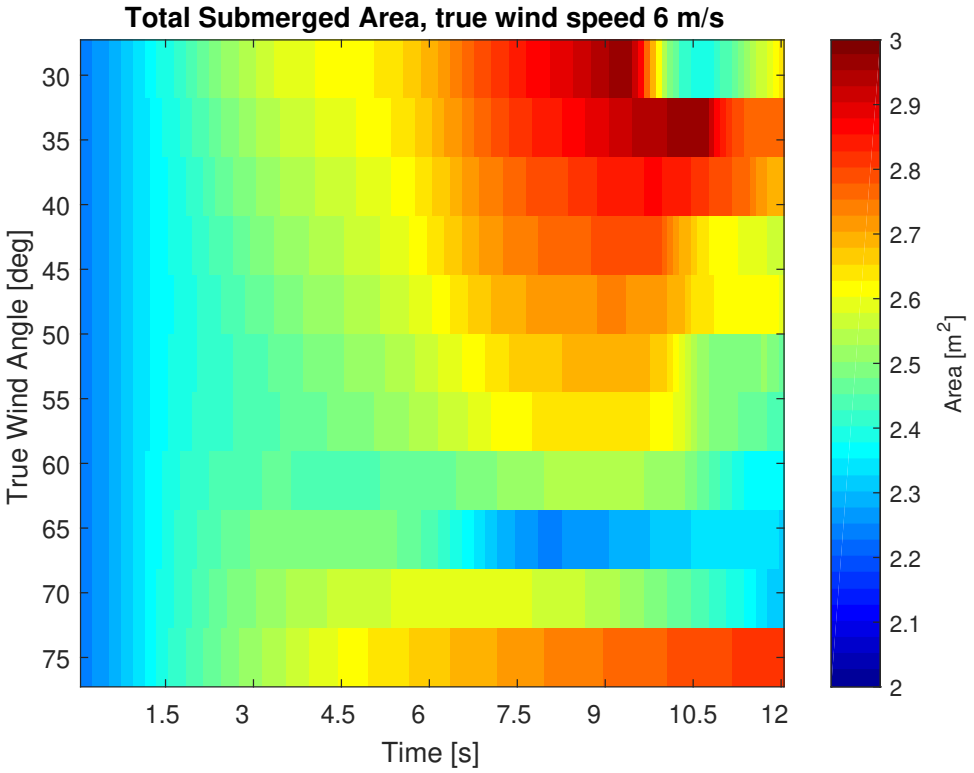


Figure 6.24: Total submerged area, for all wind angles over time, at $6\frac{m}{s}$ true wind speed at 30°.

Chapter 7

Discussion

In this section are thoughts and comments on the results and overall simulation model.

7.1 Results

This section is a further discussion on the results. First the peak occurrence is discussed before a discussion on the short simulation time in 4 Degrees of freedom.

Heave Peak Occuring at 8.5 seconds

The magnitude of the peak which occurs for some true wind angles is concerning, although the peak as formerly discussed, is contributed to an increase of submerged total area, diminished relative size between heave and surge velocities and the roll angle. The magnitude of the peak results in a roll acceleration equivalent to 2.5 G's, which according to NASA is just below the threshold astronauts are subject to when propelling themselves of the planet (NASA, 1990).

Another explanation for the peak is regrettably a fault in the lifting model. This is based on the peak like behaviour seen when investigating whether the wake roll up had any effect in Section 5.3 visualised in Figure 5.10. Where a peak is seen during the development of the wake field, here the lift also experiences a sudden jump doubling its lift coefficient. Although, this occurs only when the wake field has not been fully established, as the peak occurs far into the simulation the wake field is established. Although, as mentioned in Section 4.4 the wake is cut after 3 seconds to reduce the computational time, which was based on the lift being unchanged when removing the wake after 10 chord lengths.

Thus an explanation to the peak could be a ripple effect where the wake is increasingly faulty as the simulation runs its course. Where for each time step the wake has been removed the wake becomes increasingly different from an uncut wake field. However, it is peculiar that it only occurs during the second de-powering, in addition the peak diminishes with increasing true wind angle seen in Figure 6.8. Also, this phenomenon does not appear for 1 DOF simulations which lasted longer.

To conclude it is unknown whether the vessels specific conditions triggered the peak or whether the lifting model has a flaw. This based on both arguments being equally valid, what is known is that the magnitude of the peak is infeasible as the acceleration is far to large.

Termination after 12 seconds

Based on the 1 DOF simulations the vessel is able to restore itself to its upright position. However, 4 DOF simulations failed the criteria of foil-borne conditions which suggests the vessel set-up is not able to maintain foil-borne conditions. Although as discussed in the results, there are indications the de-powering scheme of the sail is creating an unstable environment due to increased velocities between oscillations.

Furthermore, the zero tolerance of hull in water might be to strict for the current model to maintain. As many control devices are primitive and non-existent. These include a more realistic representation of the crew, a ride height control mechanism to maintain a stable altitude between the hull and sea surface and lastly the rudder would create a roll torque on the vessel. As sway and yaw were neglected for the simulations the motions could contribute towards a more stable vessel. This is further discussed in Section 7.2.

In addition, the roll angle and pitch angle of the vessel seen in Figure 6.13 and 6.17 respectively is rather low for the vessel to fail the foil-borne condition. Although it can be seen that the vessel has in general a period of negative heave velocity seen in Figure 6.20. The decreased gap between the hull sea surface means the hull will enter the water a lower roll angle.

Furthermore, the dihedral foils are positioned far out from the hull which is an attempt at creating a longer moment arm to counter act the sail force. This due to a dihedral foil has a force component in both z and y direction, the relative size between these depends on the dihedral angle. As the vessel heels the difference in submerged area changes as one side is lifted out of the water while the other sinks in. This ensures the force difference between the port and starboard side increases allowing one side to counteract the heel moment of the sail. However, as the dihedral has a force component in both z and y direction one of these will work in the direction of the heeling angle which depends on which side the vessel is heeling to. Thus to ensure that the force component creating a heeling torque is kept to a minimum the moment arm either has to be reduced or the moment arm for the restoring force needs to be increased. Which is why the dihedral foils are positioned far out, which in turn is why the simulation terminates as the extension dips below the water long before the hull.

This is a flaw in the simulation and a work around would require an overhaul to how the submerged section of each foil is determined. In hindsight, the method used described in Section 4.3 which is a simple plane line intersection function, works perfectly when it is assumed the vessel will always be foil-borne. In consideration of the results, the method seems to be sub-optimal, as the results indicate the simulation criteria is too strict.

7.2 Simulation Model

This section is a discussion on the neglections, control devices and general weak aspects of the model.

Potential Flow Approach

Based on the investigations carried out in section 5 to determine and verify the validity of the lifting model it is safe to conclude the model is adequate during attached flow conditions. However, the validity of the model suffers greatly when overstepping its boundaries seen in Figure 5.6 where the lift increases linearly where it should have dropped due to viscous effects. This means the model can be correct only under certain conditions, which is a problem with regard to a training simulator. A work around to this, could be a training simulator where the crew is trained for certain scenarios. Upwind sailing and tacking manoeuvres are two such scenarios which can be considered, although there will be a period during the tacking manoeuvre where the model will not be entirely correct. This however is not a major problem as the time frame is small and during a tacking manoeuvre the sail is mostly luffing losing most of its power. Thus there should not be a problem implementing a suitable routine during the short time the sail will experience a large angle of attack.

Downwind sailing conditions are out of the question for a purely potential based method. This is due to the angle of attack being to large, which is a consequence of the assumptions and neglections made in the lifting surface model with regard to boundary conditions. This is due to the model assumes the separation point is always at the trailing edge of the foil whereas for larger angles of attack this point will move away from the trailing edge. Which is due to the model ignores the effects of boundary layer theory.

Neglection of Sway and Yaw

It was decided to neglect the effects of sway and yaw motion based on (Masuyama, 1987) who performed a similar analysis and found that the sideslip angle was small. However, the sideslip angle would mean a different angle of attack on the hydrofoils. The addition of sway and yaw would be most noticeable on the dihedral foils as they are angled thus based on the direction of the sideslip the forces would increase on one side while decreasing on the other. The effect of a sideslip angle would only be noticeable for the rudder T-foil when the vessel has a roll angle. The rudder would also require either a larger angling to minimize the yaw motion or lower depending on the size and direction of the sideslip. In addition, it would require a control mechanism for the rudder which if based on the current implementation of the de-powering scheme of the sail, will most likely end in an even more unstable vessel

In addition, the sideslip would change the apparent wind speed as the true wind angle would no longer be constant. Thus the true wind speed could increase or decrease pending on the direction of the sideslip. The total effect of sideslip would thus be quite noticeable as both the sail and hydrofoils would perform differently.

Neglection of Resistance Components

In hindsight, a major flaw in the model is that the only drag component the lifting model considers is the induced drag. Thus the total force in surge should be lower as resistance due to friction, spray and intersection resistance is not considered. Thus the surge accelerations in Figure 6.22 are tenuous at best. The vessel should have been decelerating faster and acceleration slower as the total drag should be higher, this also means the heave velocity would have a stronger effect. The effect on total force in heave however, would largely remain the same as it self stabilises due to changes in submerged area.

Control Devices

The control device for the sail is believed to have worked as intended based on Figure 6.6d, 6.6b and 6.13. Expectedly, the reduced angle of attack on the sail which reduced the sails force in sway also meant a reduction in the sails surge force. Thus decelerating the vessel leading to a reduced relative velocity on the hydrofoils. However, as discussed it is believed it contributed to an unstable solution. Thus the current implementation should be remodelled when going forward.

Other control devices which were not implemented as their algorithms and search parameters were not optimal include: A rudder algorithm to minimize yaw motion and an adjustable foil to minimize heave motion. The search parameters used were the total torque and force in yaw and heave respectively. This resulted in an unstable solution where the vessel would gain a constant velocity due to the net gain in acceleration was either positive or negative. Ideally the algorithm would initiate only when the vessel was exceeding a certain angle for yaw control and when a certain altitude was exceeded. The implementation would be similar to the de-powering scheme of the sail, although based on the results it indicate that the current implementation is unstable.

Real Time Simulation

Current work was able to severely diminish the required computational time for the lifting model. Where for a single foil the computational time reached as low as 0.06 seconds. However, as there are multiple foils to simulate and a sail, the simulation failed to accomplish real time speeds. This is also in addition to the time required to run several other routines handling geometry and dynamics of the vessel. Although, as the foils and sail are modelled independently from one another and the computation was not done in parallel, it is believed close to real-time speeds would have been possible if an emphasis on parallel computing was focused on at an earlier date. However, this would have drawn time from focusing on modelling the dynamics of a sailing hydrofoil which was the main task.

Geometry Parametrisation

An attempt was made to parametrises the geometry of the vessel, but not implemented correctly. As such It is not able to adjust how many foils there are, nor can it differentiate between a mono-hull and a catamaran. The fault is the author did not divest enough time into this section, and the task was deemed to difficult based on the authors prior experience with programming and geometry parametrisation. In addition, the shape of the hydrofoils are limited to rectangular foils or foils with a sweep, this is not a problem for the current lifting model implemented as complex shapes would require a different model.

Chapter 8

Further Work

Further work can be categorised into three sections; improving the physical considerations of the model, improving kinematics and control mechanisms and lastly improving computational efficiency and speed.

Physical Considerations

- Hydrodynamics

If there is a desire to use more complex shapes than rectangular or swept wings, the alternative would be a panel method. This would also mean form drag due to the thickness of the foil would be considered. The implementation is very similar to the current implementation, where the main difficulty would be defining the geometry accurately. However the number of elements will increase drastically which is not ideal for real-time simulations.

Interaction effects due to the created wake is also important as the aft foil of a tandem hydrofoil system has decreased lift. If implemented, several more design choices can then be investigated. Such as optimum distance between hydrofoils and optimum geometry for each foil (Kemal, 2015).

Improving upon drag estimations is an important aspect when improving upon the model. Drag components to be considered are then; wave resistance due to free surface, interference resistance, spray resistance and frictional resistance.

Non-foilborne conditions should also be considered. This would entail implementing a suitable model for the hull resistance. A possible candidate is the strip-theory potential model, the real challenge would be how to define the wetted surface.

- Aerodynamics

Downwind sailing is an aspect which should be looked at when improving the model. A quasi-empirical approach could be viable as potential based methods are not up to the task due to large angles of attack. If a wing sail is not desired, then a method to handle aeroelasticity should be considered as fluttering will affect the effectiveness. In addition, as previously discussed interaction effects will make it possible to more accurately model

the jib sail, the sail ahead of the mainsail. Although it is possible to model it without interaction effects, however the effect of the mainsail without including interaction effects would be wrong, which is due to their close proximity to one another.

Kinematics and Control mechanisms

Implementing a model such as presented by (Masuyama, 1987) can be useful when considering manoeuvres. However, it would also mean finding suitable approximations of the stability derivatives. The control mechanisms can be improved by adding three algorithms; a path generator, a tracking function and path following. The path generators function is to define an area the vessel should move through, while the tracking function determines where the vessel has been in time, lastly the path following function is to adjust controllable mechanisms such as sails, crew, rudders and foil flaps to maintain course.

Model Architecture

High computational power is a given when attempting to simulate in real time, however increased power means nothing if the model is not able to utilise said power. As such parallelisation should be be kept in mind when selecting hardware and programming. Further playing to the program languages strong side should always be considered. The author is not well versed, but an example is utilising Matlab's matrix manipulation and vectorisation.

Conclusion

Based on the investigations carried out it is believed the created lifting surface model sufficiently models the effects of camber and aspect ratio during attached conditions. Furthermore, the modelling of dynamic lift is also believed to be sufficiently accurate during attached conditions, which was investigated against Theodorsen's function.

Current work was able to severely diminish the required computational time for the lifting model, however as the computations are not done in parallel real-time speeds were not accomplished.

It is believed the current model is able to simulate the dynamic behaviour of a sailing hydrofoil. Although, the model is far from perfect and experienced an abnormal behaviour during specific conditions. In addition, the current implementation of crew behaviour is believed to have made the vessel unstable.

With regards to a potential based simulator, the method is adequate but suffers predictably during conditions where viscous effects are dominant. This means, the simulator will either have to be made for certain scenarios or switch to adequate models when potential flow suffers.

Bibliography

Abbot, I., von Doenhoff, A., and Albert, E. (1945). Summary of airfoil data. 1945. *NACA Report*, 824.

Anderson Jr, J. D. (2010). *Fundamentals of aerodynamics*. Tata McGraw-Hill Education.

