



**NTNU – Trondheim**  
Norwegian University of  
Science and Technology

# Hydrodynamic performance of sail-assisted merchant vessels

**Jarle Andre Kramer**

Marine Technology

Submission date: August 2014

Supervisor: Sverre Steen, IMT

Norwegian University of Science and Technology  
Department of Marine Technology





## MASTER THESIS IN MARINE TECHNOLOGY

SPRING 2014

FOR

**Jarle Andre Kramer**

### **Hydrodynamic performance of sail-assisted merchant vessels**

In order to significantly reduce the consumption of fossil fuel in deep sea shipping, sail assisted vessels are again being proposed for merchant vessels, such as bulkers and tankers. In the literature, we have seen proposal for adding a row of wing sails to for instance bulk carriers, and significant fuel savings are claimed. However, normal sailing ships have hulls specifically designed for use of sails, and important features of such ships are keels to provide the side force and yaw moment necessary to match the aerodynamic side force and yaw moment acting on the rig and superstructure. Thus, the objective of the master thesis is to determine the hydrodynamic performance of typical merchant vessels when sails are used as additional propulsion. This means to answer questions like:

- What will be typical drift angles and how large is the related added resistance?
- What will the heel angle be?

In order to reach the objective and answer questions like those just mentioned, it is recommended to: Assume that the rig will consist of a row of wing sails, and establish suitable aerodynamic characteristics for the rig.

Select initially one case vessel. If time allows additional vessels should be studied. The hydrodynamic maneuvering coefficients of the case vessel(s) can possibly be determined, for instance using ShipX. Using the models mentioned, the equilibrium condition of the case vessel should be studied for different wind speeds and directions.

If time allows, a more detailed aerodynamic model of the rig, taking interaction effects between the sails into account, should be developed.

The developed model should be used to discuss the need for improved hydrodynamic design of the hull to optimize the sailing performance, and to propose specific changes, like sharpening of bow or addition of keel(s).

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 should 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 should 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 should 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 should include a budget for the use of computer and laboratory 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 DAIM:

- Signed by the candidate
- The text defining the scope (signed by the supervisor) included
- Computer code, input files, videos and other electronic appendages can be uploaded in a zip-file in DAIM. Any electronic appendages shall be listed in the main thesis.

The candidate will receive a printed copy of the thesis.

Supervisor : Professor Sverre Steen  
Start : 14.01.2014  
Deadline : 31.08.2014

Trondheim, 14.01.2014

Sverre Steen  
Supervisor

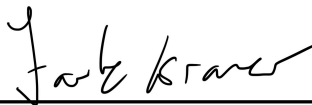
## Preface

This report is written as the final part of my master thesis, within the field of marine hydrodynamics, at the Norwegian University of Science and Technology (NTNU). This report does however not feel final. This project was specifically done in order to get a flying start on my PhD project, that will continue the day after delivery of this report. The main goal has been to get a better overview of the main topic of my PhD project, which will be about wing sail driven cargo vessels. This led me to explore not only the physics, but the modeling of wing sail physics as well. For instance, quite a lot of time has been spent on developing a Boundary Element Method (BEM) in this project, which might seem like a bit of an unnecessary amount of work, as it is only used to do rather simple experiments. The development of this code was started because I arrogantly believed that it would be a simple thing to implement, so why not make it, so that rectangular wings could be modeled properly. It ended up taking a lot of time, and it produced little results, however, I learned a *lot* in the process. Since this is hopefully only the start of a bigger project, this is not seen as waste of time, although the reader might think so while exploring the rest of the report.

Some mistakes have also been done during this project, were one important mistake happened when I was trying to model the full viscous flow around a ship hull going with a yaw angle. This is unfortunate for this project alone, but in the bigger schemes of things, this was actually very helpful. I now know a lot about stuff not to do while running CFD simulations. In the end, some results that should be reliable were produced. This was possible due to the discovery of external experimental data that allowed the results to be more trustworthy. In addition, I have had direct help from my class mate Steffen Hasfjord. He has done a master thesis were experiments were involved, so that he could do some quick tests specifically for me. This was greatly appreciated!!

I would also like to thank (soon to be Dr.) Eirik Bøckmann for many long talks about the topic of this project. He provided a lot of encouragement as he was almost as interested in this topic as I am, and even though he has had a lot of other things to do, he has helped me a lot. Off course, I would not be able to explore this topic without my supervisor Professor Sverre Steen, and my co-supervisor Dr. Luca Savio. They have provided insight and motivation through their clarifying discussions.

Trondheim 31.08.2014



---

Jarle Kramer

## Summary

The main goal of this project was to explore the different physical effects that are involved when a ship is using the technology of wing sails as propulsion. The wing sails can provide a lot of thrust, but they also create a side force. This side force will force the ship to go with a yaw angle and a heeling angle. The consequence of these angles on the ship resistance was of interest. That is, what are the negative effects to be considered when calculating the potential thrust from a wing sail?

Two ship models were selected and they were equipped with wing sails of considerable size. Realistic wind conditions were assumed. The goal was then to simulate the resulting force on these ships, from both the air and the water.

In order to do this, a custom Boundary Element Method (BEM) was developed. This code can simulate the flow around an arbitrary amount of wings, with an arbitrary shape, under the assumption of potential flow. OpenCL was used to make the code fast, while the open source 3D geometry software Blender was used for handling the geometry and post processing. The interaction effects for 8 wings in a row, under the assumption of potential flow, was explored. Some discussion about wing sails in general, along with some simple experiments involving foil shapes has also been done.

In order to model the flow around a ship hull traveling with a yaw angle, Computational Fluid Dynamics (CFD) was used, through the commercial software STAR CCM+. The author did some mistakes while using this software, but was able to at least estimate the effect of yaw on a ship. External experimental data was used to complement the data from the CFD simulations, so that reliable results could be produced in the end.

It was found that yaw has a significant effect on the resistance of a ship, while heel has almost no effect at all. The side force generated by a ship hull is non-linear in nature, which is explained by the presence of cross flow drag.

The results from the BEM code, the CFD simulations and some external experiments were coupled together to estimate the importance of yaw on a ship. It was discovered that even though a ship hull is a poor lifting surface, the resulting yaw angle for a ship under the influence of wing sails is not very large. The high density of water limits the yaw angles considerably. That is, at almost-head-wind conditions, the added resistance due to yaw is significantly reducing the effective thrust from a wing sail, but there is very little thrust at these wind conditions to begin with.

## Sammendrag

Hovedmålet med dette prosjektet var å utforske de forskjellige fysiske effektene som er involvert når et skip bruker vingeseilteknologien til å skape fremdrift. Vingeseil kan gi mye fremdriftskraft, men de skaper også en sidekraft. Denne sidekraften vil tvinge skipet til å gå med en girvinkel og en krengevinkel. Konsekvensen av disse vinkleene på skipsmotstanden var av interesse. Det vil si, hva er de negative effektene av å bruke vingeseil til å skape fremdrift?

To skipsmodeller ble valgt ut og utstyrt med vingeseil av betydelig størrelse. Realistiske vindforhold ble antatt. Målet var da å simulere de resulterende kreftene på disse skipene, fra både luft og vann.

For å gjøre dette, ble en spesialutviklet "Boundary Element Method" (BEM) utviklet. Denne koden kan simulere strømmingen rundt en vilkårlig mengde av vinger, med en vilkårlig form, under forutsetning av potensialstrømning. OpenCL ble brukt til å gjøre koden rask, mens "open source" 3D-geometriprogramvaren Blender ble brukt for å håndtere geometri og gjennomføre etterbehandling. Interaksjonseffekter for 8 vinger på rad, under forutsetning av potensialstrøm, ble utforsket. En generell diskusjon om vingeseil, sammen med noen enkle forsøk med foilgeometri har også blitt gjort.

Computational Fluid Dynamics (CFD) via en kommersiell programvare, kalt STAR CCM+, ble brukt til å modellere strømmingen rundt et skipsskrog som beveger seg med en girvinkel. Forfatteren gjorde noen feil i utførelsen av simuleringene, men var i stand til i det minste å anslå effekten av girvinkel på et skip. Eksterne eksperimentelle data ble brukt til å komplettere data fra CFD-simuleringene, slik at pålitelige resultater ble produsert til slutt.

Girvinkler har en betydelig innvirkning på motstanden til et skip, mens krengevinkler nesten ikke har noen effekt i det hele tatt. Sidekraften som genereres av et skipsskrog er ikke-lineær i karakter, noe som kan forklares ved tilstedeværelsen av tverrstrømningsmotstand.

Resultatene fra BEM koden, CFD-simuleringene og de eksterne eksperimentene ble koblet sammen for å estimere betydningen av girvinkler på et skip. Det ble oppdaget at selv om et skipsskrog er en dårlig løftende flate, så er den resulterende girvinkel for et skip under påvirkning av vingeseil ikke særlig stor. Den høye tettheten til vann begrenser størrelsen på girvinkelen betraktelig. Unntaket er ved nesten motvind. Her vil den resulterende motstanden på grunn av girvinkel gi betydelig reduksjon i den effektive fremdriftskraften, men siden det er lite fremdriftskraft i utgangspunktet, er ikke det av så stor betydning.

# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	General Overview of the Test Cases Used in this Project	11
1.1.1	Ship Models	11
1.1.2	Wing Sail Size Limitations	12
1.1.3	Wind	14
<b>2</b>	<b>Literature Review</b>	<b>17</b>
2.1	Projects that Investigates Wing Sails	18
2.1.1	Walker wing sails	18
2.1.2	B9 Shipping, Modern Clipper	18
2.1.3	UTC Wind challenger	18
2.2	Forces due to Heel and Yaw based on Yacht literature	19
2.3	Conclusion from literature review	21
<b>3</b>	<b>Boundary Element Method for 3D Lifting Surfaces</b>	<b>22</b>
3.1	Mathematical Implementation	24
3.1.1	Solid Wall Boundary Conditions and Singularity Mix	26
3.1.2	Potential wake	27
3.1.3	Evaluation of Panel Integrals	29
3.1.4	Ship Deck Modeling	32
3.1.5	Analysis of Velocity and Pressure	32
3.1.6	Force Calculation	33
3.2	Numerical Implementation	33
3.2.1	Overview of the Program	34
3.2.2	OpenCL	35
3.2.3	Blender	36
3.3	Verification of BEM code	37
3.3.1	Near-Infinite Cylinder	37
3.3.2	Near-Infinite Wing	39
3.3.3	Validation of Lift by Comparison with Experimental Data	40
3.3.4	Validation of Lift-Induced Drag	41
<b>4</b>	<b>CFD using STAR CCM+</b>	<b>42</b>
4.1	Theory	43
4.2	Parameters Used	45
4.2.1	CFD Ship Model	45
4.2.2	Domain Size	45
4.2.3	Mesh	47
4.2.4	Time stepping	50
4.3	Results from CFD	53
4.4	Comparison With Experiments	54
4.5	Discussion Regarding the Accuracy of the Results	55
<b>5</b>	<b>Other Methods</b>	<b>57</b>
5.1	External Experiments	57
5.1.1	EFD Data from the University of Iowa	57



5.1.2	Cargo Ship Heel Experiments . . . . .	59
5.1.3	Geitbåt Experiments . . . . .	60
5.2	Viscous Modeling of Wing Profiles using XFOIL . . . . .	60
5.3	Stability Calculations . . . . .	62
5.3.1	Mathematical Theory of Stability . . . . .	62
5.3.2	Ship stability data . . . . .	63
5.4	Going from Model Scale to Full Scale . . . . .	63
5.5	Final Coupled Model . . . . .	64
5.5.1	Effective Thrust Prediction . . . . .	64
5.5.2	Velocity Prediction . . . . .	65
<b>6</b>	<b>The Physics of Wing Sails and How to Model it</b>	<b>66</b>
6.1	Forces from a wing sail . . . . .	67
6.1.1	Lift . . . . .	67
6.1.2	Drag . . . . .	68
6.1.3	Thrust and Side Force . . . . .	69
6.2	Wing Sails vs Conventional Wings . . . . .	70
6.3	Foil Profile . . . . .	72
6.3.1	4-Digit NACA Profile with Varying Thickness . . . . .	72
6.3.2	Other Foil Profiles . . . . .	74
6.4	3D Effects . . . . .	75
6.4.1	Rectangular Wing . . . . .	76
6.4.2	Interaction Between Wing Sails in a Row . . . . .	77
6.5	Complete Wing Sail Model . . . . .	80
<b>7</b>	<b>Hydrodynamic Effects</b>	<b>81</b>
7.1	Calm Water Resistance without Yaw . . . . .	81
7.2	Effect of Heel . . . . .	82
7.3	Effect of Yaw . . . . .	82
<b>8</b>	<b>Results</b>	<b>85</b>
8.1	Forces From Wing Sails, including Interaction Effects . . . . .	85
8.2	Using Wing Sails as Auxilarary Propulsion . . . . .	86
8.2.1	Chemical Tanker . . . . .	87
8.2.2	Series 60 . . . . .	88
8.3	Using Wing Sails as the Only Propulsion . . . . .	89
8.4	Comparing Hulls to Wing . . . . .	90
<b>9</b>	<b>Final Discussion and Conclusion</b>	<b>91</b>
<b>10</b>	<b>Further Work</b>	<b>94</b>
<b>11</b>	<b>Appendix</b>	<b>98</b>
11.1	BEM code . . . . .	98
11.1.1	Example control script . . . . .	98
11.1.2	Geometry.py . . . . .	99
11.1.3	Computation.py . . . . .	104
11.1.4	Kernels.cl . . . . .	110
11.1.5	Example Control Script . . . . .	117
11.1.6	Convergence Analysis . . . . .	117
11.2	Final Results - Matlab scripts . . . . .	120
11.2.1	wingCoefficientsChemicalTanker.m . . . . .	120
11.2.2	wingCoefficientsSeries60.m . . . . .	121
11.2.3	finalResultsChemicalTanker.m . . . . .	123
11.2.4	finalResultsSeries60.m . . . . .	124
11.2.5	SailsOnlySeries60.m . . . . .	126

# List of Figures

1.1	Artistic illustration of a ship with wing sails . . . . .	9
1.2	Line drawings of chemical tanker . . . . .	12
1.3	Histogram of "clearance below bridge" for bridges in Norway that are at least 400 m long . .	12
1.4	Illustration showing the two ship hulls next to each other, with wing sails on deck and correct dimensions. The large red hull is the chemical tanker, the small yellow hull is Series 60 . . .	14
1.5	Annual average wind speed at 80 m above sea level, Norway. Source: [41] . . . . .	15
2.1	An illustratuin of the proposed ship design from B9 Shipping . . . . .	18
2.2	An illustratuin of the UTC wind challenger, at sea. Source: [30] . . . . .	19
2.3	Different types of resistance taken from "Fluid Mechanics For Sailing Vessel Design", source: [26] . . . . .	20
2.4	Different types of resistance taken from "Principles of Yacht Design", source: [22] . . . . .	20
3.1	Illustration of model used in the custom BEM code, with calculated wake shape, and color mapped pressure values on the wing . . . . .	23
3.2	Illustration of fluid domain, used as a reference for developing the theory for the BEM code .	25
3.3	The wake and wing divided into strips, where each separate color represents a separate strip	28
3.4	Illustration of some integration points in the wake (conceptual illustration) . . . . .	29
3.5	Nomenclature used while expressing the result of the integral equations for a flat polygon shaped panel . . . . .	30
3.6	Time to build an equation system, done with different calculation methods . . . . .	35
3.7	The interface used when running BEM simulations in Blender . . . . .	37
3.8	Geometry of the cylinder used in the verification test . . . . .	38
3.9	Pressure coefficient for a near-infinite cylinder. Theoretical and numerical values . . . . .	39
3.10	Pressure coefficient for a near-infinite wing with foil profile NASA LS - 0417 . . . . .	39
3.11	Pressure coefficient for a near-infinite wing with foil profile NACA0014 at 10 deg angle of attack	40
3.12	Lift coefficient for rectangular wings with different Asp. Data from BEM and source [31] . . .	41
3.13	Induced drag coefficient for rectangular wing with Asp = 6, compared with theoretical values from source [34] . . . . .	41
4.1	Illustration of domain dimensions . . . . .	46
4.2	The mesh used to simulate the chemical tanker at 6 deg yaw angle, viewed from the top . .	48
4.3	The mesh used to simulate the chemical tanker at 6 deg yaw angle, viewed from the side . .	48
4.4	The mesh used to simulate the chemical tanker at 6 deg yaw angle, viewed from the front .	49
4.5	Drag force value as a function of time step size for the chemical tanker. Performed with base size = 0.1, and model scale speed = 1.42 m/s . . . . .	51
4.6	Wave pattern for the chemical tanker with time step = 0.01 s . . . . .	52
4.7	Wave pattern for the chemical tanker with time step = 0.1 s . . . . .	52
4.8	Resulting forces from CFD experiment at $U_m = 0.95$ m/s which corresponds to $U_S = 10$ knots	53
4.9	Resulting forces from CFD experiment at $U_m = 1.18$ m/s which corresponds to $U_S = 12.5$ knots . . . . .	53
4.10	Resulting forces from CFD experiment at $U_m = 1.42$ m/s which corresponds to $U_S = 15$ knots	54
4.11	Validation test of drag force on Series 60, for different yaw angles, compared to EFD data .	55
4.12	Validation test of lift force on Series 60, for different yaw angles, compared to EFD data . . .	55
5.1	Drag coefficient, $C_T$ , for Series 60 based on experimental results, for different yaw angles .	58

5.2	Side force coefficient, $C_S$ , for Series 60 based on experimental results, for different yaw angles	59
5.3	Viscous drag predicted by XFOIL compared to empiric shape factor for foils, and different friction lines. Performed with NACA 0012 as geomtry	61
6.1	Nomenaclure used to describe wing sails	66
6.2	Illustration of the relationship between the different forces acting on a wing sail	69
6.3	Relationship between wind, ship velocity and apparent wind	70
6.4	Example of airplane wing profiles. source: [42]	71
6.5	Optimal angle of attack for a wing sail, calculated based on elliptic wing theory, using viscous data from XFOIL and different aspect ratios	71
6.6	Thinnest and thickest NACA 4-digit foils used	73
6.7	Lift and drag coefficients for different NACA 4-digit foil profiles	73
6.8	Thrust coefficient for different NACA 4-digit foil profiles	74
6.9	Non-NACA foils tested	74
6.10	Lift and drag coefficients for different foil profiles	75
6.11	Thrust coefficient for different foil profiles	75
6.12	Illustration of the model used in the BEM simulation for single rectangular wing with physical aspect ratio = 5. The wake is deformed according to the final solution, and the colors on the wing corresponds to pressure values	76
6.13	Lift and drag data for single wing, calculated with different methods	77
6.14	Illustration of the simulation model used in the BEM code in order to estimate interaction effects between wing sails standing in a row. The ship hull is included in the illustration, but was not directly a part of the simulation	78
6.15	Interaction effects on lift for 8 sails in a row	79
6.16	Interaction effects on drag for 8 sails in a row	79
7.1	Full scale calm water resistance, at 0 deg yaw, for the two test ships, as a function of ship speed in knots	81
7.2	Resistance due to yaw, as a function of yaw angle	83
7.3	Side force due to yaw, as a function of yaw angle	84
8.1	Maximum thrust coefficient for the wing sails for the different ships, calculated with different methods. The chemical tanker wings have physical Asp = 2.67 while the Series 60 wings have physical Asp = 5	86
8.2	Side force coefficient for the wing sails for the different ships, calculated with different methods. The chemical tanker wings have physical Asp = 2.67 while the Series 60 wings have physical Asp = 5	86
8.3	Effective thrust from wing sails on the chemical tanker, with and without hydrodynamic effects from yaw. $U_s$ is the ship speed, and the wind speed is set to be 7 m/s	87
8.4	Required yaw angle of the chemical tanker, in order to balance the side forces from the wing sails. $U_s$ is the ship speed, and the wind speed is set to be 7 m/s	87
8.5	Maximum value of KG for the chemical tanker in order to withstand the heeling moment generated by the sails, at 10 deg heeling angle. $U_s$ is the ship speed, and the wind speed is set to be 7 m/s	88
8.6	Effective thrust from wing sails on Series 60, with and without hydrodynamic effects from yaw. $U_s$ is the ship speed, and the wind speed is set to be 7 m/s	88
8.7	Required yaw angle of Series 60, in order to balance the side forces from the wing sails. $U_s$ is the ship speed, and the wind speed is set to be 7 m/s	89
8.8	Maximum value of KG for Series 60 in order to withstand the heeling moment generated by the sails, at 10 deg heeling angle. $U_s$ is the ship speed, and the wind speed is set to be 7 m/s	89
8.9	Maximum ship speed, while using wing sails as the only form of propulsion, for different wind speeds. Based on Series 60 data	90
8.10	Lift force divided by drag force, for the two test hulls, and a theoretical elliptic wing, with viscous effects modeled by XFOIL	90
11.1	Iteration test to see the influence of wake shape on lift coefficient	118
11.2	Lift and drag coefficient as a function of number of panels pr strip	118
11.3	Lift and drag coefficient as a function of number of strips	119
11.4	Lift and drag coefficient as a function of wake length	120