Blazek, J. (2015). *Computational fluid dynamics: principles and applications*. Butterworth-Heinemann.

Caponneto-Hueber (2019). Caponneto-Hueber. http://www.caponnetto-hueber.com. Accessed: 2019-02-17.

Chattot, J. and Hafez, M. (2015). *Theoretical and applied aerodynamics*. Springer.

Delbosc, N. (2015). *Real-Time Simulation of Indoor Air Flow using the Lattice Boltzmann Method on Graphics Processing Unit*. PhD thesis, University of Leeds.

Faltinsen, O. M. (2005). *Hydrodynamics of high-speed marine vehicles*. Cambridge university press.

Fiddes, S. and Gaydon, J. (1996). A new vortex lattice method for calculating the flow past yacht sails. *Journal of Wind Engineering and Industrial Aerodynamics*, 63(1-3):35–59.

Fossen, T. I. (2011). *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons.

HochBaum, A. C. (2019). Ship manoeuvrability. *Lecture notes*.

Katz, J. and Plotkin, A. (2001). *Low-speed aerodynamics*, volume 13. Cambridge university press.

Kemal, O. (2015). A numerical parametric study on hydrofoil interaction in tandem. *International Journal of Naval Architecture and Ocean Engineering*, 7(1):25–40.

Kenny, J., Takeda, K., and Thomas, G. (2008). Real-time computational fluid dynamics for flight simulation. In *The Interservice/Industry Training, Simulation&Education Conference (I/ITSEC)*.

Lewis, E. V. (1989). Principles of naval architecture vol iii: Motions in waves and controllability. *The Society of Naval Architects and Marine Engineers, Jersey City, New Jersey*.

Lidtke, A. K., Marimon Giovannetti, L., Breschan, L. M., Sampson, A., Vitti, M., and Taunton, D. (2013). Development of an america's cup 45 tacking simulator.

Linxweiler, J., Krafczyk, M., and Tölke, J. (2010). Highly interactive computational steering for coupled 3d flow problems utilizing multiple gpus. *Computing and visualization in science*, 13(7):299–314.

Marchaj, C. A. (1979). Aero-hydrodynamics of sailing.

Marchaj, C. A. (2003). *Sail performance: techniques to maximize sail power*. International Marine/Ragged Mountain Press.

Masuyama, Y. (1987). Stability analysis and prediction of performance for a hydrofoil sailing boat. *International shipbuilding progress*, 34(398):178–188.

Melnik, R., Chow, R., and Mead, H. (1977). Theory of viscous transonic flow over airfoils at high reynolds number. In *10th Fluid and Plasmadynamics Conference*, page 680.

NASA (1990). NASA. https://www.nasa.gov/. Accessed: 2019-04-25.

NASA (2017). Lattice Boltzmann and Navier-Stokes Cartesian CFD Approaches for Airframe Noice Predicitions. https://www.nas.nasa.gov/assets/pdf/ams/2017/$AMS_20171012_Barad.pdf.Accessed$: $2019 - 02 - 20$.

NVIDIA (2019). NVIDIA - Computer Game Company. https://www.nvidia.com/. Accessed: 2019-03-14.

Ou, Y.-F., Liu, T., Zhao, Z., Ma, Z., and Wang, Y. (2008). Modeling the impact of frame rate on perceptual quality of video. In *2008 15th IEEE International Conference on Image Processing*, pages 689–692. IEEE.

Roper, D., Owen, I., Padfield, G., and Hodge, S. (2006). Integrating cfd and piloted simulation to quantify ship-helicopter operating limits. *The Aeronautical Journal*, 110(1109):419–428.

Sheahan, M. (2009). World Sailing Speed Record Smashed. https://www.yachtingworld.com/news/world-sailing-speed-record-smashed-11654. Accessed: 2019-05-2.

Spalart, P. and Venkatakrishnan, V. (2016). On the role and challenges of cfd in the aerospace industry. *The Aeronautical Journal*, 120(1223):209–232.

Succi, S. (2001). *The lattice Boltzmann equation: for fluid dynamics and beyond*. Oxford university press.

WASZP (2016). WASZP - the one design foiler. http://www.waszp.com/. Accessed: 2019-01-29.

Wenisch, P., Borrmann, A., Rank, E., Van Treeck, C., and Wenisch, O. (2005). Collaborative and interactive cfd simulation using high performance computers. In *18th Symposium AG Simulation (ASIM) and EuroSim. Erlangen, Germany*.

Wolfson Unit (2019). Wolfson Unit. http://www.wumtia.soton.ac.uk/. Accessed: 2019-02-2.

Appendices

First code is the overall simulation model, thereafter are the codes which are used in succession, however multiple functions are used in different functions.

A Code: Simulation Model

```
clear all
close all
clc
%-------------------------------------------------%
% This a simulation model for a sailing hydrofoil vessel. There is a zero
% tolerance allowance for any foil to be entirely submerged, which stops
% terminates the simulation. For a further explenation see the
% Master Thesis
% "An Unsteady Vortex Lattice Approach To Sailing Hydrofoil Dynamics."

% Specificed number of elements in chord and span directions respectively.
n = 10;
m = 10;

dt = 0.1;        % Simulation time step
time = 20;       % Simulation time
t = dt:dt:time; % Simulation time vector
% Simulation requires a few time steps to develope the wake field for the
% foils and sail.
simstart = 30;

% True wind angle, can either be a vector or single unit. Positive value
% indicates the wind is on the port side of the vessel.
wa = 25:5:75;

% Truw Wind Speed, can either be a vector or single unit, negative value
% indicates upwind conditions, negative downwind. N.b the model is not
% accurate for downwind conditions.
Vt = -6;

% Initial value for saving results
pp = 0;

for ll = 1:length(Vt)
```

```
35
36 for jj = 1:length(wa)
37
38     pp = pp+1;
39
40     % Vessel Parameters are specificed in a different file
41 [wind,sail,hull,pos,foil] = simparam(n,m);
42
43 wind.angle = wa(jj);
44 wind.Vt = Vt(ll);
45 % Placeholder due to programming.
46 hydro.placeholder = 1;
47 aero.placeholder = 1;
48
49 %----Initial Conditions------%
50
51 % Creating initial conditions to be evaluated against the
       criteria
52 x = pos.crit(1,:); y = pos.crit(2,:); z = pos.crit(3,:);
53 cr = [x' y' z']; % Creating Initial Criteria
54 % Initial conditions in terms of angle and forward speed
55 Para.Vs = 7; Para.phi = 0; Para.theta = 0; Para.xi = 0; Para.
       h = 0;
56 Para.Vsy = 2;
57
58
59 % Initial conditions for vessel velocities
60 % p,q,r are the rotational velocities in roll, pitch and yaw
       respectively.
61 p = 0; q = 0; r = 0;
62 % u,v,w are the translational velocities in heave, surge and
       sway
63 % respectively.
64 u = Para.Vs; v = 0; w = 0;
65 % Initial values for time step and saving.
66 k = 0;
67 kk = 0;
68 % Critiera of the simulation
69 while (k < time/dt && all([cr(2,3) cr(4,3) cr(8,3)] < 0) &&
       all([cr(1,3) cr(3,3) cr(7,3)] > 0))
70 % time step counter
71 k = k+1;
72 % Determining the rotational matrix
73 [B] = transmatrix(Para.phi,Para.theta,Para.xi);
74
75 Vel = [u; v; w];
76 % Translating the velocities from space to body fixed system.
77 Velw = B*Vel;
```

```
% Determining true wind angle and speed
[wind] = VelocityTriangle(Vel,wind,B);

sail.vel = -wind.Va; sail.dt = dt; sail.k = k;
% Saving the vessel orientation
sail.rot = B;
hydro.rot = B;
aero.rot = B;

[foil,amr,pos,hull] = geomdef(sail,wind,B,hull,pos,foil,Velw)
   ;

sail.pos = [0 0 1];
sail.p = Para.phi;

% Determining the lift and drag from the sail.
aero = liftsail(sail,wind,dt,k,aero,pos,simstart);

% Determining the lift and induced drag from each hydrofoil.
hydro = liftmodel(foil,Vel,dt,k,hydro,amr,pos,simstart);

% Translating the forces into the body fixed system
[Force] = transforce(B,hydro,aero,wind,hull);

if k == 1
[Ig,CG] = MMI(pos);
end
% Determining the added mass of the hydrofoil system.
hull.Ma = addedmass(hydro);

CG = [0 0 hull.char(5)];
% velocity vector for the equation of motion.
etadot = [u; v; w; p; q; r];

if k == 1

   momentarm = pos.momentarm;
end

% Determining the total torque and force in each direciton.
[equil,hull] = equi(Force,momentarm,hull,Para);
% Solving Equation of Motion with respect to accelerations
dydt = solveEOM(equil,Ig,CG,hull,etadot);

% Euler time integration for translational velocities
   uvw = Vel + dydt(1:3)*dt*0.5;
    % Euler time integration for rotational velocities
    pqr = [p; q; r] + dydt(4:6)*dt;
```

```
125
126 if k > simstart
127
128     p = pqr(1); q = pqr(2); r = pqr(3);
129     u = uvw(1); v = uvw(2); w = uvw(3);
130     % if applicable, remove comments from below to simulate
          less degreese
131     % of freedom.
132     u = Para.Vs;
133     v = 0;
134     w = 0;
135     q = 0;
136     r = 0;
137 %     p = 0;
138 end
139 Para.phi = Para.phi+p*dt;
140 Para.theta = Para.theta+q*dt;
141 Para.xi = Para.xi+r*dt;
142
143 % Saving current parameters
144 cr = pos.crit';
145 results.equil = equil;
146 results.uvw = Velw;
147 results.pqr = pqr;
148 results.hydro = hydro;
149 results.aero = aero;
150 results.ang = [Para.phi; Para.theta; Para.xi];
151 results.force = [Force.Ff Force.Fs Force.Fv];
152
153 subArea = 0;
154 for j = 1:4
155     domain = hydro.domain(j);
156
157     subArea = subArea+sum(sum(domain.S));
158     clear domain
159 end
160
161 % Saving values for post-prcessing once the wake field is
      established.
162 if k > simstart
163 kk = kk+1;
164 ee.Sb(kk) = subArea;
165 ee.mx(kk) = equil.Mx;
166 ee.my(kk) = equil.My;
167 ee.mz(kk) = equil.Mz;
168 ee.fx(kk) = equil.Fx;
169 ee.fz(kk) = equil.Fz;
170 ee.v.x(kk) = u;
```

```
ee.v.y(kk) = v;
ee.v.z(kk) = w;
ee.v.p(kk) = p;
ee.v.q(kk) = q;
ee.v.r(kk) = r;
ee.ang.p(kk) = Para.phi;
ee.ang.q(kk) = Para.theta;
ee.ang.r(kk) = Para.xi;
ee.sp(kk) = sqrt(Velw(1)^2+Vel(2)^2+Vel(3)^2);
ee.lam(kk) = atand(Velw(2)/Vel(1));

ee.acc.x(kk) = dydt(1);
ee.acc.y(kk) = dydt(2);
ee.acc.z(kk) = dydt(3);
ee.acc.p(kk) = dydt(4);
ee.acc.q(kk) = dydt(5);
ee.acc.r(kk) = dydt(6);
end


end
% Displays time in simulation
display(['Finnished at ', num2str(k*dt)])
% Displays vessel orientaton
figure(124)
plot3(pos.foil.cord(:,1),pos.foil.cord(:,2),pos.foil.cord
   (:,3),'x')
hold on
plot3(pos.hull.cord(:,1),pos.hull.cord(:,2),pos.hull.cord
   (:,3),'^')
hold on
plot3(pos.sail.cord(:,1),pos.sail.cord(:,2),pos.sail.cord
   (:,3),'v')
hold off
axis equal
grid on
drawnow

% Saving results for post processing
saveresults(pp) = results;
saveresults2(pp) = ee;
% Resetting values for the next true wind angle/speed
clear hydro equil aero pos results B wind sail hull foil crit
clear aa Velw Vela u v w p q r Para AreaA pqr uvw Vel Force
   amr angle
clear etadot ee kk k momentarm subArea Ig dydt x y z cr

end
```

```
215
216 end
```

```
1  function [wind,sail,hull,pos,foil] = simparam(n,m)
2
3  %-----Vessel Parameters-------%
4
5  hull.char = [6;... % LOA
6               0.5;... % Beam
7               3;... % Gap between hulls, 0 if monohull
8               1;... % Hull Depth
9               1;... % Assumed initial gap between hull and
                   water surface
10              1;... % Draft
11              0.6;... % Midships Coefficient
12              0.5;... % Block Coefficient
13              0.5]; % Prism Coefficient
14
15 hull.weight = 250; % weight of the vessel
16 % Weight of crew, only 2 crew members simulated, assumed
      equal weight of
17 % crew members, below is total weight.
18 hull.crew = 160;
19
20 %-----Sail Parameters---------%
21
22 sail.chord = [0.2 0.75]; % Sail chord, requires two values
23 sail.height = 4.5; % Sail height;
24 sail.NACAS = '0012'; % Sail NACA profile
25 sail.rho = 1; % Air Density
26 sail.densit = 100; % Material density of the sail
27 sail.n = n; % Number of chordwise elements
28 sail.m = m; % Number of spanwise elements
29
30 %-----Foil Parameters--------%
31
32 foil.chord = [0.4 0.4 0.4 0.4];      % Chord lengths
33 foil.angle = [60 -60 90 0];     % Angle of of foils, 0 if not
      used.
34 foil.angle2 = [0 180 0 0];      % Had to rotate port side
      foils by 180
35 foil.fore = 0.6;                % Position relativ COG
36 foil.aft = -hull.char(1)/2;     % Position relativ COGx
37 foil.spans = [4 4 1 2];         % Total Span length of
      different foils
38 foil.NACA = {'2412','2412','0012','0012'}; % NACA profiles
39 foil.rho = 1025;     % Density of water
40 foil.densit = 1000; % Material density of the foil, assumed
```

```
   carbon fibre
foil.n = n; %  Number of elements chordwise
foil.m = m; % Number of elements spanwise
foil.extension = 4.5; % fore foil extension from the hull in
   y direction.

%-----Generating Foil coordinates-----%

pos = vesselgeom(hull,foil,sail);

%-----Wind Condition-------------------%
% Initialising values due to programming.
wind.Vt = -7; % Truw Wind Speed
wind.Va = 0;  % Apparent Wind Speed
wind.beta = 0; % Apparent Wind Angle

end
```

```
function [pos] = vesselgeom(hull,foil,sail)
n = foil.n;
m = foil.m;
hullchar = hull.char;

fangle = foil.angle(1);
foilfor = foil.fore;
foilaft = foil.aft;
spans = foil.spans;

% Determining vessel end points

a = fangle;
x1 = foilfor;
x2 = foilaft;

foilx = [x1 x1 x1 x1 x2 x2 x2 x2]';
beam = hullchar(2)*0.5;
beamgap = hullchar(3)*0.5;
depth = hullchar(4);
airgap = hullchar(5);
draft = hullchar(6);
extension = foil.extension;

y1 = extension;
z1 = 0;
y2 = y1-(spans(1)*cosd(90-a));
z2 = z1-(spans(1)*sind(90-a));

y3 = -y1;
```

```
z3 = z1;

y4 = -y2;
z4 = z2;

y5 = spans(3)/2;
z5 = -spans(4);

y6 = -y5;
z6 = z5;

y7 = 0;
z7 = 0;

y8 = 0;
z8 = z5;

foily = [y1 y2 y3 y4 y5 y6 y7 y8]';
foilz = airgap+[z1 z2 z3 z4 z5 z6 z7 z8]';


fpos = [foilx foily foilz]';
% Didtribution foil points
for k = 1:length(foil.spans)
    if k == 1
        x = linspace(fpos(1,k),fpos(1,k+1),n)';
        y = linspace(fpos(2,k),fpos(2,k+1),n)';
        z = linspace(fpos(3,k),fpos(3,k+1),n)';
    elseif k == 2
        x = [x; linspace(fpos(1,k+1),fpos(1,k+2),n)'];
        y = [y; linspace(fpos(2,k+1),fpos(2,k+2),n)'];
        z = [z; linspace(fpos(3,k+1),fpos(3,k+2),n)'];
    elseif k == 3
        x = [x; linspace(fpos(1,k+2),fpos(1,k+3),n)'];
        y = [y; linspace(fpos(2,k+2),fpos(2,k+3),n)'];
        z = [z; linspace(fpos(3,k+2),fpos(3,k+3),n)'];
    elseif k == 4
        x = [x; linspace(fpos(1,k+3),fpos(1,k+4),n)'];
        y = [y; linspace(fpos(2,k+3),fpos(2,k+4),n)'];
        z = [z; linspace(fpos(3,k+3),fpos(3,k+4),n)'];
    end

end

for k = 1:length(foil.spans)

    [~,~,~,XP,ZP] = foilpoints(cell2mat(foil.NACA(k)),foil.n)
       ;
```

```
78      XP = XP*foil.chord(k);
79      ZP = ZP*foil.chord(k);
80      Volume = foil.spans(k)*polyarea(XP,ZP);
81      mass = Volume*0.3*foil.densit;
82      if k == 1
83          md = ones(n,1)*(mass/n);
84      else
85          md = [md; ones(n,1)*(mass/n)];
86      end
87
88  end
89
90  pos.ff = [foilx foily foilz];
91  pos.foil.cord = [x y z];
92  pos.foil.mass = md;
93  % Distribution hull and sail points
94  [~,~,~,XP,ZP] = foilpoints(sail.NACAS,sail.n);
95  Volume = sail.height*polyarea(XP,ZP);
96  mass = Volume*sail.densit*0.3;
97  sailmass = ones(n,1)*mass/n;
98
99  sailx = (linspace(-max(sail.chord)/2,-min(sail.chord)/2,n));
100 saily = linspace(0,0,n);
101 sailz = linspace(hull.char(5),sail.height+hull.char(5),n);
102
103 pos.sail.cord = [sailx; saily; sailz]';
104 pos.sail.mass = sailmass;
105
106 hullx = linspace(-hull.char(1)/2,hullchar(1)/2,n);
107 hully = zeros(1,n);
108 hullz = ones(n,1)*hull.char(5);
109 [dd,ff] = size(hullx);
110 hullmass = ones(ff,1)*hull.weight/(ff);
111 pos.hull.cord = [hullx; hully; hullz']';
112 pos.hull.mass = hullmass;
113 pos.crit = fpos;
114
115
116 pos.body.sail = pos.sail.cord;
117 pos.space.sail = pos.sail.cord;
118
119 pos.body.hull = pos.hull.cord;
120 pos.space.hull = pos.hull.cord;
121
122 pos.body.foil = pos.foil.cord;
123 pos.space.foil = pos.foil.cord;
124 end
```

```
function [B] = transmatrix(phi,theta,xi)

% This Function creates the rotational matrix

A = [cosd(xi) -sind(xi) 0; sind(xi) cosd(xi) 0; 0 0 1];
C = [cosd(theta) 0 sind(theta); 0 1 0; -sind(theta) 0 cosd(
   theta)];
D = [1 0 0; 0 cosd(phi) -sind(phi); 0 sind(phi) cosd(phi)];

B = A*C*D; % yaw, pitch, roll

end
```

```
function [wind] = VelocityTriangle(Vel,wind,B)

% This function determines the true wind speed and angle.

  Vs = sqrt(Vel(1)^2+Vel(2)^2+Vel(3)^2);
  Vt = wind.Vt;
   wind_angle = wind.angle;
   Va    =   sqrt(Vt^2 + Vs^2 -2*Vt*Vs*cosd(wind_angle));
   beta    = acosd((Vs^2+Va^2 - Vt^2)/(2*Vs*Va));

    W = [-Va*cosd(beta); -Va*sind(beta); 0];

    Wphi = B*W;

   wind.Va = sqrt(Wphi(1)^2+Wphi(2)^2+Wphi(3)^2);

    wind.beta = atand(Wphi(2)/Wphi(1));

end
```

```
function [foil,amr,pos,hull] = geomdef(sail,wind,B,hull,pos,
   foil,Vel)

% This function rotates and translats the geometry of the
   vessel. In
% addition it also determines the moment arms for torque
   calculation.

n = foil.n;
m = foil.n;
k = sail.k;
dt = sail.dt;
hullchar = hull.char;
beta = max(wind.beta);