# List of Tables

1.1	Ship models used in this project . . . . .	11
1.2	Wing sail numbers . . . . .	13
2.1	Typical leeway angles for different ship types, based on reference [3] . . . . .	21
4.1	Data for the CFD ship models, at model scale . . . . .	45
4.2	Simulation domain length scales, reference values and values used in this project. All values are divided by the ship length used in the corresponding simulation . . . . .	46
4.3	Local refinement for special areas in the simulation domain . . . . .	47
4.4	Results from mesh convergence test with yaw angle equal to 8 degrees . . . . .	50
4.5	Results from mesh convergence test without yaw angle . . . . .	50
5.1	Model scale data, Series 60 . . . . .	58
5.2	Ship model data for Rolls-Royce chemical tanker, used to find the importance of heel for a cargo ship . . . . .	59
5.3	Result of heel experiments, done by Steffen Hasfjord, performed with Rolls-Royce chemical tanker . . . . .	60
5.4	Full scale resistance [N] for "Geitbåt", for different speeds and heeling angles . . . . .	60
5.5	Full scale resistance [N] for "Møringbåt", for different speeds and heeling angles . . . . .	60
5.6	Stability data for the ship models used in this project . . . . .	63
6.1	Non-geometry variables used in the foil experiment . . . . .	72

# Nomenclature

$B_{wl}$  Waterline breadth of ship. 11, 45, 58, 59

$C_{Dv}$  Viscous drag coefficient. 61

$C_{L,2D}$  Lift coefficient for a 2D wing profile. 67

$C_L$  Lift coefficient. 67

$C_R$  Residual drag coefficient for ship. 64

$C_S$  Side force coefficient for ship. 54, 58, 64

$C_T$  Total drag coefficient for ship. 54, 58

$C_x$  Thrust coefficient for wing sail. 69

$C_y$  Side force coefficient for wing sail. 69

$D$  Draft of ship. 11, 45, 58, 59, 63

$I_{yy}$  Second moment of area for the y-axis for the water line plane of a ship. 63

$L_{oa}$  Over all length of ship. 11

$L_{pp}$  Length between perpendiculars of ship. 11, 59

$L_{wl}$  Waterline length of ship. 11, 45, 58, 59

$S$  Wetted surface of ship. 11, 45, 58, 59

$U_S$  Full scale/ship speed. 3, 53, 54

$U_m$  Model scale speed. 3, 45, 53, 54

$\Delta$  Weight displacement of ship. 11

$\nabla$  Volume displacement of ship. 11, 45, 58, 59, 63

$\nu$  Kinematic viscosity. 45

$\rho$  Density. 45

**apparent wind** The vector sum of the incoming velocity due to the wind and the incoming velocity due to the ships movement. 64--67, 69, 71, 72, 80

**Asp** Aspect ratio of wing. 3, 13, 41, 68

**Blender** Open source 3D modeling software. Used to create and manage 3D geometry to be used in simulations in this project. 10, 23, 24, 33, 34, 36, 37, 93

**BM** Distance from the center of buoyancy to the meta center for a ship. 63

**GM** Distance from the center of gravity to the metacenter of a ship. 62

**GZ** The restoring moment-arm. 62

**KB** Distance from the keel to center of buoyancy for a ship. 63

**KG** Distance from the keel to the center of gravity for a ship. 62, 64, 65, 85, 86

**Matlab** Numerical computing environment, developed by the company MathWorks. 64, 71, 80

**Re** Reynolds number. 45

**STAR CCM+** Commercial CFD software, developed by the company CD-Adapco. 10, 42--45, 47, 53, 56

**XFOIL** Open source 2D panel code, with viscous integral boundary layer formulation for viscous effects. Capable of finding 2D drag and lift coefficients for foil profiles. 10, 37, 39, 57, 60, 61, 67, 68, 71, 72, 76, 80, 85

# Acronyms

**API** Application Programming Interface. 34

**BEM** Boundary Element Method. 10, 17, 22--24, 26, 30, 32--38, 41, 56, 67, 68, 76, 78, 79, 91, 92, 94

**CFD** Computational Fluid Dynamics. 10, 11, 17, 19, 42, 43, 45, 50, 56, 57, 64, 77, 82, 83, 91, 92, 94

**CPU** Central Processing Unit. 10, 23, 35, 36

**DWT** Deadweight tonnage. 18, 59

**EFD Data** Experimental Fluid Dynamics Data. 10, 11, 42, 43, 45, 54--58, 64, 82, 83, 91

**FDM** Finite Different Method. 22

**FEM** Finite Element Method. 22

**FVM** Finite Volume Method. 22, 43

**GPU** Graphics Processing Unit. 10, 23, 35, 36

**MARINTEK** The Norwegian Marine Technology Research Institute. 59, 60, 63, 82

**NTNU** Norwegian University of Science and Technology. 60

**NumPy** Numerical Python. 34

**NURBS** Non-uniform rational B-spline. 36, 37

**OpenCL** Open Computing Language. 10, 23, 24, 33--36, 92

**RANS** Reynolds-Averaged Navier-Stokes. 44, 45

**SIMPLE** Semi-Implicit Method for Pressure Linked Equations. 43

**VOF** Volume Of Fluid. 44, 47

**VPP** Velocity Prediction Program. 60

# Chapter 1

## Introduction

This main goal of this project is to figure out the importance of different physical effects when evaluating the benefits of using wing sails as auxiliary propulsion. Wing sails are a technology that allow a ship to extract energy from the wind. They work very much like normal sails, except from the fact that they are rigid structures rather than soft cloth. They work like normal wings, except that for the fact that the lift they generate can push something forward, rather than keep something up. Wing sails are an interesting technology, because they can provide a supplement to fossil fuel based engines. Using wing sails could be a good way to reduce fuel cost and CO<sub>2</sub> emissions for ships.

An important challenge with this type of propulsion is that the wings will generate a side force and a heeling moment on the hull. It is impossible to have thrust from a wing sail, without side force and heeling moment, unless the wind is coming from directly behind the ship. The side force and heeling moment must be balanced by the hull. The question is, will the act of balancing the unwanted effects of wing sails have a large negative effect on the resistance of the ship?

The ship type that will be analyzed is cargo ships. Specifically, a medium sized tank ship and the container ship model known as *Series 60* will be used as a representative hulls in the analysis. An artistic illustration of how a cargo ship with wing sails could look is shown in figure 1.1



Figure 1.1: Artistic illustration of a ship with wing sails



The wing sails itself will be modeled using a custom Boundary Element Method (BEM) code developed specifically for this project. This code can quantify the lift, lift induced drag, and some interaction effects between sails standing in a row. A big part of this project, in terms of time, has been devoted to the development of this code. Some novel features has been included, such as coupling with an open source 3D modeling software called Blender, and a highly flexible parallel code, that can be executed on any computational device in a modern computer, using Open Computing Language (OpenCL). More specifically, using OpenCL allows the code to be executed in parallel on both multicore Central Processing Units (CPUs) and Graphics Processing Units (GPUs)

Viscous effects for the wing sails are treated with integral boundary layer equations, through the open source software XFOIL. There is some discussion about the physics of wing sails in general along with some preliminary investigation into design options for such sails.

The hydrodynamics effects on the hull, under the influence of a wing sail, is investigated using a combination of external Experimental Fluid Dynamics Data (EFD Data) along with Computational Fluid Dynamics (CFD), using the commercial software STAR CCM+. In particular, the effect of letting the hull go with a yaw angle, and a heel angle is investigated.

With data from several sources, a complete model is built, that aims to quantify the importance of hydrodynamic effects on the hull when using wing sails. The goal is to determine whether these types of effects are important when designing a hull to be used with wing sails. For instance, will the fact that the ship must go with a yaw angle in order to balance the side force from the sail create a large added resistance?

In addition, some investigation is done regarding interaction effects between wing sails standing in a row. This is done with the BEM code, so viscosity is not modeled, and the interaction effects are therefore not completely captured. However, the results can be considered as an indication of whether or not interaction between wing sails are an important effect to model when considering wing sails as propulsion. They are also interesting in terms of explaining the physics involved.

This project does not try to quantify how much fuel one can save by using wing sail technology. Partly because this is left for later work, but also because this has, at least to a certain degree, been done before. Some previous studies regarding wing sails, that investigate this question can be found under the literature review. When that is said, many of the final results graphs are showing thrust from wing sails, related to the resistance of the ship, so realistic wind speeds and sails dimensions are assumed.

This project aims to be a pre-study before the author is embarking on a bigger PhD project concerning the design of wing sail driven cargo ships. The results from this master thesis is expected to give some guidance on how to proceed in the future. This is also the reason why several types of methods has been used in the project (BEM, CFD, some experiments). The author was interested in testing some different approaches to model the physics of a wing sail driven cargo ships.

The BEM code was of interest as this type of simulation can quantify many of the important physical effects that influences the performance of wing sails, but since it is based on a simplified model of the physics involved (neglecting viscosity) it can do it much faster than general CFD methods. Short simulation time can be a benefit in many cases, but perhaps particularly when one wishes to couple the flow simulation with other types simulations/calculations. The author had some experience with BEM from before, but then only in 2D. A 3D BEM code is significantly different in many ways, specifically because it needs to model the potential wake and 3D geometry is more complex than 2D geometry.

The CFD was of interest as this is the only way to model the *complete* Navier-Stokes equations, and therefore the method that makes the least amount of physical simplifications. It is known to be possible to model the flow around a ship using CFD, but much of the purpose of using CFD in this project has been to see if it is a practical way to do it. That is, can CFD give reasonable result without taking to much time or resources? The author had some experience with CFD from before this project, but this is the first time the author has used it for flow around ship hulls, i.e. flow with a free surface. Another important reason for using CFD was that simpler methods for estimating ship resistance can not modeled the added resistance due to yaw properly.

Many of the results in this report will be shown relative to a simpler model that is commonly assumed when one are discussing wing sails as a propulsion alternative. That is, a ship where the resistance is evaluated without taking yaw and heel into considerations, and the wings are modeled using theoretical elliptical wings, without any interaction effects. Considering the difference between this simple model, and a more complex model with "more physics" involved is used as a way to quantify the importance of the physical effect in question.

## 1.1 General Overview of the Test Cases Used in this Project

---

This section gives an overview of the general *environment* that is assumed in this project. That is, the ship hull geometry, the limitations on the design of the wing sail, and the wind conditions assumed. These things are important, as they influence many of the details presented later in this report. The goal is to assume realistic boundary conditions, that are representative of how a real ship with wing sails could be designed. At the same time, the general parameters are evaluated using relative simple considerations, as there are many practical aspects that this project does not have time to consider.

### 1.1.1 Ship Models

What type of ship to use was decided based on two factors. For one, it is assumed that "slow moving" cargo is a more realistic market for wing sails than "fast moving cargo". This is due to limitation in the available wind speed, and due to the fact that the resistance of a ship quickly increases with increasing velocity. Based on this, it was assumed that a tank ship could be a good test case. A medium sized tank ship model was therefore wanted. The geometry of the ship must also be available, in a format that is understandable. The ship geometry software "FreeShip" has several built in test ships, where one of them happens to be a chemical tank ship. FreeShip can export the geometry into many common geometry formats, so this ship was decided to be a good candidate. The details of the ship can be seen in table 6.1

While working on this project, some experimental data was wanted in order to compare CFD data. A perfect set of EFD Data was found for the container ship "Series 60". EFD Data data was available for several speeds and several yaw angles. This data is therefore used, both for comparing CFD results, and used directly in the final modeling. The detail of the ship can be seen in table 6.1

Table 1.1: Ship models used in this project

	FreeShip Chemical Tanker	Series 60 Container Ship
$L_{oa}$ [m]	174	122
$L_{pp}$ [m]	170	122
$L_{wl}$ [m]	170	122
$B_{wl}$ [m]	29.5	16.3
$D$ [m]	10	6.5
$\nabla$ [m <sup>3</sup> ]	38147	7744
$\Delta$ [tonnes]	39135	7938
$S$ [m <sup>2</sup> ]	6862	2526.4

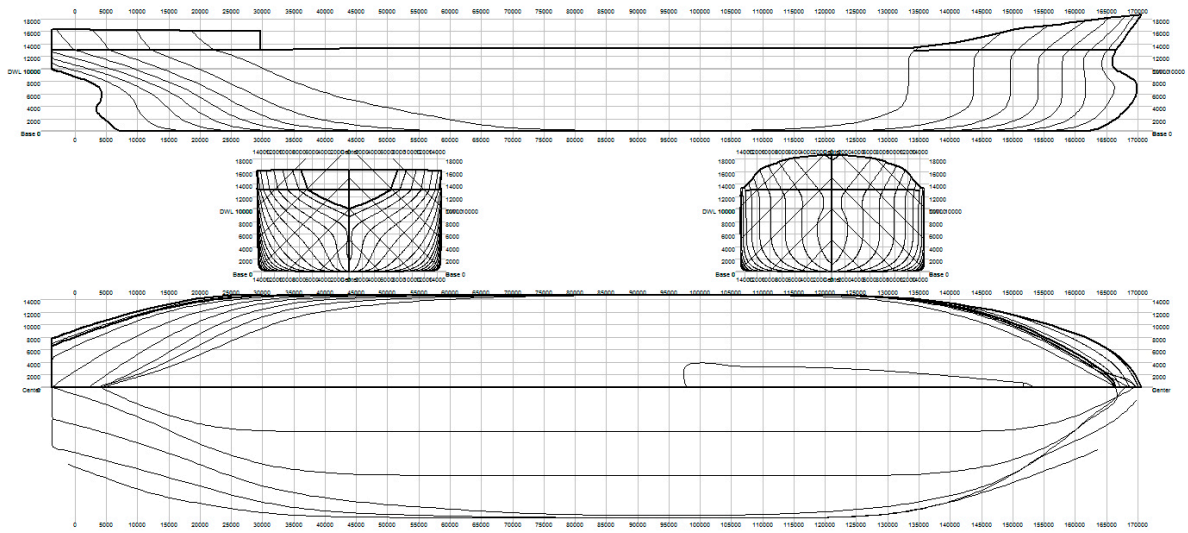


Figure 1.2: Line drawings of chemical tanker

### 1.1.2 Wing Sail Size Limitations

Realistic dimensions of the wing sails are important as the forces from the wing sail are proportional to its size. Many practical aspects could have influence on the design of wing sail. For instance, cargo off- and on-loading are often done with cranes that operates directly above the ship. This could set severe restrictions to where the wing sails could be located, and maybe even the height. This problem is not evaluated in this project. Another big limitations, in terms of height of the wing sails, is bridges. In order to get an overview of "normal" clearance below a bridge, data for 70 bridges in Norway that are at least 400 m long was collected from reference [45]. This data can be viewed as a histogram, with distribution fitting on top, in figure 1.3

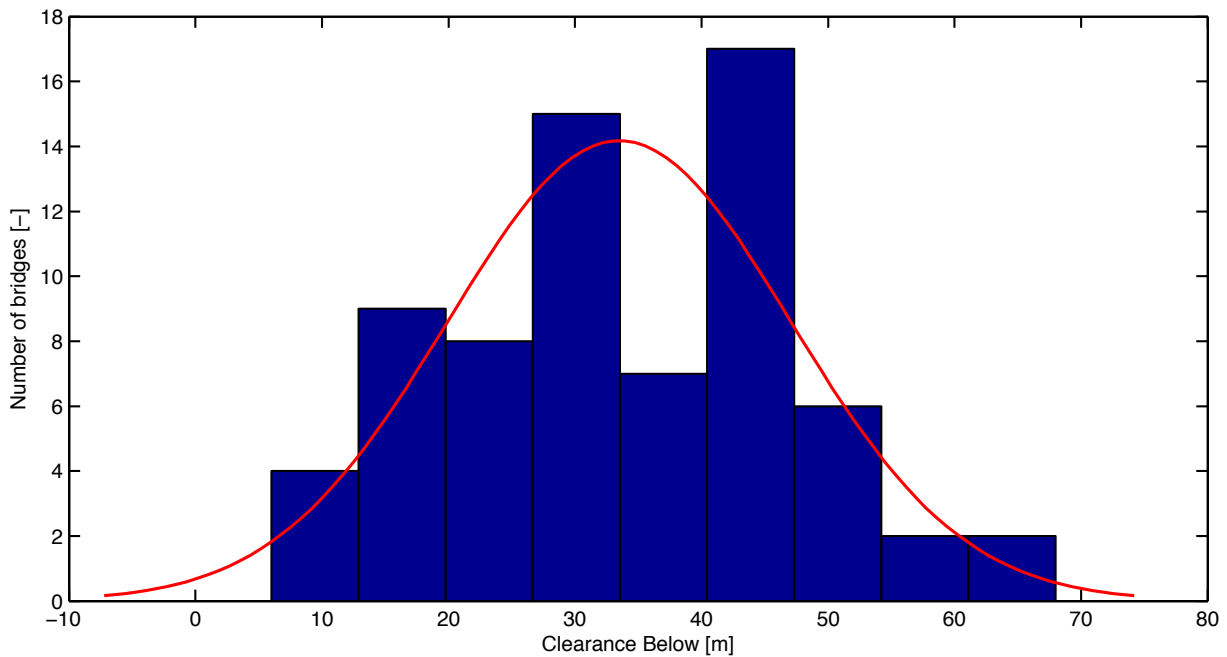


Figure 1.3: Histogram of "clearance below bridge" for bridges in Norway that are at least 400 m long

The mean value of all the bridges are 33.5 m. Not all of the bridges will be a problem. Many bridges are

located in areas without shipping traffic. This is evident if one considers the fact that there exist quite a few bridges with less than 10 m clearance. Many normal ships are significantly taller than this. The cruise ship "Hurtigruta MS Trollfjord", for instance, has a height of 29.9 m, which can cause problems with Norwegian bridges when there is a very high tide [28]. On the other hand, very few bridges are taller than 50 m.

Based on this, the height of the wing sail is set to be, at maximum, 40 m. This gives a total height that is a bit more than 40 m (as the lower end of the sail starts a bit above water).

40 m is also the height of the wing sails used on the Americas Cup Catamarans AC-72 [44], which at the time of writing this project perhaps is the most famous wing sail driven vessels.

In terms of chord length of each sail, and number of sails, the restrictions are set according to ship hull geometry. The wing sail can not extend outside the ship in either direction. That is, the maximum chord length can not be larger than the width of the ship. In order to be a bit more limiting, as there are plenty of practical considerations that this simple analysis does not account for, 50% of the ship width is set to be the maximum chord length. This gives a chord length of maximum 15 m for the Chemical tanker and 8 m for the Series 60 Container ship.

The number of sails should probably be as high as possible. More sails, more thrust. But the sum of the chord lengths can not exceed the total ship length. That is, for the chemical tanker, the absolute maximum number of sails is  $170/15 \approx 11$ , while for the the Series 60 container ship, the number is  $122/8 \approx 15$ . That many sails would off course cover the entire ship length in sails, which is probably not a realistic case. There needs to be some space between the sails, and it needs to be room for the bridge and other superstructures on the ship. In keeping with the simple considerations done so far, the number of sails for both ships is set to be 8. This leaves 50 m of sail-free ship length for the chemical tanker, and 58 m for the Series 60 ship. If the sails are divided evenly along the ship length, the average sail-free space between each sail will be approximately 7 m for the chemical tanker and 7.25 for the series 60 ship.

The summary of the parameters that were decided in this simple analysis is shown in table 1.2

Table 1.2: Wing sail numbers

	FreeShip Chemical Tanker	Series 60 Container Ship
Height [m]	40	40
Chord length [m]	15	8
Area of each sail [m <sup>2</sup> ]	600	320
Physical Asp, rectangular wing [-]	2.67	5
Number of sails [-]	8	8
Total sail area [m <sup>2</sup> ]	4800	2560

In order to get an overview of what these dimensions mean, relative to the ship hulls, a simple 3D model of both ships, with wing sails, and correct dimensions was built. This can be seen in figure 1.4

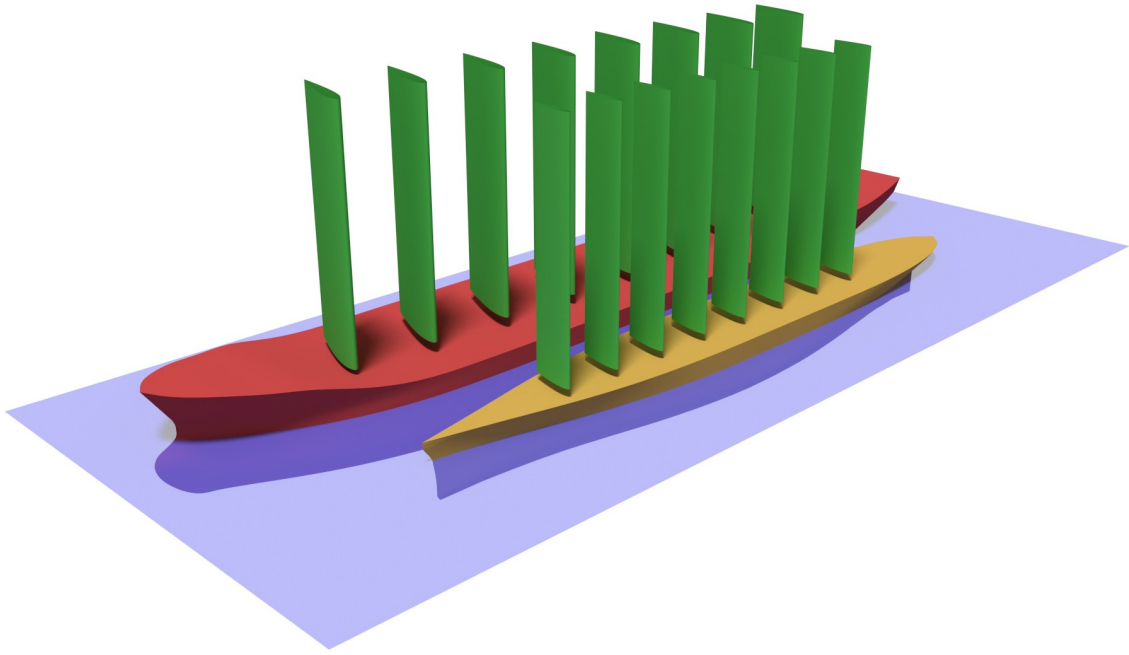


Figure 1.4: Illustration showing the two ship hulls next to each other, with wing sails on deck and correct dimensions. The large red hull is the chemical tanker, the small yellow hull is Series 60

### 1.1.3 Wind

A realistic value for the wind speed is important in order to get an overview of realistic forces on the sail. As mentioned before, it is not the purpose of this project to figure out the potential energy savings from wing sails, however, it is important that the forces from the sail is at an order of magnitude that is realistic.