```

```
13 ffpos = pos.foil.cord; spos = pos.sail.cord; hpos = pos.hull.
      cord;
14 fpos = pos.ff; criteria = pos.crit';
15
16
17 B0 = transmatrix(0,0,0);
18
19 origin = [0; 0; hull.char(5)];
20 % Rotating geometry and translating it.
21 if k == 1
22
23 newcrit = rotgeom(B,origin,criteria,[0 0 0],dt);
24 newfpos = rotgeom(B,origin,fpos,[0 0 0],dt);
25 newffpos = rotgeom(B,origin,ffpos,[0 0 0],dt);
26 newspos = rotgeom(B,origin,spos,[0 0 0],dt);
27 newhpos = rotgeom(B,origin,hpos,[0 0 0],dt);
28 charts = rotgeom(B,origin,spos,[0 0 0],dt);
29 charth = rotgeom(B,origin,hpos,[0 0 0],dt);
30 CG = rotgeom(B,origin,[0 0 hull.char(5)],[0 0 0],dt);
31
32     pos.bod.foil = newfpos;
33     pos.bod.foil2 = newffpos;
34     pos.bod.sail = newspos;
35     pos.bod.hull = newhpos;
36     pos.bod.crit = newcrit;
37
38 else
39
40      vel = Vel';
41
42     bodcrit = pos.bod.crit';
43     bodfpos = pos.bod.foil';
44     bodffpos = pos.bod.foil2';
45     bodspos = pos.bod.sail';
46     bodhpos = pos.bod.hull';
47     bodCG = pos.CG;
48
49     bodcrit = rotgeom(B0,pos.CG,bodcrit,vel,dt);
50     bodfpos = rotgeom(B0,pos.CG,bodfpos,vel,dt);
51     bodffpos = rotgeom(B0,pos.CG,bodffpos,vel,dt);
52     bodspos = rotgeom(B0,pos.CG,bodspos,vel,dt);
53     bodhpos = rotgeom(B0,pos.CG,bodhpos,vel,dt);
54     bodCG = rotgeom(B0,pos.CG,bodCG',vel,dt);
55
56     pos.bod.foil = bodfpos;
57     pos.bod.foil2 = bodffpos;
58     pos.bod.sail = bodspos;
59     pos.bod.hull = bodhpos;
```

```
60      pos.bod.crit = bodcrit;
61      pos.bod.CG = bodCG;
62
63      newfpos = rotgeom(B,pos.CG,bodfpos',[0 0 0],dt);
64      newspos = rotgeom(B,pos.CG,bodspos',[0 0 0],dt);
65      newhpos = rotgeom(B,pos.CG,bodhpos',[0 0 0],dt);
66      newffpos = rotgeom(B,pos.CG,bodffpos',[0 0 0],dt);
67      newcrit = rotgeom(B,pos.CG,bodcrit',[0 0 0],dt);
68
69      charts = rotgeom(B,pos.CG,pos.chart.s',vel,dt);
70      charth = rotgeom(B,pos.CG,pos.chart.h',vel,dt);
71      CG = rotgeom(B0,origin,pos.CG',vel,dt);
72
73  end
74  pos.crit = newcrit;
75  fpos = newffpos;
76  newhpos = newhpos';
77  newspos = newspos';
78
79  % Finding the submerged section of the hydrofoils
80  P0 = (newfpos(:,2))';
81  P1 = (newfpos(:,1))';
82
83  I = plane_line_intersect(P0,P1);
84
85  newfpos(:,1) = I';
86
87  P0 = (newfpos(:,3))';
88  P1 = (newfpos(:,4))';
89
90  I = plane_line_intersect(P0,P1);
91
92  newfpos(:,3) = I';
93
94  P0 = (newfpos(:,7))';
95  P1 = (newfpos(:,8))';
96
97  I = plane_line_intersect(P0,P1);
98
99  newfpos(:,7) = I';
100 % Determining submerged span lengths
101 span1 = sqrt(((newfpos(2,2)-newfpos(2,1))^2)+((newfpos(3,2)-
        newfpos(3,1))^2)+((newfpos(1,2)-newfpos(1,1))^2));
102 span2 = sqrt(((newfpos(2,4)-newfpos(2,3))^2)+((newfpos(3,4)-
        newfpos(3,3))^2)+((newfpos(1,4)-newfpos(1,3))^2));
103 span3 = sqrt(((newfpos(2,6)-newfpos(2,5))^2)+((newfpos(3,6)-
        newfpos(3,5))^2)+((newfpos(1,6)-newfpos(1,5))^2));
104 span4 = sqrt(((newfpos(2,8)-newfpos(2,7))^2)+((newfpos(3,8)-
```

```
   newfpos(3,7))^2)+((newfpos(1,8)-newfpos(1,7))^2));
105
106 foil.span = [span1 span2 span3 span4];
107 % Determining the centre of the hydrofoils
108 arm1 = ((newfpos(:,1)+newfpos(:,2))/2)-CG;
109 arm2 = ((newfpos(:,3)+newfpos(:,4))/2)-CG;
110 arm3 = ((newfpos(:,5)+newfpos(:,6))/2)-CG;
111 arm4 = ((newfpos(:,8)+newfpos(:,7))/2)-CG;
112
113 amr.f = [arm1 arm2 arm3 arm4];
114 % Determining the maximum moment arm obtainable from the
      hydrofoils. Some
115 % vector manipulation is performed.
116 fpos2 = fpos';
117 px = reshape(fpos2(:,1),10,[]);
118 py = reshape(fpos2(:,2),10,[]);
119 pz = reshape(fpos2(:,3),10,[]);
120
121 maxmomentarm = [mean(px(:,1)) mean(px(:,2)) mean(px(:,3))
      mean(px(:,4));...
122                 mean(py(:,1)) mean(py(:,2)) mean(py(:,3))
                       mean(py(:,4));...
123                 mean(pz(:,1)) mean(pz(:,2)) mean(pz(:,3))
                       mean(pz(:,4))];
124 % Determining moment arm of sail. Centroid of sail, not
      perfectly accurate
125 % but sufficient.
126 x = linspace(min(sail.chord),max(sail.chord),n);
127 z = linspace(0,sail.height,m)+hull.char(5);
128
129 XCE = sum(x)/length(x);
130 ZCE = (sum(z)/length(z))/2;
131
132 spos(:,2) = ones(n,1)*XCE*abs(sind(beta));
133
134 amr.sail = [XCE/4; XCE*abs(sind(beta)); ZCE];
135
136 L = hullchar(1);
137 B = hullchar(2);
138 D = hullchar(4);
139 Cm = hullchar(7);
140 Cb = hullchar(8);
141 Cp = hullchar(9);
142
143 hull.area = 2*L*D*Cp*Cm*Cb*B*0.6;
144
145 amr.hull = [0; 0; 0];
146
```

```
147 pos.CG = CG;
148 pos.ff = newfpos;
149 pos.chart.s = charts;
150 pos.chart.h = charth;
151 pos.sail.cord = newspos;
152 pos.hull.cord = newhpos;
153 pos.foil.cord = fpos';
154 hull.mass = sum(pos.hull.mass)+sum(pos.sail.mass)+sum(pos.
       foil.mass)+hull.crew;
155
156 % Determining the moment arms for the different hydrofoils.
157 if k == 1
158    maxmomentarm(3,:) = -maxmomentarm(3,:)-1;
159    pos.momentarm.f = maxmomentarm;
160    pos.momentarm.sail = amr.sail;
161    pos.momentarm.hull = amr.hull;
162 end
163
164 end
```

```
1  function newpos = rotgeom(B,origin,pos,vel,dt)
2
3
4  [dd,ff] = size(pos);
5
6  center = repmat(origin,1,dd)';
7  pos = pos-center;
8  for k = 1:dd
9      if k == 1
10     newpos = B*(pos(k,:)');
11     else
12        newpos = [newpos B*(pos(k,:)')];
13     end
14
15 end
16 newpos = newpos+center';
17 vel = vel.*dt;
18 [dd,ff] = size(newpos);
19 newpos = newpos+repmat(vel,ff,1)';
20
21 end
```

```
1  function I=plane_line_intersect(P0,P1)
2  %plane_line_intersect computes the intersection of a plane
       and a segment(or
3  %a straight line)
4  % Inputs:
5  %       n: normal vector of the Plane
```

```
%        V0: any point that belongs to the Plane
%        P0: end point 1 of the segment P0P1
%        P1:  end point 2 of the segment P0P1
%
%Outputs:
%       I    is the point of interection
%     Check is an indicator:
%       0 => disjoint (no intersection)
%       1 => the plane intersects P0P1 in the unique point I
%       2 => the segment lies in the plane
%       3 => the intersection lies outside the segment P0P1
%
% Example:
% Determine the intersection of following the plane x+y+z+3=0
    with the segment P0P1:
% The plane is represented by the normal vector n=[1 1 1]
% and an arbitrary point that lies on the plane, ex: V0=[1 1
   -5]
% The segment is represented by the following two points
% P0=[-5 1 -1]
%P1=[1 2 3]
% [I,check]=plane_line_intersect([1 1 1],[1 1 -5],[-5 1
   -1],[1 2 3]);
%This function is written by :
%                                Nassim Khaled
%                                Wayne State University
%                                Research Assistant and Phd
   candidate
%If you have any comments or face any problems, please feel
   free to leave
%your comments and i will try to reply to you as fast as
   possible.

n = [0 0 1];
V0  = [1 1 0];
I=[0 0 0];
u = P1-P0;
w = P0 - V0;
D = dot(n,u);
N = -dot(n,w);
check=0;
if abs(D) < 10^-7        % The segment is parallel to plane
        if N == 0           % The segment lies in plane
            check=2;
            return
        else
            check=0;       %no intersection
            return
```

```
        end
end
%compute the intersection parameter
sI = N / D;
I = P0+ sI.*u;
if (sI < 0 || sI > 1)
    check= 3;           %The intersection point  lies outside
       the segment, so there is no intersection
else
    check=1;
end
```

```
function [X,Y,N,X2,Y2] = foilpoints(NACA,n)


Minit = str2double(NACA(1));
Pinit = str2double(NACA(2));
Tinit = str2double(NACA(3:4));

gridPts = n;

M = Minit / 100;
P = Pinit / 10;
T = Tinit / 100;

x = linspace(0,1,gridPts)';

a0 = 0.2969; a1 = -0.126; a2 = -0.3516; a3 = 0.2843; a4 =
   -0.1015;

theta = zeros(gridPts,1); yc = zeros(gridPts,1);
dyc_dx = zeros(gridPts,1);

for i = 1:1:gridPts

    if (x(i) >= 0 && x(i) < P)

        yc(i) = (M/P^2)*((2*P*x(i))-x(i)^2);

        dyc_dx(i) =  ((2*M)/(P^2))*(P-x(i));

    elseif (x(i) >= P &&  x(i) <=1)

        yc(i) =  (M/(1-P)^2)*(1-(2*P)+(2*P*x(i))-(x(i)^2));

        dyc_dx(i) =  ((2*M)/((1-P)^2))*(P-x(i));
    end