The wind speed varies greatly across the globe, and is also very dependent on the season. For this project, the oceans around Norway is used, with annual average values. The data is collected from reference [41], and can also be vied in figure 1.5.

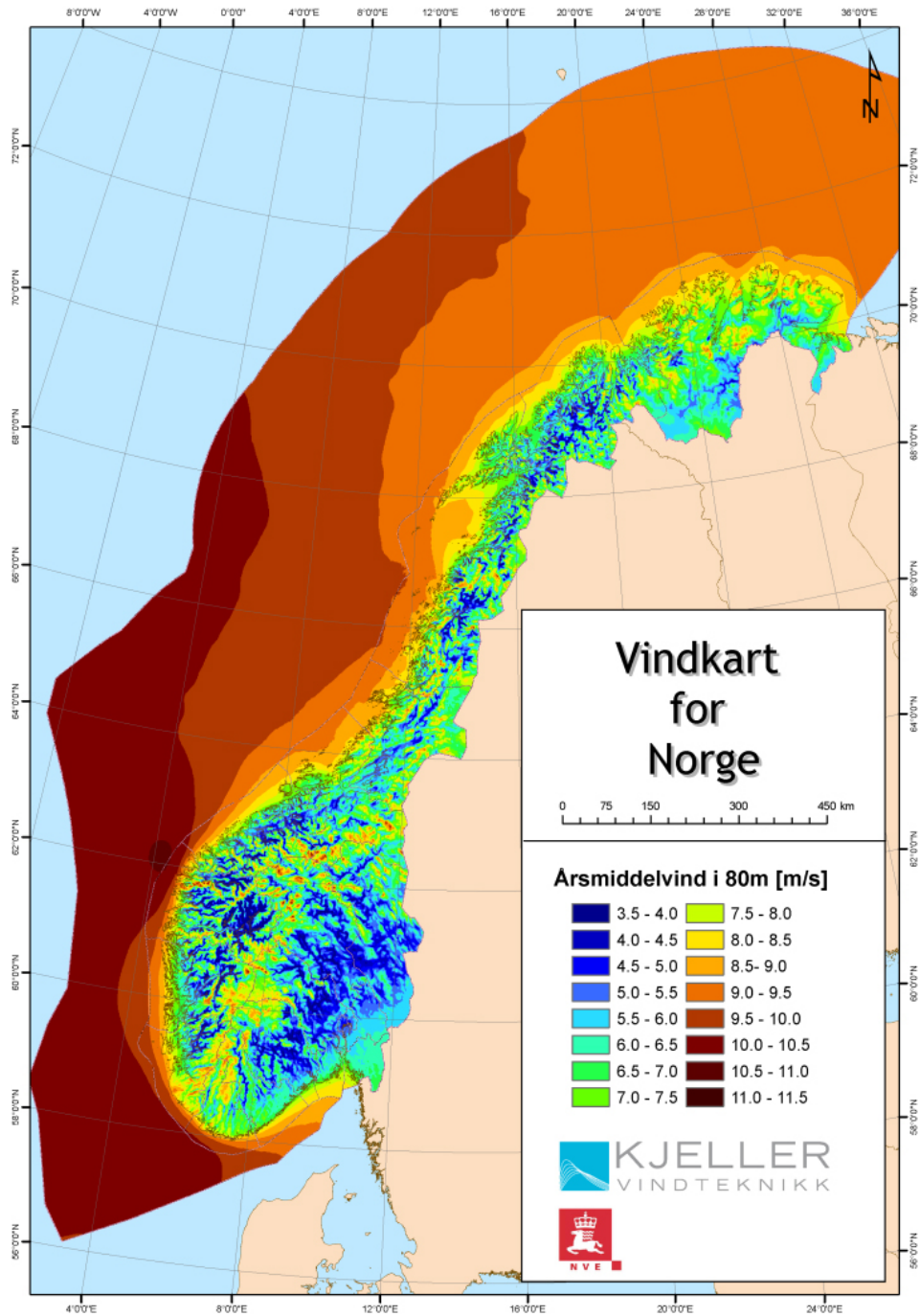


Figure 1.5: Annual average wind speed at 80 m above sea level, Norway. Source: [41]

The wind speed is significantly higher over the ocean than over land. At 80 m above sea level, the average values over ocean is 8-10 m/s. When getting closer to the ground, the wind speed will decrease due to viscosity. A common assumed velocity profile for wind speeds are the *wind profile power law*, which is given as follows:

$$\frac{u}{u_r} = \left( \frac{z}{z_r} \right)^\alpha \quad (1.1)$$

$u_r$  is the reference velocity at the reference height,  $z_r$ , while  $z$  and  $u$  are the new height and new velocity.  $\alpha$  is a factor that is set empirically, which depends on the roughness of the ground. Over land,  $\alpha$  is often set to be  $1/7 = 0.143$ , while over water,  $\alpha = 0.11$  is more appropriate [48].  $\alpha = 0.11$  is used as a value, and the

average value of the wind speed at 80 m height above water is assumed to be 9 m/s. The average wind velocity from  $z = 0$  to  $z = 40$  can then be calculated, which results in a wind speed of 7.5 m/s

Based on this, 7 m/s is used as a representative value for the wind velocity at the oceans outside Norway, and when calculating the forces from wing sails, this value will usually be used. Some calculations are done with different wind velocities, but most use the average value.

## Chapter 2

# Literature Review

In this chapter, the results from an early literature review will be given. At an early stage of the project, some investigations were done in order to get an overview of what the likely outcome of this project would be. The idea of putting wing sails on a cargo ship is not new, and several people have investigated this idea in the past. However, these projects have mostly focused on the potential fuel savings a wing sail can provide. The details of hydrodynamic effects seems to be secondary. Literature about more traditional sailing vessels, such as yachts and small sailboats, have discussions about hydrodynamic effects on the hull under influence of sails, but the focus seems to be conceptual explanation, written in order to make a sailor understand whats going on beneath the ocean surface.

No literature has been found that directly investigates the hydrodynamic effects on a cargo ship under the influence of sails. However, it is known, from general knowledge of lifting surfaces (see for instance reference [2]), that any generation of side force from the hull will create lift-, or side-force-induced resistance. Since the hull have a low aspect ratio, this induced resistance is expected to be high, compared to normal lifting surfaces, such as airplane wings. Experimental data that predicts this resistance due to yaw for a cargo ship have been found, and will be presented in chapter 5. It will not be covered here as this data is used to produce the final results in this report, and therefore deserves special attention.

The effect of letting a cargo ship go with a heel angle has not been found at all. This is probably due to the fact that traveling with a heel angle is a rather uncommon situation for a normal cargo ship. Heel angles are only investigated from a stability perspective, and any effects of heel angle on the resistance is not of any concern when designing a normal cargo ship. Literature that focus on yachts, and yacht design, have discussions about the effects of heel, but the shapes of these boats are rather different than cargo ships, so that the results might not be completely transferable.

First a few projects that have investigated wing sails as a way of propulsion will be given, in order get some examples of the potential fuel savings a wing sail can provide. Second, the hydrodynamic effects of heel and yaw, based on the literature is discussed.

This chapter only contains the result from the literature review that was performed in order to get an overview of the original task at hand. The literature that has been read in order to answer more "minor" questions are given as references when it is natural later in the report. For instance, quite a lot of literature has been read in order to develop the custom Boundary Element Method (BEM) code. A short recap of this literature review is given in the introduction of chapter 3. Some literature investigations were also done in order to get a starting point on how to simulate the fluid flow around ship hulls using Computational Fluid Dynamics (CFD). The result from this literature review can be found in chapter 4.



## 2.1 Projects that Investigates Wing Sails

---

### 2.1.1 Walker wing sails

In 1986, the company "Walker wingsail systems" sold a wing sail to be used on the 3000-ton freighter called "MV Ashington". A article about the company written in 1985 can be found in reference [35]. The article claims that the wing sail could provide 15-25 % fuel savings, however no details are given, as this is not a scientific article, rather a "Popular Science" article. The company went bankrupt right after this, because of loss of interest in wing sail technology, due to the fall in oil prices that happened around that time, however the Walker wing sails are interesting as they seem to be one of the first to use this technology. When they were unable to sell their wing sails to the cargo ship owners, they tried to enter the pleasure craft market, without success.

### 2.1.2 B9 Shipping, Modern Clipper

The company "B9 shipping" is advertising their attempt to develop a sail driven cargo ship, which have gotten quite a lot of media attention, such as in reference [43]. As far as the author can tell, the ship is not yet built, and one should remember that this is company that are selling a technology. When that is said, their claims about the potential fuels savings using wing sails are very good. Based on route analysis and wind tunnel tests, the company claims that their ship design can accomplish 46-55% reduction in fuel. An illustration of the ship can be seen in figure 2.1



Figure 2.1: An illustratuin of the proposed ship design from B9 Shipping

### 2.1.3 UTC Wind challenger

The university of Tokyo presents the "Wind Challenger" project in reference [30]. The project investigates the use of wing sails for a 180 000 Deadweight tonnage (DWT) bulk carrier. The wing sails are 50 m high, with a chord length of 20 m. They propose to use telescopic wings, so that the wing sails can be lowered when in harbor. An illustration of the proposed ship with wing sails can be viewed in figure 2.2



Figure 2.2: An illustration of the UTC wind challenger, at sea. Source: [30]

Reference [30] reports that the wind challenger can achieve a speed of 14 knots when the wind speed is 12 m/s, and directly from the side, and that the average fuel savings when traveling a specific route between Tokyo and Seattle could be around 30 %. The performance of the wing sails were investigated using CFD, and the aerodynamic interaction effects on the wing sails were also studied in reference [27]. These interaction effects will be further discussed in chapter 6.

The wind challenger ship is rather large, and the report of reaching 14 knots in 12 m/s wind directly from the side is not necessarily a very good result. 12 m/s is rather strong wind, and wind directly from the side is the best case scenario (at least, almost the best case scenario, the ship speed will influence the optimal wind angle, but it should be close to direct side wind). However, the specific route simulations shows 30 % reduction in fuel, which is significant. This is however for the "optimal" route, which is longer than the standard shipping route between the two cities. That is, they have found a route with a lot of wind, and defines the savings from wing sails as the savings relative to the case when the ship travels at the same route, only without wing sails. Using the standard shipping route between the two cities only gives fuel savings of 22 %. The article claims to have included hydrodynamic effects due to leeway of the ship, however no details about how this happens are given. They only say that their "energy prediction program" includes the effect of leeway angle.

## 2.2 Forces due to Heel and Yaw based on Yacht literature

---

Yacht literature has been studied in order to get an overview of forces due to heel and yaw for a yacht. A yacht is a very different vessel than a cargo ship. It is a light, small vessel, where almost all of the weight is due to the hull construction. The hull shapes of yachts are also considerably different from cargo ships. They have large transom sterns, and long thin bows, compared to the almost box-shaped cargo ship hulls. Non the less, the general physics should be similar.

Reference [26] goes through the current fluid dynamics knowledge connected to sailing vessel design. The author of reference [26] have worked on several America's cup racing class yachts, and much of the material in reference [26] is therefore connected to high performance yachts

Figure 2.3 is taken from source [26], and shows different resistance components relative to the total resistance for "upwind sailing". That is, it should be a situation with relatively high heel and leeway angles. The figure groups resistance due to heel and side force into one, and based on figure 2.3 it can be seen that it is around 10-20% of the total resistance. This must be considered significant.

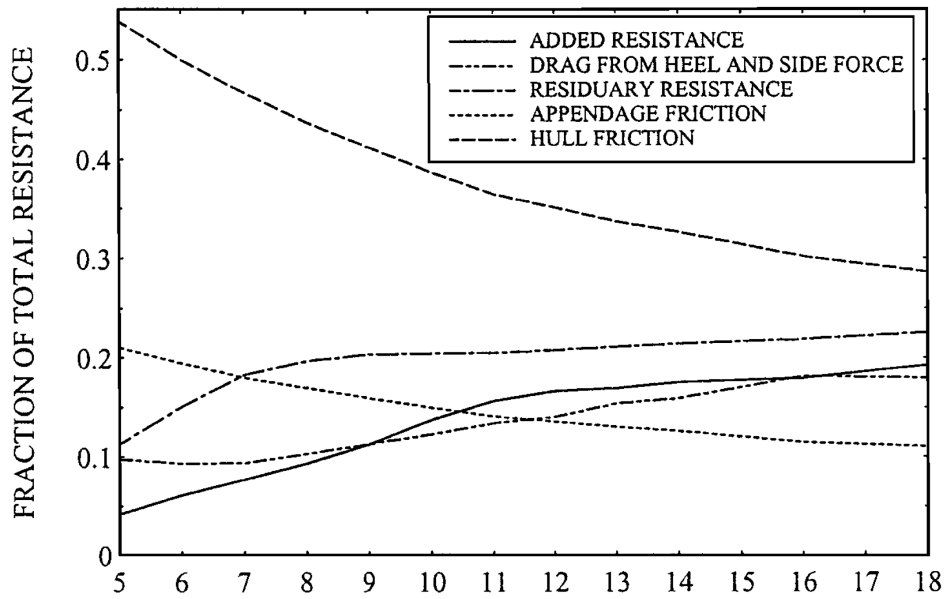


Figure 2.3: Different types of resistance taken from "Fluid Mechanics For Sailing Vessel Design", source: [26]

The resistance due to heel alone is also discussed in reference [26]. The author discuss result from a model test where a yacht model was heeled to 20 degrees, with zero side force, and the result is that the resistance is increased with 4% compared to the resistance without heel. The author of reference [26] concludes that heel is an important effect to consider when trying to evaluate the resistance of a sailing yacht.

Reference [22] is a general book about yacht design. One of the graphs shows a "typical" sailing situation, where the boat is going upwind, and the resulting resistance is increased due to heel and leeway angle. This graph can be seen in figure 2.4, and the result from this situation is that heel causes 5% added resistance, while yaw causes 8.5%. The author of reference [22] therefore makes the point that one should be careful about estimating the resistance of a sailing yacht based on straight-line theories alone. The situation is more complicated for sailing yachts then it is for normal ships.

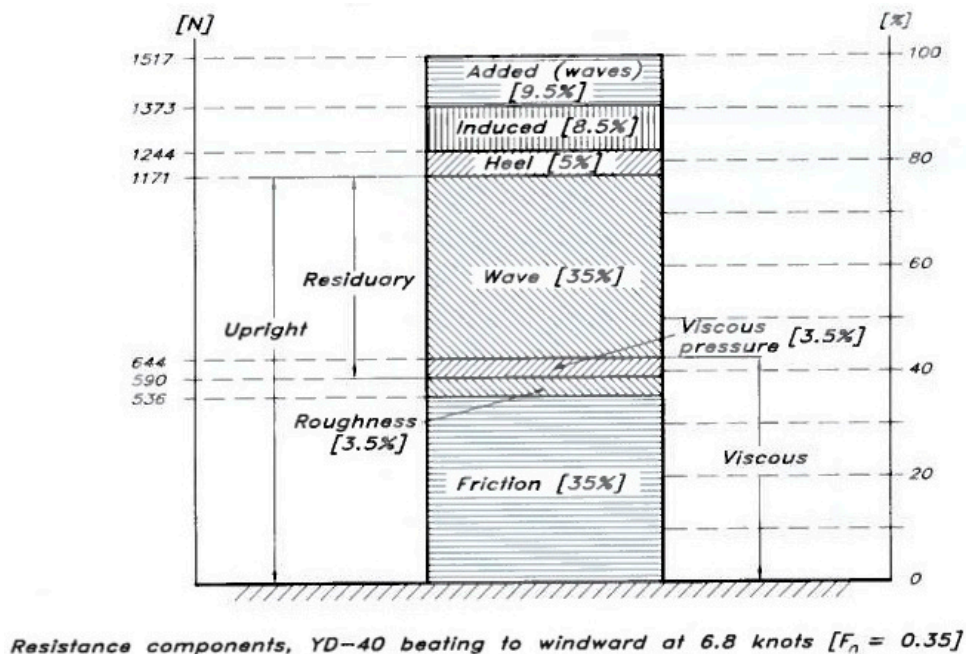


Figure 2.4: Different types of resistance taken from "Principles of Yacht Design", source: [22]

Another book about yacht design is reference [3]. This book lists typical values for increase in resistance due to heel and leeway, together with typical leeway angles. The author of reference [3] concludes that the effect of heel is not very significant. At 10 degrees heel angle, the increase in resistance is only 2%. The added resistance at 20 degrees is reported to be 7%, which is higher than the reports from reference [26], yet the conclusion is still the opposite. While the author of reference [26] states that heel is an important factor for the resistance based on 4% increase at 20 degrees heel, the author of reference [3] concludes that it is not important even though the increase in resistance at this angle is 1.75 times the predicted increase from [26]. The reason for this disagreement must be due to expectations of typical heel angles, and the purpose of the different yachts. Reference [26] seems to have racing yachts in mind.

They all agree that leeway is an important factor for the resistance though. Reference [3] reports that 2 degree leeway might cause 14% increase for a typical yacht, while 4 degrees leeway increases the resistance by 34%. Typical leeway angles are also reported, divided into ship type. For small heel angles (less than 20 degrees) the result from reference [3] is as written in table 2.1

Table 2.1: Typical leeway angles for different ship types, based on reference [3]

Ship type	Typical leeway angles [deg]
Normal sailboat pointing exceptionally high	3
Modern racing yacht	4
Modern ocean racer	4
Modern ocean going cruiser	5
Cruiser of "bad" shape	8
Squared-rigged training ship	12
Sixteenth-century caravel	45

## 2.3 Conclusion from literature review

---

There is no doubt that yaw angles have an considerable effect on the ship resistance. This is fitting with general knowledge of lifting surfaces, and the yacht literature agrees. The effect of heel on the other hand is a bit more uncertain. There seems to be added resistance for large heel angles (>20 degrees) but whether or not such large heel angles are realistic is a different question. Different sources disagree. For smaller heel angles (<20 degrees) the added resistance due to heel seems to be small. If table 2.1 are correct, the leeway/yaw angle of a ship under the influence of sail will be very dependent on the ship shape. However, "normal" sailboats does not seem to experience large heel angles. The question is, what will the effect be for a cargo ship?

## Chapter 3

# Boundary Element Method for 3D Lifting Surfaces

A custom Boundary Element Method (BEM), also known as a panel method, was developed for this project. BEM can simulate the flow around 3D solid objects under the assumption of irrotational, incompressible and inviscid flow, i.e. *potential flow*. Lifting surfaces can be modeled by including a potential wake extending from the trailing edge of the wing (see section 3.1.2). BEM has the benefit of being able to simulate the flow by only discretizing the boundary of the region of interest, rather than the entire volume, which is necessary for Finite Volume Methods (FVMs), Finite Element Methods (FEMs) and Finite Different Methods (FDMs). This reduces the number of unknowns involved in the simulation, and thereby the simulation time.

A potential model of a lifting surface is capable of modeling many of the important effects for a wing sail, such as finding the 3D lift coefficient, the lift induced drag, and some of the interaction effects between wing sails standing in a row (see chapter 6 for more on this). BEM has been the traditional work horse of the aerospace industry when it comes to simulation of lifting surfaces, due to the fact that modeling of the full set of Navier-Stokes equations is very computationally demanding. Even though this is possible today, it is still a significant benefit in having fast simulation methods. Since many of the physical effects involved in the flow around a wing sail can be explained by a potential model, BEM seemed like a good tool.

The specific features of this code is the capability to model an arbitrary number of 3D wings, with thickness, under the assumption of potential flow. Pressure and velocity values can be found, and forces on the wing can be calculated, both by integration of surface pressure, or by the use of the Kutta-Joukowski theorem on the trailing edge of the wing (see section 3.1.6 for more on this). The shape of the potential wake extending from the trailing edge of the wing is calculated with streamline integration. The geometry of the wings are discretized using flat panels, and each panel has a constant value source and doublet strength. Figure 3.1 shows an illustration of a wing model used with the BEM code, along with the calculated wake shape, and color mapped pressure values on the wing.