```

```
     theta(i) = atan(dyc_dx(i));

end


    term0 = a0 .* sqrt(x);
    term1 = a1 .* x;
    term2 = a2 .* x.^2;
    term3 = a3 .* x.^3;
    term4 = a4 .* x.^4;
    yt = 5 * T .* (term0 + term1 + term2 + term3 + term4);

    xu = x - yt .* sin(theta);

    yu = yc + yt .* cos(theta);

    xl = x + yt .* sin(theta);

    yl = yc - yt .* cos(theta);

X = xu;
Y = (yu+yl)/2;
X2 = [xu; (fliplr(xl'))'];
Y2 = [yu; (fliplr(yl'))'];
Nx = sin(theta);
Nz = cos(theta);
N = [Nx,Nz];
end
```

```
function aero = liftsail(sail,wind,dt,k,aero,pos,simstart)
% This function determines the lift and drag produced by a
   sail.
n = sail.n;
m = sail.m;
Vel = [wind.Va*cosd(11); wind.Va*sind(11); 0];
% De-powering the sail if a certain roll angle has been
   reached.
if sail.p > 1.5
    Vel = [wind.Va*cosd(7); wind.Va*sind(7); 0];
end
Vel = -Vel;
if k == 1

    res.k = 1;
    domain.CG = pos.CG;
    domain.rot = aero.rot;
    domain.vel = Vel;
    domain.dt = dt;
```

```
18      domain.k = k;
19      domain =  geomsail(sail,domain);
20      domain.rho = 1.25;
21      domain.dt = dt;
22      domain.mm = m; domain.nn = n;
23
24      domain.test.x2 = domain.test.x;
25      domain.test.y2 = domain.test.y;
26      domain.test.z2 = domain.test.z;
27
28      domain.ang.phi = 90;
29      domain.ang.the = 0;
30
31      [dd,ff] = size(domain.test.x2);
32      res.wake = ones(dd-1,ff-1);
33
34      [res] = UVLM(Vel,domain,res);
35      res.wake = res.circ(n,:);
36
37      domain.chart.x = [domain.test.x(1,1) domain.test.x(1,m)];
38      domain.chart.y = [domain.test.y(1,1) domain.test.z(1,m)];
39      domain.chart.z = [domain.test.y(1,1) domain.test.z(1,m)];
40
41
42  else
43
44      domain = aero.domain;
45      domain.rot = aero.rot;
46      domain.CG = pos.CG;
47      domain.vel = Vel;
48      domain.dt = dt;
49      domain.k = k;
50      domain = geomsail(sail,domain);
51
52      res = aero.res;
53      res.k = k;
54      domain = diplacefast(domain,Vel);
55      [res] = UVLM(Vel,domain,res);
56      domain.test.x2 = [domain.test.x(1,:); domain.test.x2];
57      domain.test.y2 = [domain.test.y(1,:); domain.test.y2];
58      domain.test.z2 = [domain.test.z(1,:); domain.test.z2];
59      res.wake = [res.circ(n,:); res.wake];
60
61  end
62
63  cutoff = simstart;
64  if k > cutoff
65
```

```
    res.wake(cutoff-1:end,:) = [];
    domain.test.x2(cutoff:end,:) = [];
    domain.test.y2(cutoff:end,:) = [];
    domain.test.z2(cutoff:end,:) = [];

end

aero.domain = domain;
aero.res = res;
end
```

```
function [domain] = geomsail(sail,domain)

n = sail.n;
m = sail.m;
span = sail.height;
sailchord = sail.chord;
NACA = sail.NACAS;

xg = domain.CG(1);
yg = domain.CG(2);
zg = domain.CG(3);

chord = abs(linspace(-max(sailchord),-min(sailchord),m+1));
[XP,ZP,~] = foilpoints(NACA,n+1);
XP = XP.*chord+domain.vel(1)*domain.dt*domain.k;
ZP = ZP.*chord;
YS = linspace(0,span,m+1)+1;
wlength = 400;
wlength2 = sail.vel(1)*sail.dt;
domain.case = 0;

B = transmatrix(90,180,180);

xp = XP;
ys = repmat(YS,n+1,1);
zp = ZP;
% Creating wake points
x2 = [xp(n+1,:); ones(1,m+1).*(xp(n+1)+wlength)];
y2 = [YS; YS];
z2 = [ones(1,m+1).*zp(n+1); ones(1,m+1).*zp(n+1)];
% Discretising the geometry of the sail
x = [xp(n+1,:); ones(1,m+1).*(xp(n+1)+wlength2)];
y = [YS; YS];
z = [ones(1,m+1).*zp(n+1); ones(1,m+1).*zp(n+1)];

% Finding control points.
for j = 1:m
```

```
38
39      for k = 1:n
40
41          if (j == m && k == n)
42
43              P = [XP(k,j) YS(j) ZP(k,j); XP(k,j+1) YS(j+1) ZP(
                    k,j+1);...
44            XP(k+1,j+1) YS(j+1) ZP(k+1,j+1); XP(k+1,j) YS(j) ZP
                  (k+1,j)];
45
46
47          WW = [P(4,1) P(4,2) P(4,3); P(3,1) P(3,2) P(3,3);...
48            wlength P(3,2) P(3,3); wlength P(4,2) P(4,3)];
49
50
51          elseif j == m
52
53            P = [XP(k,j) YS(j) ZP(k,j); XP(k,j+1) YS(j+1) ZP(k,
                  j+1);...
54            XP(k+1,j+1) YS(j+1) ZP(k+1,j+1); XP(k+1,j) YS(j) ZP
                  (k+1,j)];
55
56          elseif k == n
57
58            P = [XP(k,j) YS(j) ZP(k,j); XP(k,j+1) YS(j+1) ZP(k,
                  j+1);...
59            XP(k+1,j+1) YS(j+1) ZP(k+1,j+1); XP(k+1,j) YS(j) ZP
                  (k+1,j)];
60
61          WW = [P(4,1) P(4,2) P(4,3); P(3,1) P(3,2) P(3,3);...
62            wlength P(3,2) P(3,3); wlength P(4,2) P(4,3)];
63
64          else
65
66            P = [XP(k,j) YS(j) ZP(k,j); XP(k,j+1) YS(j+1) ZP(k,
                  j+1);...
67            XP(k+1,j+1) YS(j+1) ZP(k+1,j+1); XP(k+1,j) YS(j) ZP
                  (k+1,j)];
68
69          end
70
71          % Rotating based on sail orientation
72
73          [Px,Py,Pz] = rotgeom2(B,sail.pos,P(:,1),P(:,2),P(:,3)
                ,0,0,0);
74      P = [Px Py Pz];
75
76      domain.CP{k,j} = mean(P);
```

```
    domain.P{k,j} = P;
    [domain.N{k,j}, S(k,j)] = normalvec(P);

    end

    domain.W{j} = WW;

end

% Rotating sail based on orientation
[x2,y2,z2] = rotgeom2(B,sail.pos,x2,y2,z2,0,0,0);
[x,y,z] = rotgeom2(B,sail.pos,x,y,z,0,0,0);
[xp,ys,zp] = rotgeom2(B,sail.pos,xp,ys,zp,0,0,0);

domain.S = S;
domain.wake.x = x2;
domain.wake.y = y2;
domain.wake.z = z2;
domain.test.x = x;
domain.test.y = y;
domain.test.z = z;
domain.test2.x = xp;
domain.test2.y = ys;
domain.test2.z = zp;


end
```

```
function [N,S] = normalvec(P)
% This function determines the normal vector and surface area
    of each
% element
A = P(3,:)-P(1,:); B = P(2,:)-P(4,:);
% N2 = cross(A,B);
X = A(2)*B(3)-A(3)*B(2); Y = A(3)*B(1)-A(1)*B(3); Z = A(1)*B
   (2)-A(2)*B(1);

A2 = sqrt(X^2+Y^2+Z^2);

N = [X/A2; Y/A2; Z/A2];

E = P(2,:)-P(1,:); F = P(4,:)-P(1,:);

S11 = F(2)*B(3)-F(3)*B(2);
S12 = F(3)*B(1)-F(1)*B(3);
S13 = F(1)*B(2)-F(2)*B(1);
S21 = E(2)*B(3)-E(3)*B(2);
S22 = E(1)*B(3)-E(3)*B(1);
```

```
S23 = E(1)*B(2)-E(2)*B(1);

S = 0.5*(sqrt(S11^2+S12^2+S13^2)+sqrt(S21^2+S22^2+S23^2));

end
```

```
function [x,y,z] = rotgeom2(B,pos,x,y,z,xg,yg,zg)

[a,b] = size(x);
xg = ones(a,b)*xg;
yg = ones(a,b)*yg;
zg = ones(a,b)*zg;
x = x-xg;
y = y-yg;
z = z-zg;
for i = 1:a
    for j = 1:b

     trans = B*[x(i,j); y(i,j); z(i,j)];
     x(i,j) = trans(1)+pos(1);
     y(i,j) = trans(2)+pos(2);
     z(i,j) = trans(3)+pos(3);
    end
end

x = x+xg;
y = y+yg;
z = z+zg;
end
```

```
function res = UVLM(Vel,domain,res)
tic
m = domain.mm;
n = domain.nn;
d = 0;
nn = cell2mat(domain.N');
nn = nn(:);
nn = reshape(nn,3,[])';
dy = abs(diff(abs(domain.test2.y(1,:))));
dy = repmat(dy,n,1);
dx = abs(diff(abs(domain.test2.x(:,1))));
dx = repmat(dx,1,m);

% Creating incoming velocity vector for
Vx = Vel(1).*linspace(0,1,n).^2;
    Vy = Vel(2).*linspace(0,1,n).^2;
    Vz = Vel(3).*linspace(0,1,n).^2;
    Vair = [Vx; -Vy; Vz]';
```

```
% Changing which direction the spanlength is determined
if domain.ang.phi == 90
    asd = 1;
    dy = abs(diff(abs(domain.test2.z(1,:))));
    dy = repmat(dy,n,1);
end

if domain.ang.the == 180

   Vel(3) = -Vel(3);
end

V2 = Vel;


for kk = 1:n
    for jj = 1:m

    d = d+1;
    cp = cell2mat(domain.CP(kk,jj));
    % Induced Velocity from the body onto the body
    vel = inducedfast(cp,domain);
    % Induced Velocity from the wake onto the body
    vel2 = inducedfastwake(cp,domain);
    % Induced Downwash from the body onto the body
    vel3 = inducedfastdrag(cp,domain);
    % Induced Downwash from the wake onto the body
    vel4 = inducedfastdragwake(cp,domain);

    vel(end-m+1:end,:) = vel(end-m+1:end,:)+vel2;

    vel3(end-m+1:end,:) = vel3(end-m+1:end,:)+vel4;
    % Was unable to simulate the free surface as geometry
       definition was
    % not optimal. i.e foil was sticking above z = 0, even
       though it
    % shouldn't
%     if domain.case == 1
% % Determining effect of free surface
%     velim = inducedmirror(cp,domain);
%     velim(:,3) = -velim(:,3);
%     vel = vel+velim;

%     end
    % Establishing influence matrix and downwash influence
       matrix
LHSa = vel*nn(d,:)';

```

```
LHSb = vel3*nn(d,:)';
    % Perturbation potential
    Vw = inducedvel3(cp,domain,res);
    % Perturbation Potential + Inflow Potential
    V = V2'-Vw;
    % Neumann Dirichlett Boundary Condition V phi * n = 0
    if domain.case == 0
        % Used if sail
        RHS(d) = -Vair(kk,:)*nn(d,:)';

    else
        % Used if hydrofoil
        RHS(d) = -V*nn(d,:)';

    end

    if d == 1
       Vt = V;
       a = LHSa';
       b = LHSb';
    else
        Vt = [Vt; V];
        a = [a; LHSa'];
        b = [b; LHSb'];
    end
    end
end
% Solving the boundary condition for circulation
circulation = a\RHS';
drag = b*circulation;
circulation = reshape(circulation,m,[])';
drag = reshape(drag,m,[])';

    if res.k < 2
    changecirc = zeros(kk,jj);
    else
        oldcirc = res.circ;
        changecirc = (circulation-oldcirc)./domain.dt;
    end

    d = 0;
    circulation = circulation';
    dx = dx';
    dy = dy';
    changecirc = changecirc';
    [kk, jj] = size(circulation);
    dp = zeros(kk,jj);
    % Determining the pressure on each element
```

```
112 for j = 1:n
113
114     for i = 1:m
115
116     d = d+1;
117     np = null(nn(d,:));
118     npy = np(:,1); npx = np(:,2);
119         if i == 1 && j > 1
120     dp(i,j) =  domain.rho*(((Vt(d,:)*npx*circulation(i,j))/dx
           (i,j))...
121               +(Vt(d,:)*npy*(circulation(i,j)-circulation(i,j
                     -1))/dy(i,j))+changecirc(i,j));
122
123         elseif j == 1 && i > 1
124     dp(i,j) =  domain.rho*((Vt(d,:)*npx*(circulation(i,j)-
           circulation(i-1,j))/dx(i,j))...
125          +(Vt(d,:)*npy*(circulation(i,j))/dy(i,j))+changecirc(
                i,j));
126         elseif i == 1 && j == 1
127     dp(i,j) =  domain.rho*(((Vt(d,:)*npx*circulation(i,j))/dx
           (i,j))...
128          +(Vt(d,:)*npy*(circulation(i,j))/dy(i,j))+changecirc(
                i,j));
129         else
130     dp(i,j) =  domain.rho*((Vt(d,:)*npx*(circulation(i,j)-
           circulation(i-1,j))/dx(i,j))...
131               +(Vt(d,:)*npy*(circulation(i,j)-circulation(i,j
                     -1))/dy(i,j))+changecirc(i,j));
132
133         end
134
135         if i > m/2
136
137             if i == m && j < n
138     dp(i,j) =  domain.rho*(((Vt(d,:)*npx*circulation(i,j))/dx
           (i,j))...
139               +(Vt(d,:)*npy*(circulation(i,j)-circulation(i,j
                     +1))/dy(i,j))+changecirc(i,j));
140
141         elseif j == n && i < m
142     dp(i,j) =  domain.rho*((Vt(d,:)*npx*(circulation(i,j)-
           circulation(i+1,j))/dx(i,j))...
143          +(Vt(d,:)*npy*(circulation(i,j))/dy(i,j))+changecirc(
                i,j));
144         elseif i == m && j == n
145     dp(i,j) =  domain.rho*(((Vt(d,:)*npx*circulation(i,j))/dx
           (i,j))...
146          +(Vt(d,:)*npy*(circulation(i,j))/dy(i,j))+changecirc(
```

```
            i,j));
147         else
148     dp(i,j) =  domain.rho*((Vt(d,:)*npx*(circulation(i,j)-
            circulation(i+1,j))/dx(i,j))...
149               +(Vt(d,:)*npy*(circulation(i,j)-circulation(i,j
                  +1))/dy(i,j))+changecirc(i,j));
150
151             end
152
153         end
154
155     end
156 end
157
158 S = domain.S.';
159 S = S(:);
160 res.dP = dp;
161 dp = dp(:);
162 dF = -S.*dp.*nn;
163 F = sum(dF);
164
165 res.Ly = F(2);
166 res.Lz = F(3);
167 if domain.ang.the == 180
168
169  res.Ly = -F(2);
170 end
171
172 res.D = F(1);
173 res.t = toc;
174 res.RHS = RHS;
175 res.circ = circulation;
176 res.LHSa = a;
177 res.LHSb = b;
178 res.k = res.k + 1;
179
180 end
```

```
1  function vel = inducedvel3(CP,domain,sav)
2
3  x = domain.test.x2;
4  y = domain.test.y2;
5  z = domain.test.z2;
6  gamma = sav.wake;
7
8  ax = x(1:1:end-1,1:1:end-1); ay = y(1:1:end-1,1:1:end-1); az
       = z(1:1:end-1,1:1:end-1);
9  bx = x(1:1:end-1,2:1:end); by = y(1:1:end-1,2:1:end); bz = z
```

```
   (1:1:end-1,2:1:end);
cx = x(2:1:end,2:1:end); cy = y(2:1:end,2:1:end); cz = z(2:1:
   end,2:1:end);
dx = x(2:1:end,1:1:end-1); dy = y(2:1:end,1:1:end-1); dz = z
   (2:1:end,1:1:end-1);

xa = [dx' ax' bx' cx']; xb = [ax' bx' cx' dx'];
ya = [dy' ay' by' cy']; yb = [ay' by' cy' dy'];
za = [dz' az' bz' cz']; zb = [az' bz' cz' dz'];
[c,d] = size(xa);
gamma = repmat(gamma',1,4);
e = 1e-10;

xp = CP(1); yp = CP(2); zp = CP(3);

R1R2x2 = (yp-ya).*(zp-zb)-(zp-za).*(yp-yb);
R1R2y2 = -((xp-xa).*(zp-zb))+(zp-za).*(xp-xb);
R1R2z2 = (xp-xa).*(yp-yb)-(yp-ya).*(xp-xb);

R1R22 = R1R2x2.^2+R1R2y2.^2+R1R2z2.^2;

r12 = sqrt((xp-xa).^2+(yp-ya).^2+(zp-za).^2);
r22 = sqrt((xp-xb).^2+(yp-yb).^2+(zp-zb).^2);

r0r12 = (xb-xa).*(xp-xa)+(yb-ya).*(yp-ya)+(zb-za).*(zp-za);
r0r22 = (xb-xa).*(xp-xb)+(yb-ya).*(yp-yb)+(zb-za).*(zp-zb);
K2 = gamma.*((r0r12./r12)-(r0r22./r22))./(4.*pi.*R1R22);

u = K2.*R1R2x2;
v = K2.*R1R2y2;
w = K2.*R1R2z2;

u(isnan(u)) = 0;
v(isnan(v)) = 0;
w(isnan(w)) = 0;
cond = (find(R1R22 < e));
u(cond) = 0;
v(cond) = 0;
w(cond) = 0;

    velx = sum(sum(u));
    vely = sum(sum(v));
    velz = sum(sum(w));

vel = [velx vely velz];

end
```

```
function vel = inducedfast(CP,domain)
m = domain.mm;
n = domain.nn;

% This function calculates the induced velocity from a surface distribution
% of vortex rings at a point P. For references see the master thesis or
% Joseph Katz Low Speed Aerodynamics section 10.4.6
% The code requires a point P with (x,y,z) and 3 vectors which determine
% every single corner point of the vortex distribution across the surface
% n and m are number of elements in chord wise and span wise direction
% respectively.


x = domain.test2.x;
y = domain.test2.y;
z = domain.test2.z;
gamma = 1;
% Splitting the geometry into (i,j),(i+1,j),(i+1,j+1) and (i+1,j)
ax = x(1:1:end-1,1:1:end-1); ay = y(1:1:end-1,1:1:end-1); az = z(1:1:end-1,1:1:end-1);
bx = x(1:1:end-1,2:1:end); by = y(1:1:end-1,2:1:end); bz = z(1:1:end-1,2:1:end);
cx = x(2:1:end,2:1:end); cy = y(2:1:end,2:1:end); cz = z(2:1:end,2:1:end);
dx = x(2:1:end,1:1:end-1); dy = y(2:1:end,1:1:end-1); dz = z(2:1:end,1:1:end-1);

xa = [dx' ax' bx' cx']; xb = [ax' bx' cx' dx'];
ya = [dy' ay' by' cy']; yb = [ay' by' cy' dy'];
za = [dz' az' bz' cz']; zb = [az' bz' cz' dz'];
e = 1e-10;

xp = CP(1); yp = CP(2); zp = CP(3);

R1R2x2 = (yp-ya).*(zp-zb)-(zp-za).*(yp-yb);
R1R2y2 = -((xp-xa).*(zp-zb))+(zp-za).*(xp-xb);
R1R2z2 = (xp-xa).*(yp-yb)-(yp-ya).*(xp-xb);

R1R22 = R1R2x2.^2+R1R2y2.^2+R1R2z2.^2;

r12 = sqrt((xp-xa).^2+(yp-ya).^2+(zp-za).^2);
r22 = sqrt((xp-xb).^2+(yp-yb).^2+(zp-zb).^2);
```

```

r0r12 = (xb-xa).*(xp-xa)+(yb-ya).*(yp-ya)+(zb-za).*(zp-za);
r0r22 = (xb-xa).*(xp-xb)+(yb-ya).*(yp-yb)+(zb-za).*(zp-zb);
K2 = gamma.*((r0r12./r12)-(r0r22./r22))./(4.*pi.*R1R22);

u = K2.*R1R2x2;
v = K2.*R1R2y2;
w = K2.*R1R2z2;

% Removing singularities

u(isnan(u)) = 0;
v(isnan(v)) = 0;
w(isnan(w)) = 0;
cond = (find(R1R22 < e));
u(cond) = 0;
v(cond) = 0;
w(cond) = 0;

% Rearranging data for future use

tx = u(:); ty = v(:); tz = w(:);
tx2 = reshape(tx,n*m,[]); ty2 = reshape(ty,n*m,[]); tz2 =
   reshape(tz,n*m,[]);
tx3 = sum(tx2'); ty3 = sum(ty2'); tz3 = sum(tz2');

vel = [tx3' ty3' tz3'];

end
```

```
function vel = inducedfastdrag(CP,domain)

m = domain.mm;
n = domain.nn;
x = domain.test2.x;
y = domain.test2.y;
z = domain.test2.z;
gamma = 1;

ax = x(1:1:end-1,1:1:end-1); ay = y(1:1:end-1,1:1:end-1); az
   = z(1:1:end-1,1:1:end-1);
bx = x(1:1:end-1,2:1:end); by = y(1:1:end-1,2:1:end); bz = z
   (1:1:end-1,2:1:end);
cx = x(2:1:end,2:1:end); cy = y(2:1:end,2:1:end); cz = z(2:1:
   end,2:1:end);
dx = x(2:1:end,1:1:end-1); dy = y(2:1:end,1:1:end-1); dz = z
   (2:1:end,1:1:end-1);

```

```
xa = [bx' dx']; xb = [cx' ax'];
ya = [by' dy']; yb = [cy' ay'];
za = [bz' dz']; zb = [cz' az'];
e = 1e-10;

xp = CP(1); yp = CP(2); zp = CP(3);

R1R2x2 = (yp-ya).*(zp-zb)-(zp-za).*(yp-yb);
R1R2y2 = -((xp-xa).*(zp-zb))+(zp-za).*(xp-xb);
R1R2z2 = (xp-xa).*(yp-yb)-(yp-ya).*(xp-xb);

R1R22 = R1R2x2.^2+R1R2y2.^2+R1R2z2.^2;

r12 = sqrt((xp-xa).^2+(yp-ya).^2+(zp-za).^2);
r22 = sqrt((xp-xb).^2+(yp-yb).^2+(zp-zb).^2);

r0r12 = (xb-xa).*(xp-xa)+(yb-ya).*(yp-ya)+(zb-za).*(zp-za);
r0r22 = (xb-xa).*(xp-xb)+(yb-ya).*(yp-yb)+(zb-za).*(zp-zb);
K2 = gamma.*((r0r12./r12)-(r0r22./r22))./(4.*pi.*R1R22);

u = K2.*R1R2x2;
v = K2.*R1R2y2;
w = K2.*R1R2z2;

u(isnan(u)) = 0;
v(isnan(v)) = 0;
w(isnan(w)) = 0;
cond = (find(R1R22 < e));
u(cond) = 0;
v(cond) = 0;
w(cond) = 0;

vel.x = u;
vel.y = v;
vel.z = w;

    tx = vel.x(:); ty = vel.y(:); tz = vel.z(:);
tx2 = reshape(tx,2,[]); ty2 = reshape(ty,2,[]); tz2 = reshape
   (tz,2,[]);
tx3 = sum(tx2); ty3 = sum(ty2); tz3 = sum(tz2);
vel = [tx3' ty3' tz3'];

end
```

```
function vel = inducedfastdragwake(CP,domain)

m = domain.mm;
n = domain.nn;
```

```
x = domain.wake.x;
y = domain.wake.y;
z = domain.wake.z;
gamma = 1;

ax = x(1:1:end-1,1:1:end-1); ay = y(1:1:end-1,1:1:end-1); az = z(1:1:end-1,1:1:end-1);
bx = x(1:1:end-1,2:1:end); by = y(1:1:end-1,2:1:end); bz = z(1:1:end-1,2:1:end);
cx = x(2:1:end,2:1:end); cy = y(2:1:end,2:1:end); cz = z(2:1:end,2:1:end);
dx = x(2:1:end,1:1:end-1); dy = y(2:1:end,1:1:end-1); dz = z(2:1:end,1:1:end-1);

xa = [bx' dx']; xb = [cx' ax'];
ya = [by' dy']; yb = [cy' ay'];
za = [bz' dz']; zb = [cz' az'];
e = 1e-10;

xp = CP(1); yp = CP(2); zp = CP(3);

R1R2x2 = (yp-ya).*(zp-zb)-(zp-za).*(yp-yb);
R1R2y2 = -((xp-xa).*(zp-zb))+(zp-za).*(xp-xb);
R1R2z2 = (xp-xa).*(yp-yb)-(yp-ya).*(xp-xb);

R1R22 = R1R2x2.^2+R1R2y2.^2+R1R2z2.^2;

r12 = sqrt((xp-xa).^2+(yp-ya).^2+(zp-za).^2);
r22 = sqrt((xp-xb).^2+(yp-yb).^2+(zp-zb).^2);

r0r12 = (xb-xa).*(xp-xa)+(yb-ya).*(yp-ya)+(zb-za).*(zp-za);
r0r22 = (xb-xa).*(xp-xb)+(yb-ya).*(yp-yb)+(zb-za).*(zp-zb);
K2 = gamma.*((r0r12./r12)-(r0r22./r22))./(4.*pi.*R1R22);

u = K2.*R1R2x2;
v = K2.*R1R2y2;
w = K2.*R1R2z2;

u(isnan(u)) = 0;
v(isnan(v)) = 0;
w(isnan(w)) = 0;
cond = (find(R1R22 < e));
u(cond) = 0;
v(cond) = 0;
w(cond) = 0;

vel.x = u;
vel.y = v;
```

```
vel.z = w;

 tx = vel.x(:); ty = vel.y(:); tz = vel.z(:);
tx2 = reshape(tx,2,[]); ty2 = reshape(ty,2,[]); tz2 = reshape
   (tz,2,[]);
tx3 = sum(tx2); ty3 = sum(ty2); tz3 = sum(tz2);
vel = [tx3' ty3' tz3'];


end
```

```
function vel = inducedfastwake(CP,domain)

m = domain.mm;
n = domain.nn;
x = domain.wake.x;
y = domain.wake.y;
z = domain.wake.z;
gamma = 1;

ax = x(1:1:end-1,1:1:end-1); ay = y(1:1:end-1,1:1:end-1); az
   = z(1:1:end-1,1:1:end-1);
bx = x(1:1:end-1,2:1:end); by = y(1:1:end-1,2:1:end); bz = z
   (1:1:end-1,2:1:end);
cx = x(2:1:end,2:1:end); cy = y(2:1:end,2:1:end); cz = z(2:1:
   end,2:1:end);
dx = x(2:1:end,1:1:end-1); dy = y(2:1:end,1:1:end-1); dz = z
   (2:1:end,1:1:end-1);

xa = [dx' ax' bx' cx']; xb = [ax' bx' cx' dx'];
ya = [dy' ay' by' cy']; yb = [ay' by' cy' dy'];
za = [dz' az' bz' cz']; zb = [az' bz' cz' dz'];
e = 1e-10;

xp = CP(1); yp = CP(2); zp = CP(3);

R1R2x2 = (yp-ya).*(zp-zb)-(zp-za).*(yp-yb);
R1R2y2 = -((xp-xa).*(zp-zb))+(zp-za).*(xp-xb);
R1R2z2 = (xp-xa).*(yp-yb)-(yp-ya).*(xp-xb);

R1R22 = R1R2x2.^2+R1R2y2.^2+R1R2z2.^2;

r12 = sqrt((xp-xa).^2+(yp-ya).^2+(zp-za).^2);
r22 = sqrt((xp-xb).^2+(yp-yb).^2+(zp-zb).^2);

r0r12 = (xb-xa).*(xp-xa)+(yb-ya).*(yp-ya)+(zb-za).*(zp-za);
r0r22 = (xb-xa).*(xp-xb)+(yb-ya).*(yp-yb)+(zb-za).*(zp-zb);
K2 = gamma.*((r0r12./r12)-(r0r22./r22))./(4.*pi.*R1R22);
```

```

u = K2.*R1R2x2;
v = K2.*R1R2y2;
w = K2.*R1R2z2;

u(isnan(u)) = 0;
v(isnan(v)) = 0;
w(isnan(w)) = 0;
cond = (find(R1R22 < e));
u(cond) = 0;
v(cond) = 0;
w(cond) = 0;

vel.x = u;
vel.y = v;
vel.z = w;

tx = vel.x(:); ty = vel.y(:); tz = vel.z(:);
tx2 = reshape(tx,m,[]); ty2 = reshape(ty,m,[]); tz2 = reshape
   (tz,m,[]);
tx3 = sum(tx2'); ty3 = sum(ty2'); tz3 = sum(tz2');
vel = [tx3' ty3' tz3'];

end
```

```
function vel = inducedmirror(CP,domain)
m = domain.mm;
n = domain.nn;

x = domain.test2.x;
y = domain.test2.y;
z = -domain.test2.z;
gamma = 1;

ax = x(1:1:end-1,1:1:end-1); ay = y(1:1:end-1,1:1:end-1); az
   = z(1:1:end-1,1:1:end-1);
bx = x(1:1:end-1,2:1:end); by = y(1:1:end-1,2:1:end); bz = z
   (1:1:end-1,2:1:end);
cx = x(2:1:end,2:1:end); cy = y(2:1:end,2:1:end); cz = z(2:1:
   end,2:1:end);
dx = x(2:1:end,1:1:end-1); dy = y(2:1:end,1:1:end-1); dz = z
   (2:1:end,1:1:end-1);

xa = [dx' ax' bx' cx']; xb = [ax' bx' cx' dx'];
ya = [dy' ay' by' cy']; yb = [ay' by' cy' dy'];
za = [dz' az' bz' cz']; zb = [az' bz' cz' dz'];
e = 1e-10;