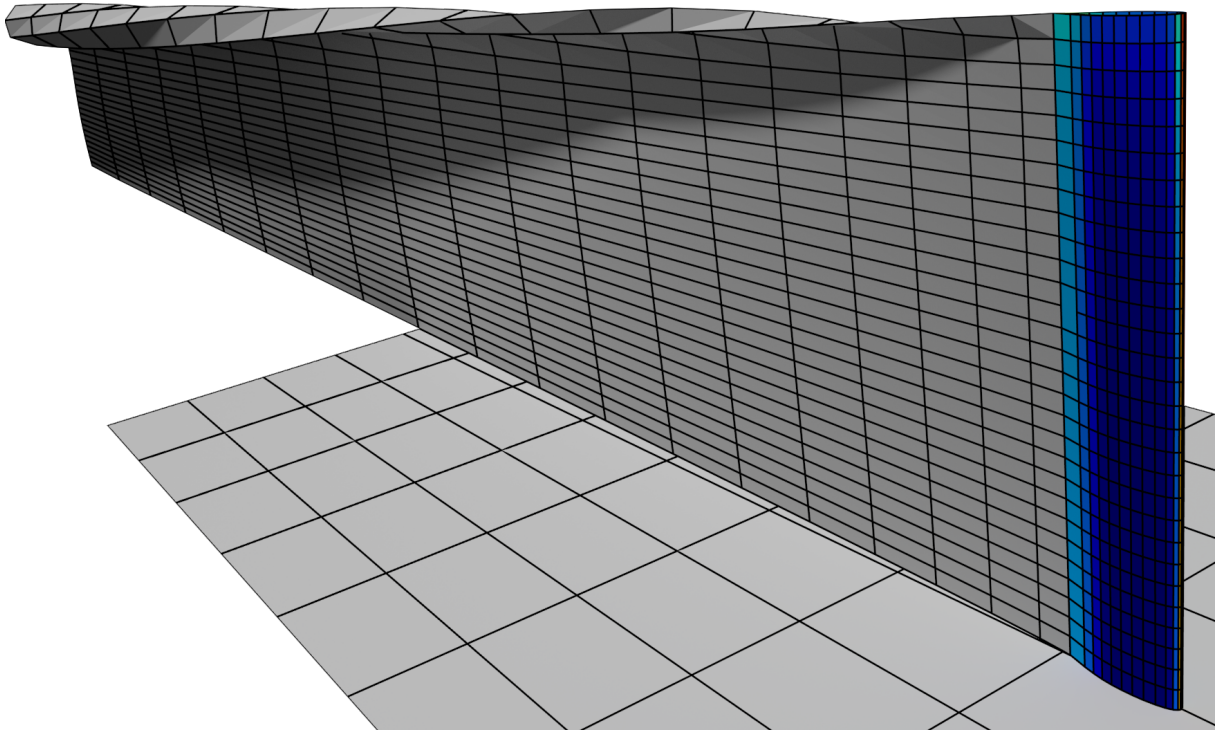


Figure 3.1: Illustration of model used in the custom BEM code, with calculated wake shape, and color mapped pressure values on the wing

There were several reasons for developing a new code from scratch, rather than using some already existing software. One reason is that many of the available codes are old, and not very user friendly, which make them hard to use. Flexibility in how to use the code has also been of importance and has been a focus area in the development process. For this project it means flexibility in terms of geometry of a single wing and number of wings used in the same simulation. In case this code can be reused at a later time, flexibility in terms of coupling this code with other types of simulation codes has also been important. Based on previous experience with 2D panel codes, it was known that *parallel* computation would be a great benefit in terms of simulation time. A lot of effort has been put into making a code that can utilize as much of the computing power available as possible.

Another important reason was that the author had a personal interest in learning this type of numerical simulation properly.

Flexibility in terms of geometry has been achieved by coupling this code with the open source 3D modeling software called Blender. In order to make the code reusable together with other (future) simulation models, it is written almost entirely in the flexible and user friendly programming language Python. Open Computing Language (OpenCL) has been used to access every computational core in parallel, available in any modern personal computer, no matter whether it is a normal (multicore) Central Processing Unit (CPU) or a Graphics Processing Unit (GPU). This is an important feature as modern computers are getting faster and faster by being more and more parallel in nature, and much of the work load in a BEM code is inherently parallel.

Many other versions of BEM codes has been used as reference and inspiration while developing this code. The well known and very popular 2D panel code XFOIL [7] has been used for finding reference pressure distributions for 2D foils. The open-source BEM code called XFLR5 [6] has been studied, which in some sense is a 3D extension of XFOIL. The most important reference has been the theory document for the VSAERO code [25], which has been very helpful, and the general approach in this custom BEM code is very similar to the method used in VSAERO. In addition to the VSAERO theory document, the original documents describing how to evaluate panel integrals (see section 3.1.3) written by Hess and Smith has been of great importance, as this is the original references all other panel codes originates from (Reference [16] and [15]). The book *Low-Speed Aerodynamics* by Katz and Plotkin [18] seems to be a very popular source when

developing BEM codes. The author has also had great success in using this book while developing a 2D panel code at a previous project [20]. The book was extensively used in the beginning of the development of this code, but certain details about an actual numerical implementation of a full 3D BEM code is a bit lacking, and caused some confusion for the author. A mix between reference [18] and [25] is believed to contain the necessary information to develop a BEM code very similar to the one that shall be described in this chapter. Reference [18] for the overall general theory, in a well written understandable manner, while reference [25] for the small, but important details.

This chapter will go through the mathematical theory behind this type of simulation, along with some details about the numerical implementation. There will be some discussion about the more special features, such as coupling with Blender and the use of OpenCL.

### 3.1 Mathematical Implementation

---

The purpose of the BEM code is to find a potential flow solution that describes the lifting flow around wings. This is achieved by solving the Laplace equation, which describes the continuity of mass in an irrotational, incompressible, and inviscid fluid:

$$\nabla^2\Phi = 0 \tag{3.1}$$

$\Phi$  is the velocity potential, and the velocity vector is defined as  $\mathbf{v} = \nabla\Phi$ . The solution must be satisfied in a fluid domain, with boundary conditions that determines the specific behavior of the solution. The boundary consist of solid wall boundaries, wake boundaries, and infinity boundaries. The general boundary conditions for a potential lifting flow problem is that the flow can not go through a solid wall, any disturbances in the flow must diminish when the distance from the disturbance source are approaching infinity, and the potential wake must be such that both the Kutta boundary condition, and Kelvin's circulation theorem is fulfilled.

The Laplace equation has a special property which is especially useful in numerical analysis of the equation. This property can be seen by using two mathematical *tricks*. The first trick is to use Green's second identity. Let  $\phi$  and  $\psi$  be two different scalar functions. If one defines a vector function  $\mathbf{F} = \psi\nabla\phi - \phi\nabla\psi$ , and uses the divergence theorem (also known as Gauss's theorem) with this vector function, one obtains Green's second identity, as follows:

$$\int_{\text{Volume}} (\psi\nabla^2\phi - \phi\nabla^2\psi) dV = \int_{\text{Surface}} (\psi\nabla\phi - \phi\nabla\psi) \cdot \mathbf{n} dS \tag{3.2}$$

If  $\phi$  and  $\psi$  happens to also satisfy the Laplace equation, the term on the left hand side of equation 3.2 will disappear, which leaves the right hand side equal to zero. Equation 3.2 is only valid if the surface integration is performed on a closed surface. That is, if the fluid domain of interest are the domain that extends from the wing, to infinity, with a potential wake included, then the surface of integration must be the "surface" at infinity, the wing surface, and the wake surface. Referring to figure 3.2, the complete surface must be  $S = S_\infty + S_W + S_B$ .

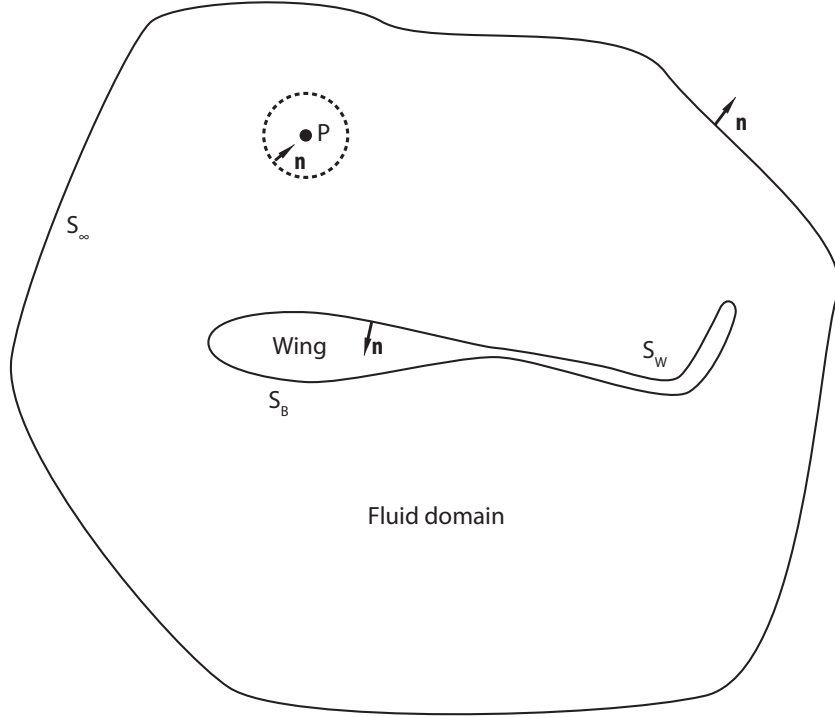


Figure 3.2: Illustration of fluid domain, used as a reference for developing the theory for the BEM code

The second trick is to choose  $\phi$  to be the actual velocity potential that is of interest,  $\Phi$ , and let  $\psi = 1/r$ , where  $r$  is the distance from an arbitrary point,  $P$ . This expression of  $\psi$  is a known solution to the Laplace equation, and represents a source located at  $P$ . The normal derivative of a source is called a *doublet* in the direction of  $\mathbf{n}$ . If  $P$  is located outside the fluid domain, Green's second identity can be used directly, using  $S$  as the surface of integration.

$$\int_S \left( \frac{1}{r} \nabla \Phi - \Phi \nabla \frac{1}{r} \right) \mathbf{n} dS = 0 \quad (3.3)$$

However, if  $P$  is located inside the fluid domain, then one must exclude  $P$  from the domain by letting a sphere surround it, with a radius that goes toward zero. This manages the singularity that occurs for  $\psi$  when  $r \rightarrow 0$ . In doing this, one can find an expression for the value of  $\Phi$  at  $P$ , as a function of the surface integral. The details of how to evaluate the limit can be found in reference [18], chapter 3. The result is equation 3.4

$$\Phi(P) = \frac{1}{4\pi} \int_S \left( \frac{1}{r} \nabla \Phi - \Phi \nabla \frac{1}{r} \right) \mathbf{n} dS \quad (3.4)$$

If the point is located inside the boundary  $S_B$ , then  $P$  is outside the fluid domain, and equation 3.3 holds true. Since this should give zero contribution to the velocity potential at any point inside the fluid domain, another form of equation 3.4 can be written as follows:

$$\Phi(P) = \frac{1}{4\pi} \int_{S_B} \left( \frac{1}{r} \nabla (\Phi - \Phi_i) - (\Phi - \Phi_i) \nabla \frac{1}{r} \right) \mathbf{n} dS + \frac{1}{4\pi} \int_{S_W + S_\infty} \left( \frac{1}{r} \nabla \Phi - \Phi \nabla \frac{1}{r} \right) \mathbf{n} dS \quad (3.5)$$