```

```
xp = CP(1); yp = CP(2); zp = CP(3);

R1R2x2 = (yp-ya).*(zp-zb)-(zp-za).*(yp-yb);
R1R2y2 = -((xp-xa).*(zp-zb))+(zp-za).*(xp-xb);
R1R2z2 = (xp-xa).*(yp-yb)-(yp-ya).*(xp-xb);

R1R22 = R1R2x2.^2+R1R2y2.^2+R1R2z2.^2;

r12 = sqrt((xp-xa).^2+(yp-ya).^2+(zp-za).^2);
r22 = sqrt((xp-xb).^2+(yp-yb).^2+(zp-zb).^2);

r0r12 = (xb-xa).*(xp-xa)+(yb-ya).*(yp-ya)+(zb-za).*(zp-za);
r0r22 = (xb-xa).*(xp-xb)+(yb-ya).*(yp-yb)+(zb-za).*(zp-zb);
K2 = gamma.*((r0r12./r12)-(r0r22./r22))./(4.*pi.*R1R22);

u = K2.*R1R2x2;
v = K2.*R1R2y2;
w = K2.*R1R2z2;

u(isnan(u)) = 0;
v(isnan(v)) = 0;
w(isnan(w)) = 0;
cond = (find(R1R22 < e));
u(cond) = 0;
v(cond) = 0;
w(cond) = 0;

%      velx = sum(sum(u));
%      vely = sum(sum(v));
%      velz = sum(sum(w));
%
% vel = [velx vely velz];

tx = u(:); ty = v(:); tz = w(:);
tx2 = reshape(tx,n*m,[]); ty2 = reshape(ty,n*m,[]); tz2 =
   reshape(tz,n*m,[]);
tx3 = sum(tx2'); ty3 = sum(ty2'); tz3 = sum(tz2');

vel = [tx3' ty3' tz3'];

end
```

```
function [domain,sav] = wakerollup(domain,sav)
% This function calculates the induced velocity at each point
    on the wake
% of a foil using biot-savart law. It requires a mesh field
   of equal size
% in (x,y,z) and a foil with (x,y,z) coordinates.
```

```
5
6  x = domain.x;
7  y = domain.y;
8  z = domain.z;
9  [a,b] = size(x);
10
11 for k = 1:a
12
13     for l = 1:b
14    % Point in the wake to be evaluated
15     CP = [x(k,l) y(k,l) z(k,l)];
16     % total induced velocity from the foil on the point in
          the wake
17     vel = biotsavart(CP,domain,sav);
18    % total induced velocity from the wake
19     vel2 = biotsavartwake(CP,domain,sav);
20     % Combined induced velocity
21     yushx(k,l) = vel2(1)+sum(vel(:,1));
22     yushy(k,l) = vel2(2)+sum(vel(:,2));
23     yushz(k,l) = vel2(3)+sum(vel(:,3));
24
25     end
26
27 end
28
29 domain.vel.x = yushx;
30 domain.vel.y = yushy;
31 domain.vel.z = yushz;
32
33 end
```

```
1  function domain = diplace(domain,Vel)
2
3  dt = domain.dt;
4  P = domain.P;
5  CP = domain.CP;
6
7  X = Vel(1)*dt;
8  Y = Vel(2)*dt;
9  Z = Vel(3)*dt;
10
11 vx = domain.test.vel.x;
12 vy = domain.test.vel.y;
13 vz = domain.test.vel.z;
14
15 domain.test.x2 = domain.test.x2+(vx.*dt);
16 domain.test.y2 = domain.test.y2+(vy.*dt);
17 domain.test.z2 = domain.test.z2+(vz.*dt);
```

```
18
19 domain.test.x = domain.test.x + X;
20 domain.test.y = domain.test.y + Y;
21 domain.test.z = domain.test.z;
22
23 % domain.test2.x = domain.test.x + X;
24 % domain.test2.y = domain.test.y + Y;
25 % domain.test2.z = domain.test.z;
26
27 X2 = ones(4,1)*Vel(1)*dt;
28 Y2 = ones(4,1)*Vel(2)*dt;
29 Z2 = ones(4,1)*Vel(3)*dt;
30
31 P = cellfun(@(x) [x(:,1)+X2 x(:,2) x(:,3)],P,'un',0);
32 CP = cellfun(@(x) [x(1)+X x(2) x(3)],CP,'un',0);
33
34 domain.P = P;
35 domain.CP = CP;
36
37 end
```

```
1  function domain = diplacefast(domain,Vel)
2  % This function moves the foil in time.
3  dt = domain.dt;
4  P = domain.P;
5  CP = domain.CP;
6  Vel = -Vel;
7  X= Vel(1)*dt;
8  Y = Vel(2)*dt;
9  Z = Vel(3)*dt;
10
11 % vx = domain.test.vel.x;
12 % vy = domain.test.vel.y;
13 % vz = domain.test.vel.z;
14 %
15 % domain.test.x2 = domain.test.x2+(vx.*dt);
16 % domain.test.y2 = domain.test.y2+(vy.*dt);
17 % domain.test.z2 = domain.test.z2+(vz.*dt);
18
19 domain.test.x = domain.test.x + X;
20 domain.test.y = domain.test.y + Y;
21 domain.test.z = domain.test.z;
22
23 domain.test2.x = domain.test2.x + X;
24 domain.test2.y = domain.test2.y + Y;
25 domain.test2.z = domain.test2.z;
26
27 X2 = ones(4,1)*Vel(1)*dt;
```

```
Y2 = ones(4,1)*Vel(2)*dt;
Z2 = ones(4,1)*Vel(3)*dt;

P = cellfun(@(x) [x(:,1)+X2 x(:,2) x(:,3)],P,'un',0);
CP = cellfun(@(x) [x(1)+X x(2) x(3)],CP,'un',0);

domain.P = P;
domain.CP = CP;

end
```

```
function hydro = liftmodel(foil,Vel,dt,k,hydro,amr,pos,
   simstart)

n = foil.n;
m = foil.m;
% Number of foils to be analysed
[gg,hh] = size(foil.span);

for j = 1:hh

   Vel(3) = -Vel(3);
   vel = Vel;

if k == 1

    domain.CG = pos.CG; % centre of gravity of the vessel
    domain.rot = hydro.rot; % vessel orientation
    domain.ang.phi = 90-foil.angle(j); % roll rotation of
       specific foil
    domain.ang.the = foil.angle2(j); % pitch rotation of
       specific foil
    domain.pos = amr.f(:,j); % centre of specific foil

    domain.span = foil.span(j); % Submerged span of specific
       foil
    domain.chord = foil.chord(j); % chord of specific foil
    domain.n = foil.n; % number of elements chordwise
    domain.k = k; % time step counter
    domain.m = foil.m; % number of elements spanwise
    domain.rho = 1025; % water density
    domain.NACA = cell2mat(foil.NACA(j)); % NACA profile for
       specific foil
    domain.dt = dt; % time step
    domain.vel = -vel; % Vessel velocity
    % Discretisising geometry
    domain = geomdisc(domain);
    % Saving wake grid
```

```
33      domain.test.x2 = domain.test.x;
34      domain.test.y2 = domain.test.y;
35      domain.test.z2 = domain.test.z;
36      % Initialising wake for the first time step.
37      [dd,ff] = size(domain.test.x2);
38      res.wake = ones(dd-1,ff-1).*0.5;
39      res.k = k;
40      % Unsteady Vortex Lattice Method simulator
41      [res] = UVLM(Vel,domain,res);
42
43      res.wake = res.circ(n,:);
44      % If Wake roll up is preffered, the below is also used,
           however
45      % diplacefast will also have to be used.
46
47  %      [domain,res] = wakerollupfast(domain,res);
48
49  else
50      %      domain = diplacefast(domain,Vel);
51      domain = hydro.domain(j);
52      % Updating positions, lengths and orientations
53      domain.CG = pos.CG;
54      domain.rot = hydro.rot;
55      domain.pos = amr.f(:,j);
56      domain.k = k;
57      domain.vel = -vel;
58      domain.span = foil.span(j);
59      % Discretisising geometry
60      domain = geomdisc(domain);
61
62      res = hydro.res(j);
63      res.k = k+1;
64      % Unsteady Vortex lattice Method simulator
65      [res] = UVLM(Vel,domain,res);
66      % Saving circulation of wake
67      res.wake = [res.circ(n,:); res.wake];
68      % Saving wake grid
69      domain.test.x2 = [domain.test.x(1,:); domain.test.x2];
70      domain.test.y2 = [domain.test.y(1,:); domain.test.y2];
71      domain.test.z2 = [domain.test.z(1,:); domain.test.z2];
72      % If Wake roll up is preffered, the below is also used,
           however
73      % diplacefast will also have to be used.
74  %      [domain,res] = wakerollup(domain,res);
75
76  end
77  % Removing wake after a certain period.
78  cutoff = simstart;
```

```
if k > cutoff

    res.wake(cutoff-1:end,:) = [];
    domain.test.x2(cutoff:end,:) = [];
    domain.test.y2(cutoff:end,:) = [];
    domain.test.z2(cutoff:end,:) = [];

end

    hydro.domain(j) = domain;
    hydro.res(j) = res;

clear domain res

end

end
```

```
function [domain] = geomdisc(domain)

xg = domain.CG(1);
yg = domain.CG(2);
zg = domain.CG(3);

span = domain.span;
chord = domain.chord;
NACA = domain.NACA;
n = domain.n;
m = domain.m;
domain.nn = n;
domain.mm = m;
domain.vel = domain.vel;
[XP,ZP,~] = foilpoints(NACA,n+1);
ZP = ZP.*chord;
if domain.ang.phi > 90
    ZP = -ZP;
end
domain.case = 1;
% cosine distribution of elements
for k = 1:m+1

    test(k) = span*(1-cos(k*pi/(m+2)))/2;

end

for k = 1:n+1

    test2(k) = chord*(1-cos(k*pi/(n+2)))/2;
```

```
31
32  end
33
34  B = transmatrix(domain.ang.phi,domain.ang.the,0);
35
36  wlength = 2000;
37  wlength2 = domain.vel(1)*domain.dt;
38  YS = test-span/2;
39  domain.WW{1} = [wlength2+XP(n+1) YS(1) ZP(n+1)];
40  XP = test2-domain.dt*domain.k*domain.vel(1);
41
42  xp = repmat(XP',1,m+1);
43  zp = repmat(ZP,1,m+1);
44  ys = repmat(YS,n+1,1);
45
46  x = [ones(1,m+1).*XP(n+1); ones(1,m+1).*(XP(n+1)+wlength2)];
47  y = [YS; YS];
48  z = [ones(1,m+1).*ZP(n+1); ones(1,m+1).*ZP(n+1)];
49
50  x2 = [ones(1,m+1).*XP(n+1); ones(1,m+1).*wlength];
51  y2 = [YS; YS];
52  z2 = [ones(1,m+1).*ZP(n+1); ones(1,m+1).*ZP(n+1)];
53
54  for j = 1:m
55
56      for k = 1:n
57
58          if (j == m && k == n)
59
60            P = [XP(k) YS(j) ZP(k); XP(k) YS(j+1) ZP(k);...
61            XP(k+1) YS(j+1) ZP(k+1); XP(k+1) YS(j) ZP(k+1)];
62
63            WW = [XP(k+1) YS(j) ZP(k+1); XP(k+1) YS(j+1) ZP(k
                 +1);...
64            wlength YS(j+1) ZP(k+1); wlength YS(j) ZP(k+1)];
65
66          elseif j == m
67
68            P = [XP(k) YS(j) ZP(k); XP(k) YS(j+1) ZP(k);...
69            XP(k+1) YS(j+1) ZP(k+1); XP(k+1) YS(j) ZP(k+1)];
70
71          elseif k == n
72
73            P = [XP(k) YS(j) ZP(k); XP(k) YS(j+1) ZP(k);...
74            XP(k+1) YS(j+1) ZP(k+1); XP(k+1) YS(j) ZP(k+1)];
75
76          WW = [XP(k+1) YS(j) ZP(k+1); XP(k+1) YS(j+1) ZP(k+1);
               ...
```

```
77              wlength YS(j+1) ZP(k+1); wlength YS(j) ZP(k+1)];
78
79
80          else
81
82          P = [XP(k) YS(j) ZP(k); XP(k) YS(j+1) ZP(k);...
83            XP(k+1) YS(j+1) ZP(k+1); XP(k+1) YS(j) ZP(k+1)];
84
85          end
86
87      [Px,Py,Pz] = rotgeom2(B,domain.pos,P(:,1),P(:,2),P(:,3)
           ,0,0,0);
88      P = [Px Py Pz];
89
90      PP = mean(P);
91      domain.CP{k,j} = PP;
92      domain.P{k,j} = P;
93      [domain.N{k,j},S(k,j)] = normalvec(P);
94
95      end
96      domain.W{j} = WW;
97  end
98
99  [x2,y2,z2] = rotgeom2(B,domain.pos,x2,y2,z2,0,0,0);
100
101 [x,y,z] = rotgeom2(B,domain.pos,x,y,z,0,0,0);
102
103 [xp,ys,zp] = rotgeom2(B,domain.pos,xp,ys,zp,0,0,0);
104
105 domain.wake.x = x2;
106 domain.wake.y = y2;
107 domain.wake.z = z2;
108 domain.test.x = x;
109 domain.test.y = y;
110 domain.test.z = z;
111 domain.test2.x = xp;
112 domain.test2.y = ys;
113 domain.test2.z = zp;
114 domain.S = S;
115
116 end
```

```
1  function [Force] = transforce(B,hydro,aero,wind,hull)
2
3  [dd,ff] = size(hydro.res);
4
5  for j = 1:ff
6
```

```
    force = hydro.res(j);

    F = [-abs(force.D);...
              -force.Ly;...
              abs(force.Lz)];

    % Translating forces from space to body.
    F = B*F;
    F(1) = -abs(F(1));
    if j == 1
        Ff = F;
    else
        Ff = [Ff F];
    end

end
% Driving and heeling force respectively.
Fax = aero.res.Ly*sind(wind.beta)-aero.res.D*cosd(wind.beta);
Fay = aero.res.Ly*cosd(wind.beta)+aero.res.D*sind(wind.beta);

Fs = [abs(Fax); abs(Fay); 0];
% Translating forces from space to body
Fs = B*Fs;
% Determining hull resistance
Fvx = 2*hull.area*aero.domain.rho*wind.Va^2;
Fv = [-Fvx*cosd(wind.beta) 0;...
    Fvx*sind(wind.beta) 0;...
    -(hull.mass-hull.crew)*9.81 -hull.crew*9.81];

Force.Ff = Ff;
Force.Fs = Fs;
Force.Fv = Fv;


end
```

```
function [Ig,CG] = MMI(pos)
% This function determines the mass moment of inertia matrix
   for a system
% distributed with points.

Mf = pos.foil.mass;
Mh = pos.hull.mass;
Ms = pos.sail.mass;
M = [Mf; Mh; Ms];

X = [pos.foil.cord(:,1); pos.sail.cord(:,1); pos.hull.cord
   (:,1)];
```

```
Y = [pos.foil.cord(:,2); pos.sail.cord(:,2); pos.hull.cord(:,2)];
Z = [pos.foil.cord(:,3); pos.sail.cord(:,3); pos.hull.cord(:,3)];

mt = sum(M);

Xcg =(1/mt)*sum(M.*X);
Ycg =(1/mt)*sum(M.*Y);
Zcg =(1/mt)*sum(M.*Z);

CG = [Xcg; Ycg; Zcg];
Ig = [0 0 0; 0 0 0; 0 0 0];

[a,b] = size(M);
for i =1:a
    x = (X(i) - Xcg);
    y = (Y(i) - Ycg);
    z = (Z(i) - Zcg);
    m = M(i);
 Ig = Ig + [  m*(y^2 + z^2),     -m*x*y,          -m*x*z;
             -m*y*x,              m*(x^2 + z^2), -m*y*z;
             -m*x*z,             -m*y*z,          m*(y^2 + x^2)];
end

end
```

```
function Ma = addedmass(hydro)

% This function determines the total added mass of the hydrofoil system.

[dd,ff] = size(hydro);

for j = 1:ff

    domain = hydro.domain(j);
    span = domain.span;
    chord = domain.chord;
    rho = domain.rho;

m11(j) = span*rho*pi*chord^2;
m22(j) = chord*rho*pi*span^2;
m55(j) = rho*(span^2-chord^2);

end

```

```
Ma = [  sum(m11) 0 0 0 0 0;...
        0 sum(m22) 0 0 0 0;...
        0 0 sum(m22) 0 0 0;...
        0 0 0 0 0 0;...
        0 0 0 0 sum(m55) 0;...
        0 0 0 0 0 sum(m55);];


end
```

```
function [equil,hull] = equi(Force,amr,hull,Para)
% This function determines the total force and torque of the
   vessel.

armfoil = amr.f;
armsail = amr.sail;
armhull = amr.hull;

Ff = Force.Ff;
Fs = Force.Fs;
Fv = Force.Fv;
% Determining torque components
torque = [Ff(3,:).*armfoil(2,:) Fs(3)*armsail(2) Fv(3).*
   armhull(2);...
          Ff(2,:).*armfoil(3,:) Fs(2)*armsail(3) Fv(2).*
             armhull(2);...
          -Ff(1,:).*armfoil(3,:) Fs(1)*armsail(3) Fv(1).*
             armhull(3);...
          -Ff(3,:).*armfoil(1,:) Fs(3)*armsail(1) Fv(3).*
             armhull(1);...
          -Ff(2,:).*armfoil(1,:) Fs(2)*armsail(2) Fv(2).*
             armhull(2);...
          Ff(1,:).*armfoil(2,:) Fs(1)*armsail(2) Fv(1).*
             armhull(1);];
% notice My, pitch has an hadded torque due to the crew.
equil.Mx = sum(torque(1,:))+sum(torque(2,:));
equil.My = sum(torque(3,:))+sum(torque(4,:))-Force.Fv(3,2)*
   hull.char(1)/4;
equil.Mz = sum(torque(5,:))+sum(torque(6,:));
equil.Fx = sum(Force.Ff(1,:))+Force.Fs(1)+sum(Force.Fv(1,:));
equil.Fy = sum(Force.Ff(2,:))+Force.Fs(2)+sum(Force.Fv(2,:));
equil.Fz = sum(Force.Ff(3,:))+Force.Fs(3)+sum(Force.Fv(3,:));

% Finding optimum position of the additional crew member.
A = equil.Mx;
B = equil.My;
armx = (0:-0.01:-hull.char(1)/2);
```

```
army = (0:-0.01:-(hull.char(2)+hull.char(3))/2);

if Para.phi > 0
newx = A-Force.Fv(3,2)*army/2;
[a,b] = min(abs(newx));
equil.Mx = equil.Mx-Force.Fv(3,2)*army(b)/2;
else
   b = 1;
end

newx = B-Force.Fv(3,2)*armx/2;

[d,f] = min(abs(newx));
equil.torque = torque;
equil.My = equil.My-Force.Fv(3,2)*armx(f)/2;
hull.crewpos = [army(b); armx(f)];
end
```

```
function dydt = solveEOM(equil,Ig,CG,hull,etadot)
% This function solves the equation of motion.
m = hull.mass;

xg = CG(1); yg = CG(2); zg = CG(3);

Ix = Ig(1,1); Ixy = Ig(1,2); Ixz = Ig(1,3);
Iy = Ig(2,2); Iyx = Ig(2,1); Iyz = Ig(2,3);
Iz = Ig(3,3); Izx = Ig(3,1); Izy = Ig(3,2);
% Mass matrix as described in property 3.1
M = [   m 0 0 0 m*zg -m*yg;...
        0 m 0 -m*zg 0 m*xg;...
        0 0 m m*yg -m*xg 0;...
        0 -m*zg m*yg Ix -Ixy -Ixz;...
        m*zg 0 -m*xg -Iyx Iy -Iyz;...
        -m*yg m*xg 0 -Izx -Izy Iz;];
% Centripetal matrix as described in Theorem 3.2
C = [   zeros(3,3)...
        -skew(M(1:3,1:3)*etadot(1:3)+M(1:3,4:6)*etadot(4:6));
           ...
        -skew(M(1:3,1:3)*etadot(1:3)+M(1:3,4:6)*etadot(4:6))
           ...
        -skew(M(4:6,1:3)*etadot(1:3)+M(4:6,4:6)*etadot(4:6))
           ];

u = etadot(1); v = etadot(2); w = etadot(3);
p = etadot(4); q = etadot(5); r = etadot(6);
% The 6 Equilibrium states
Tau = [equil.Fx; equil.Fy; equil.Fz; equil.Mx; equil.My;
   equil.Mz];
```

```
% Adding added mass to the mass matrix
M = M+hull.Ma;

% dydt = M\(Tau-C*etadot);
dydt = M\Tau;

end
```

```
function S = skew(X)

S = [   0 -X(3) X(2);...
        X(3) 0 -X(1);...
        -X(2) X(1) 0;];

end
```

```
function a = possprossresults,results2,wa,Vt,t)
close all
[k,l] = size(Vt);
[g,h] = size(wa);
[dd,ff] = size(results);

pwa = pi.*wa./180;
pwa = repmat(pwa,l,1)';

for j = 1:ff

    b = results(j);
    uvw = b.uvw;

    V(j) = sqrt(uvw(1)^2+uvw(2)^2+uvw(3)^2);

end


V = reshape(V',[],1);
Vfail = V(find(abs(V) > 10));
pwfail = pwa(find(abs(V) > 10));
V(find(abs(V) > 10)) = 100;
pwa(find(abs(V) > 10)) = 100;

for j = 1:l
    aa = V(:,j);
    aa(aa == 100) = [];
    bb = pwa(:,j);
    bb(bb == 100) = [];
   test{j} = [aa bb];

```

```
end
a = 1;

figure
linsp = {'-','--',':','-.','-.','--',':','-.'};
linco = {'b','r','k','m','r','k','m','b'};
for j = 1:l
    aa = cell2mat(test(j));
    polarplot(aa(:,2),aa(:,1),'linestyle',linsp{j},'Color',
       linco{j})
    hold on
end
thetalim([0 90])
axi = gca;
axi.ThetaDir = 'clockwise';
axi.ThetaZeroLocation = 'top';
thetatickformat(axi,'degrees')
axi.RAxis.Label.String = 'Vessel Speed';
axi.ThetaAxis.Label.String = 'True Wind Angle';
print('polarspeed','-depsc')
[dd,gg] = size(results2);
[bb,cc] = size(t);
for j = 1:gg
    b = results2(j);
      [dd,ff] = size(b.mx);
      mx(j,1:ff) = b.mx;
      [dd,ff] = size(b.my);
      my(j,1:ff) = b.my;
      [dd,ff] = size(b.fx);
      fx(j,1:ff) = b.fx;
      [dd,ff] = size(b.fz);
      fz(j,1:ff) = b.fz;
      [dd,ff] = size(b.v.x);
      vx(j,1:ff) = b.v.x;
      [dd,ff] = size(b.v.z);
      vz(j,1:ff) = b.v.z;
      [dd,ff] = size(b.v.p);
      vp(j,1:ff) = b.v.p;
      [dd,ff] = size(b.v.q);
      vq(j,1:ff) = b.v.q;
      [dd,ff] = size(b.Sb);
      Sb(j,1:ff) = b.Sb;
      [dd,ff] = size(b.acc.x);
      ax(j,1:ff) = b.acc.x;
      [dd,ff] = size(b.acc.y);
      ay(j,1:ff) = b.acc.y;
      [dd,ff] = size(b.acc.z);
      az(j,1:ff) = b.acc.z;
```

```
        [dd,ff] = size(b.acc.p);
        ap(j,1:ff) = b.acc.p;
        [dd,ff] = size(b.acc.q);
        aq(j,1:ff) = b.acc.q;
        [dd,ff] = size(b.acc.z);
        phi(j,1:ff) = b.ang.p;
        [dd,ff] = size(b.acc.p);
        the(j,1:ff) = b.ang.q;

end
seriies = 1;
comp = 1.1;

xlab = 'Time [s]';

ylab = 'Force [N]';
tit = 'Total Force in Surge, True Wind 6 m/s, angle 25^\circ'
    ;
miin = -max(abs(fx(seriies,:)))*comp;
maax = max(abs(fx(seriies,:)))*comp;

figure
a = plotgraph((0.1:0.1:length(fx(seriies,:))/10),fx(seriies
    ,:),xlab,ylab,tit,miin,maax);
print('forcesurge_25','-depsc')

figure
miin = 5;
maax = 8;
ylab = 'Velocity [m/s]';
tit = 'Velocity in Surge, True Wind 6 m/s, angle 25^\circ';
a = plotgraph((0.1:0.1:length(vx(seriies,:))/10),vx(seriies
    ,:),xlab,ylab,tit,miin,maax);
print('velsurge_25','-depsc')

figure
miin = -max(abs(ax(seriies,:)))*comp;
maax = max(abs(ax(seriies,:)))*comp;
ylab = 'Acceleration [m/s^2]';
tit = 'Acceleration in Surge, True Wind speed 6 m/s, angle
    25^\circ';
a = plotgraph((0.1:0.1:length(ax(seriies,:))/10),ax(seriies
    ,:),xlab,ylab,tit,miin,maax);
print('accsurge_25','-depsc')

figure
ylab = 'Force [N]';
tit = 'Total Force in Heave, True Wind 6 m/s, angle 25^\circ'
```

```
   ;
miin = -max(abs(fz(seriies,:)))*comp;
maax = max(abs(fz(seriies,:)))*comp;

a = plotgraph((0.1:0.1:length(fz(seriies,:))/10),fz(seriies
   ,:),xlab,ylab,tit,miin,maax);
print('forceheave_25','-depsc')

figure
miin = -max(abs(vz(seriies,:)))*comp;
maax = max(abs(vz(seriies,:)))*comp;
ylab = 'Velocity [m/s]';
tit = 'Velocity in Heave, True Wind 6 m/s, angle 25^\circ';
a = plotgraph((0.1:0.1:length(vz(seriies,:))/10),vz(seriies
   ,:),xlab,ylab,tit,miin,maax);
print('velheave_25','-depsc')

figure
miin = -max(abs(az(seriies,:)))*comp;
maax = max(abs(az(seriies,:)))*comp;
ylab = 'Acceleration [m/s^2]';
tit = 'Acceleration in Heave, True Wind speed 6 m/s, angle
   25^\circ';
a = plotgraph((0.1:0.1:length(az(seriies,:))/10),az(seriies
   ,:),xlab,ylab,tit,miin,maax);
print('accheave_25','-depsc')

figure
ylab = 'Torque [Nm]';
tit = 'Total Torque in Pitch, True Wind 6 m/s, angle 25^\circ
   ';
miin = -max(abs(my(seriies,:)))*comp;
maax = max(abs(my(seriies,:)))*comp;
a = plotgraph((0.1:0.1:length(my(seriies,:))/10),my(seriies
   ,:),xlab,ylab,tit,miin,maax);
print('torquepitch_25','-depsc')

figure
miin = -max(abs(vq(seriies,:)))*comp;
maax = max(abs(vq(seriies,:)))*comp;
ylab = 'Velocity [deg/s]';
tit = 'Velocity in Pitch, True Wind 6 m/s, angle 25^\circ';
a = plotgraph((0.1:0.1:length(vq(seriies,:))/10),vq(seriies
   ,:),xlab,ylab,tit,miin,maax);
print('velpitch_25','-depsc')

figure
miin = -max(abs(aq(seriies,:)))*comp;
```

```
maax = max(abs(aq(seriies,:)))*comp;
ylab = 'Acceleration [deg/s^2]';
tit = 'Acceleration in Pitch, True Wind speed 6 m/s, angle 25^\circ';
a = plotgraph((0.1:0.1:length(aq(seriies,:))/10),aq(seriies,:),xlab,ylab,tit,miin,maax);
print('accpitch_25','-depsc')

figure
miin = -max(abs(the(seriies,:)))*comp;
maax = max(abs(the(seriies,:)))*comp;
ylab = 'Angle [deg]';
tit = 'Pitch Angle, True Wind speed 6 m/s, angle 25^\circ';
a = plotgraph((0.1:0.1:length(the(seriies,:))/10),the(seriies,:),xlab,ylab,tit,miin,maax);
print('angpitch_25','-depsc')

figure
ylab = 'Torque [Nm]';
tit = 'Total Torque in Roll, True Wind angle 6 m/s, 25^\circ';
miin = -max(abs(mx(seriies,:)))*comp;
maax = max(abs(mx(seriies,:)))*comp;

a = plotgraph((0.1:0.1:length(mx(seriies,:))/10),mx(seriies,:),xlab,ylab,tit,miin,maax);
print('torqueRoll_25','-depsc')

figure
miin = -max(abs(vp(seriies,:)))*comp;
maax = max(abs(vp(seriies,:)))*comp;
ylab = 'Velocity [deg/s]';
tit = 'Velocity in Roll, True Wind 6 m/s, angle 25^\circ';
a = plotgraph((0.1:0.1:length(vp(seriies,:))/10),vp(seriies,:),xlab,ylab,tit,miin,maax);
print('velroll_25','-depsc')

figure
miin = -max(abs(ap(seriies,:)))*comp;
maax = max(abs(ap(seriies,:)))*comp;
ylab = 'Acceleration [deg/s^2]';
tit = 'Acceleration in Roll, True Wind speed 6 m/s, angle 25^\circ';
a = plotgraph((0.1:0.1:length(ap(seriies,:))/10),ap(seriies,:),xlab,ylab,tit,miin,maax);
print('accroll_25','-depsc')

figure
```

```
miin = -max(abs(phi(seriies,:)))*comp;
maax = max(abs(phi(seriies,:)))*comp;
ylab = 'Angle [deg]';
tit = 'Roll Angle, True Wind speed 6 m/s, angle 25^\circ';
a = plotgraph((0.1:0.1:length(phi(seriies,:))/10),phi(seriies
   ,:),xlab,ylab,tit,miin,maax);
print('angroll_25','-depsc')

figure
miin = 1;
maax = 3.5;
ylab = 'Area [m^2]'; tit = 'Total Submerged Area, True Wind 6
    m/s, angle 25^\circ';
a = plotgraph((0.1:0.1:length(Sb(seriies,:))/10),Sb(seriies
   ,:),xlab,ylab,tit,miin,maax);
print('subarea_25','-depsc')

%%-----------------Images--------------------%

res = 40;
[dd,ff] = size(wa);
[gg,hh] = size(t);
yt = linspace(0,ff,ff);
ytlab = linspace(min(wa),max(wa),ff);
xt = 0:max(t):hh;
xtlab = 0:max(t)/10:max(t);
ylab = 'True Wind Angle [deg]';
xlab = 'Time [s]';

figure
a = 0.6*10^4;
miin = -a;
maax = a;
barlab = 'Torque [Nm]';
tit = 'Total torque in Roll, true wind speed 6 m/s';
a = plotimage(mx,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
   xlab,barlab);
print('torqueroll_va4','-depsc')

figure
a = 5;
miin = -a;
maax = a;
barlab = 'Velocity [deg/s]';
tit = 'Velocity in Roll, true wind speed 6 m/s';
a = plotimage(vp,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
   xlab,barlab);
print('velroll_va4','-depsc')
```

```
246
247 figure
248 a = 6;
249 miin = -a;
250 maax = a;
251 barlab = 'Acceleration [deg/s^2]';
252 tit = 'Acceleration in Roll, true wind speed 6 m/s';
253 a = plotimage(ap,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
      xlab,barlab);
254 print('accroll_va4','-depsc')
255
256 figure
257 a = 7;
258 miin = -a;
259 maax = a;
260 barlab = 'Angle [deg]';
261 tit = 'Roll Angle, true wind speed 6 m/s';
262 a = plotimage(phi,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
      xlab,barlab);
263 print('angroll_va4','-depsc')
264
265 figure
266 a = 2000;
267 miin = -a;
268 maax = a;
269 barlab = 'Torque [Nm]';
270 tit = 'Total torque in Pitch, true wind speed 6 m/s';
271 a = plotimage(my,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
      xlab,barlab);
272 print('torquepitch_va4','-depsc')
273
274 figure
275 a = 0.5;
276 miin = -0.5;
277 maax = a;
278 barlab = 'Velocity [deg/s]';
279 tit = 'Velocity in Pitch, true wind speed 6 m/s';
280 a = plotimage(vq,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
      xlab,barlab);
281 print('velpitch_va4','-depsc')
282
283 figure
284 a = 0.5;
285 miin = -a;
286 maax = a;
287 barlab = 'Acceleration [deg/s^2]';
288 tit = 'Acceleration in Pitch, true wind speed 6 m/s';
289 a = plotimage(aq,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
```

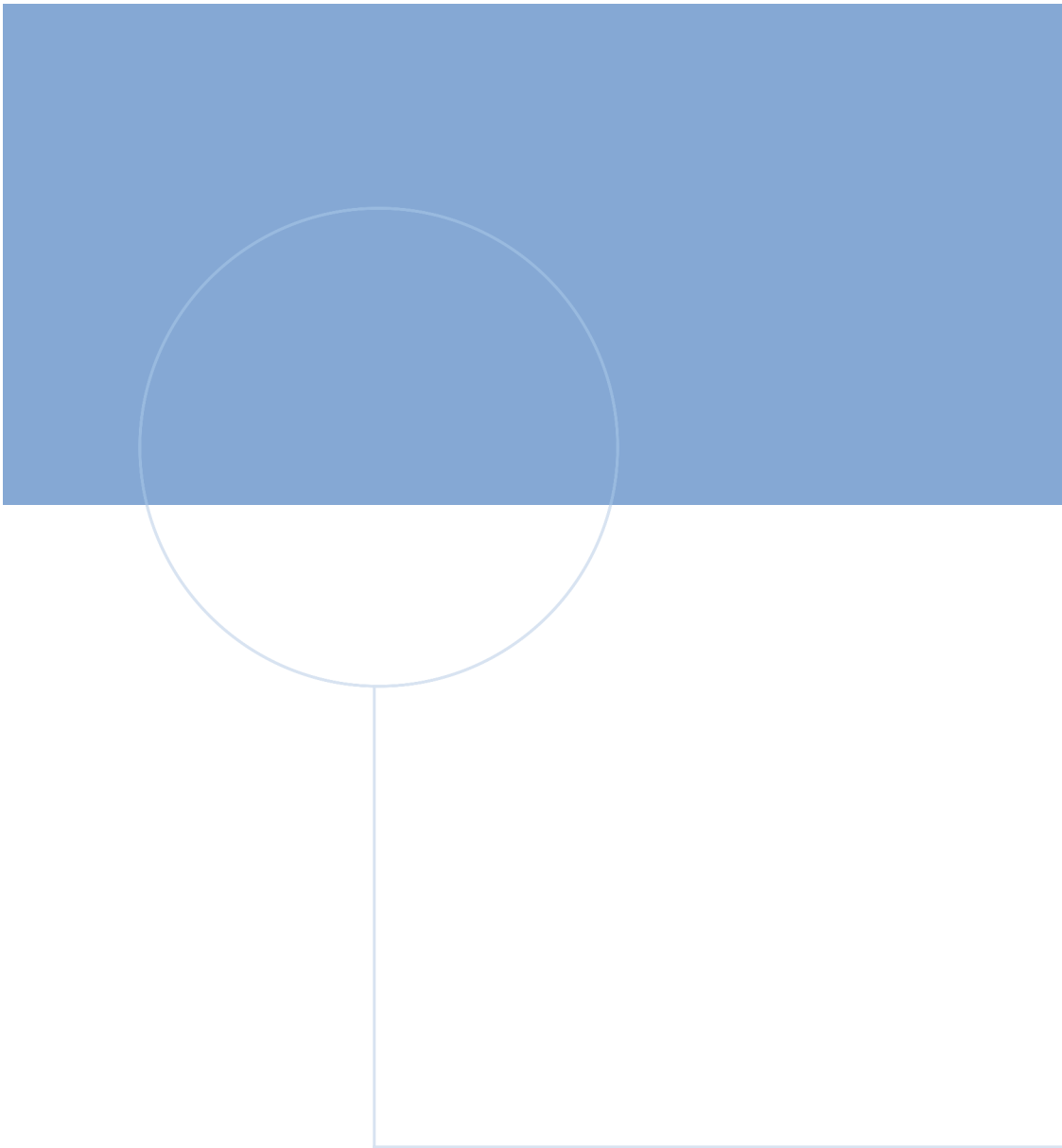
```
   xlab,barlab);
print('accpitch_va4','-depsc')

figure
a = 2;
miin = -a;
maax = a;
barlab = 'Angle [deg]';
tit = 'Pitch Angle, true wind speed 6 m/s';
a = plotimage(the,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
   xlab,barlab);
print('angpitch_va4','-depsc')

figure
a = 1000;
miin = -a;
maax = a;
barlab = 'Force [N]';
tit = 'Total Force in Surge, true wind speed 6 m/s';
a = plotimage(fx,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
   xlab,barlab);
print('forcesurge_va4','-depsc')

figure
miin = 6;
maax = 8.5;
barlab = 'Velocity [m/s]';
tit = 'Velocity in Surge, true wind speed 6 m/s';
a = plotimage(vx,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
   xlab,barlab);
print('velsurge_va4','-depsc')

figure
a = 0.5;
miin = -a;
maax = a;
barlab = 'Acceleration [m/s^2]';
tit = 'Acceleration in Surge, true wind speed 6 m/s';
a = plotimage(ax,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
   xlab,barlab);
print('accsurge_va4','-depsc')

figure
a = 2000;
miin = -a;
maax = a;
barlab = 'Force [N]';
tit = 'Total Force in Heave, true wind speed 6 m/s';
```

```
a = plotimage(fz,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
    xlab,barlab);
print('forceheave_va4','-depsc')

figure
a = 0.3;
miin = -a;
maax = a;
barlab = 'Velocity [m/s]';
tit = 'Velocity in Heave, true wind speed 6 m/s';
a = plotimage(vz,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
    xlab,barlab);
print('velheave_va4','-depsc')

figure
a = 1;
miin = -a;
maax = a;
barlab = 'Acceleration [m/s^2]';
tit = 'Acceleration in Heave, true wind speed 6 m/s';
a = plotimage(az,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
    xlab,barlab);
print('accheave_va4','-depsc')

figure
miin = 2;
maax = 3;
barlab = 'Area [m^2]';
tit = 'Total Submerged Area, true wind speed 6 m/s';
a = plotimage(Sb,tit,miin,maax,xt,xtlab,ytlab,yt,res,ylab,
    xlab,barlab);
print('subarea_va4','-depsc')

end
```

```
function a = plotgraph(x,y,xlab,ylab,tit,miin,maax)

plot(x,y)
ylim([miin maax])
xlabel(xlab)
ylabel(ylab)
title(tit)
grid on
a = 1;
end
```

```
function a = plotimage(b,d,miin,maax,xticks,Xticklabels,
    Yticklabels,yticks,res,ylab,xlab,barlab)
```

```
2
3  imagesc(b)
4  colormap(jet(res))
5  colorbar
6  cb = colorbar;
7  cb.Label.String = barlab;
8  set(gca,'Ytick',yticks, 'YTickLabel', Yticklabels)
9  set(gca,'Xtick',xticks,'XTickLabel',Xticklabels)
10 ylabel(ylab)
11 xlabel(xlab)
12 caxis([miin maax])
13 title(d)
14 a = 1;
15 end
```



NTNU

Kunnskap for en bedre verden