$\Phi_i$  is called the internal potential. Because the contribution from both sources and doublets disappear when  $r \rightarrow \infty$ , the contribution from the surface at infinity can be expressed as  $\phi_\infty$ , which has a value corresponding to the free stream velocity potential. In addition, if one assumes the wake to be thin, then



the normal velocity should be continuous across it, in order to make physical sense. The value of  $-(\phi - \phi_i)$  can be considered the doublet strength ( $\mu$ ), while the value of  $-(\nabla(\phi - \phi_i)\mathbf{n})$  can be considered the source strength ( $\sigma$ ). This gives the following relation for the velocity potential at any point in the fluid domain:

$$\Phi(P) = -\frac{1}{4\pi} \int_{S_B} \sigma \frac{1}{r} - \mu \mathbf{n} \nabla \left( \frac{1}{r} \right) dS + \frac{1}{4\pi} \int_{S_W} \mu \mathbf{n} \nabla \left( \frac{1}{r} \right) dS + \phi_\infty(P) \quad (3.6)$$

$$\Phi(P) = \phi + \phi_\infty \quad (3.7)$$

That is, the value of the unknown wanted potential at point  $P$  can be quantified by distributing sources and doublets with unknown strengths at the boundaries of the domain. There is no need to use unknown values inside the domain, as every solution to the Laplace equation can be represented by sources and doublets at the boundaries of the domain. The challenge is to find the values of the strengths. The values can be found numerically, by discretizing the boundary surface, and performing the integration numerically. The equations that determines the values of  $\sigma$  and  $\mu$  are based on the boundary conditions for the specific problem. This generates a linear equation system in the end, with unknown variables representing the strengths. When the strengths are found by solving the system of equations, the velocity and pressure in the fluid domain can be found. Velocities are found by taking the derivative of the potential, and the pressure is found using Bernoulli's principle.

### 3.1.1 Solid Wall Boundary Conditions and Singularity Mix

In order to know what the strengths at the boundaries should be, boundary conditions must be used. This is what generates the system of equations. The boundary of the solution domain is discretized, and boundary conditions are evaluated at discretized points at these boundaries. The most important boundary conditions is the one for a solid wall. There are two ways to set up the solid wall boundary condition: Neumann boundary condition, specifying the velocity at the wall, or Dirichlet boundary condition, specifying the velocity potential at the wall.

The Neumann boundary condition is the one that is easiest to see. It says that the velocity normal to a solid surface must be zero, because otherwise, the flow would go through the surface, and the surface would not be a solid wall. Or in mathematics:

$$\frac{\partial \Phi}{\partial n} = 0 \text{ at } S_B \quad (3.8)$$

$$\frac{\partial \phi}{\partial n} = -\frac{\partial \phi_\infty}{\partial n} \text{ at } S_B \quad (3.9)$$

$$\frac{\partial \phi}{\partial n} = -\mathbf{n} \mathbf{U}_\infty \text{ at } S_B \quad (3.10)$$

This type of boundary conditions is the one used by Hess and Smith in reference [15] and [16]. For thin bodies, this would also be the only option. However, for this code, and also many other BEM codes such as both VSAERO and XFLR5, Dirichlet boundary condition is used.

Dirichlet boundary condition specifies the value of the potential at the boundaries. The value of the potential is evaluated on the inside of the body surface, so that it is an evaluation of the inner potential. If the potential inside the body is constant, there is no change in  $\partial\phi/\partial n$ , so that the Neumann boundary condition is set indirectly. One form of the Dirichlet boundary condition for a closed surface is therefore the following:

$$\phi_i = \text{constant} \quad (3.11)$$

What the constant should be, is not necessarily unique. Different approaches exist, where one is to set the inner potential to be zero. In this project, a slightly different approach is used, which is the same approach used in VSAERO (see reference [25]) and also the method recommended by reference [18].

Consider the fact that the source strength represents the jump in the normal derivative of the potential. That is,  $\sigma$  is given as follows:

$$\sigma = - \left( \frac{\partial \phi}{\partial n} - \frac{\partial \phi_i}{\partial n} \right) \quad (3.12)$$

If the source strength is set to be equal to the normal component of the free stream velocity potential the following relationship would occur:

$$-\sigma = -\mathbf{n} \mathbf{U}_\infty \quad (3.13)$$

$$\left( \frac{\partial \phi}{\partial n} - \frac{\partial \phi_i}{\partial n} \right) = -\mathbf{n} \mathbf{U}_\infty \quad (3.14)$$

Or in other words, the source strength for each panel is set so that each panel enforces the Neumann solid wall boundary condition, as if there only was a free stream, by itself. However, each panel also induces velocity potentials on all the other panels as well, which means that this is not entirely correct alone. The strength of the doublets must then be set in order to balance the error from the source strength, so that the total velocity potential, induced from both sources and doublets are zero. Or in mathematics:

$$\frac{1}{4\pi} \int_{S_B+S_W} \mu \mathbf{n} \nabla \left( \frac{1}{r} \right) dS - \frac{1}{4\pi} \int_{S_B} \sigma \frac{1}{r} dS = 0 \quad (3.15)$$

Where the source strength is known, and given by equation 3.13. This approach has the added benefit of giving a reasonable choice for a singularity mix between sources and doublets as well. That is, there is no clear boundary condition that determines the relationship between sources and doublets. Theoretically, either one of  $\sigma$  and  $\mu$  could be set to zero, but this approach is known to generate numerical difficulties (see reference [18]).

Reference [18] claims that this approach is better than using Neumann boundary conditions without really specifying why it should be better. Reference [15] suggest that there is very little difference between using Neumann and Dirichlet boundary conditions, and it's a matter of personal preference. Reference [25] discusses a few different options regarding the exact way of implementing Dirichlet boundary conditions, concluding that the way it is done in this project has some numerical benefits, which in the end is due to the fact that the strengths usually end up with values that are not too high or too low, so that rounding errors is not a problem.

The author is not sure whether or not this type of boundary condition is "the best", however, it does seem to be a popular choice. Both VSAERO and XFLR5 uses the same approach, and reference [18] claims that this method is the most popular based on "recent trends".

### 3.1.2 Potential wake

In order to model a lifting flow, the Kutta-condition for lift must be satisfied. That is, at the trailing edge, the flow must leave the trailing edge smoothly. This fixes the total circulation around the wing, so that it models the inherently viscous phenomena of lift, without modeling viscous flow directly. This condition is satisfied automatically by letting a potential wake, that only contains doublet panels, extend from the trailing edge. In order to satisfy Kelvin's circulation theorem, the doublet strength in the wake must be set equal to

the doublet strength at the trailing edge. Kelvin's circulation theorem states that circulation are conserved. Doublets have circulation, as they are closely connected to vortices. In fact, a constant strength doublet panel is equivalent to a "vortex ring", or four connected vortex lines making up a quadrilateral, where the circulation of each vortex line is equal to the strength of the doublet. See reference [18] for more on this.

First, the entire wing, and wake, is divided into strips, where the strips are going in the chordwise direction of the wing. See figure 3.3 for a visual explanation. Each strip have one value of trailing edge doublet strength. The trailing edge at each strip is connected to two panels, one on the "upper" side and one on the "lower" side of the wing. The upper side is defined as the panel that have a normal that points in the same direction as the wake panels.

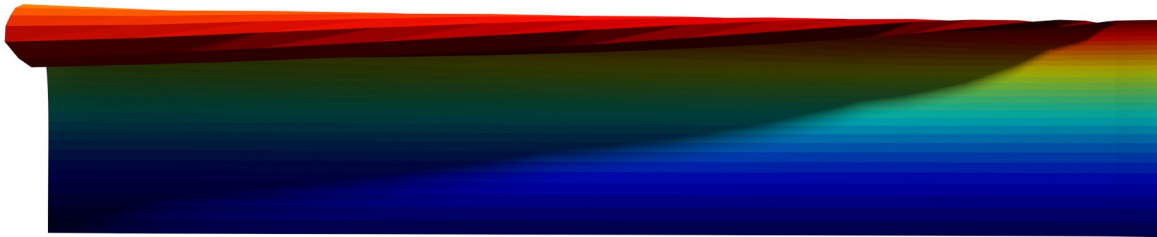


Figure 3.3: The wake and wing divided into strips, where each separate color represents a separate strip

The doublet strength of the trailing edge is defined as the strength of the upper panel, minus the strength of the lower panel. That is,  $\mu_W = \mu_U - \mu_L$ . Each wake panel in the same strip have the same strength,  $\mu_W$ . Since there is no source panels in the wake, there is no jumps in the velocity, only jumps in the potential. In order to satisfy Kelvin's circulation theorem perfectly, the wake should extend to infinity. However, as long as the wake is extended to a finite length away from the wing that is large enough, the effect of the wake on the wing will be as if the wake is infinitely long. Around 25 chord lengths are found to be long enough, based on convergence analysis, of which the results can be seen in the appendix section 11.1.6. This also matches claims about the same topic in reference [18].

Setting the wake-doublet strength in this manner enforces both the Kutta condition and Kelvin's circulation theorem. However, in order to make physical sense, the shape of the wake must be correct. That is, as the wake is not a fixed solid wall, it will not be able to carry any loads. Or in other words, it needs to follow the stream lines in the flow. In the same way as for a solid wall, there can be no normal velocity component at the wake surface, but instead of forcing the flow to follow the wake, one must force the wake to follow the flow. This will force the wake to have the characteristic roll-up structure.

There exist a few different ways of dealing with the wake shape. One is to simply let the wake follow the free stream, neglecting the influence from the wing and the wake itself. This is known as a "drag-free" wake, as it carries loads, but not in the drag direction. XFLR5 uses this approach. This is not correct, but the error is not large for small angles of attack, and it is considerably simpler to implement. Another approach is to set a wake shape based on experience. It is known that the wake will shed of the trailing edge in a smooth manner, but after a while start following the free stream. Manually adjusting the wake shape to "look" right is possible, and could be better than a drag-free wake, if the user is experienced in wake shapes. A third way is to use what's called a "wake relaxation" approach. This method is described in reference [18], and the author tried to implement this approach, only to discover that it was very unstable. It was discovered, at a later point, that the developers of XFLR5 had discovered the same thing. They had implemented the method, but deactivated it due to its instability [6]. The way the method works is by calculating the induced velocity at every wake vertex, and deform the wake by moving each wake vertex with the induced velocity. However, this easily creates "unphysical" wake roll-up, which means that the wake deforms in to itself, which

causes problems.

The way the wake is deformed in this code, is by streamline integration. Each line extending from the vertices at the trailing edge of the wing is treated as a streamline. The lines are defined as the horizontal lines in the wake mesh. For each horizontal line between two vertices, the velocity is calculated for two points. If vertex 1 is called  $p_1$  and vertex 2 is called  $p_2$ , then the first velocity evaluation point is defined as  $0.75p_1 + 0.25p_2$  while the second point is defined as  $0.25p_1 + 0.75p_2$ . The average velocity is defined as  $u = 0.5(u_1 + u_2)$ . See figure 3.4 for a visual explanation.

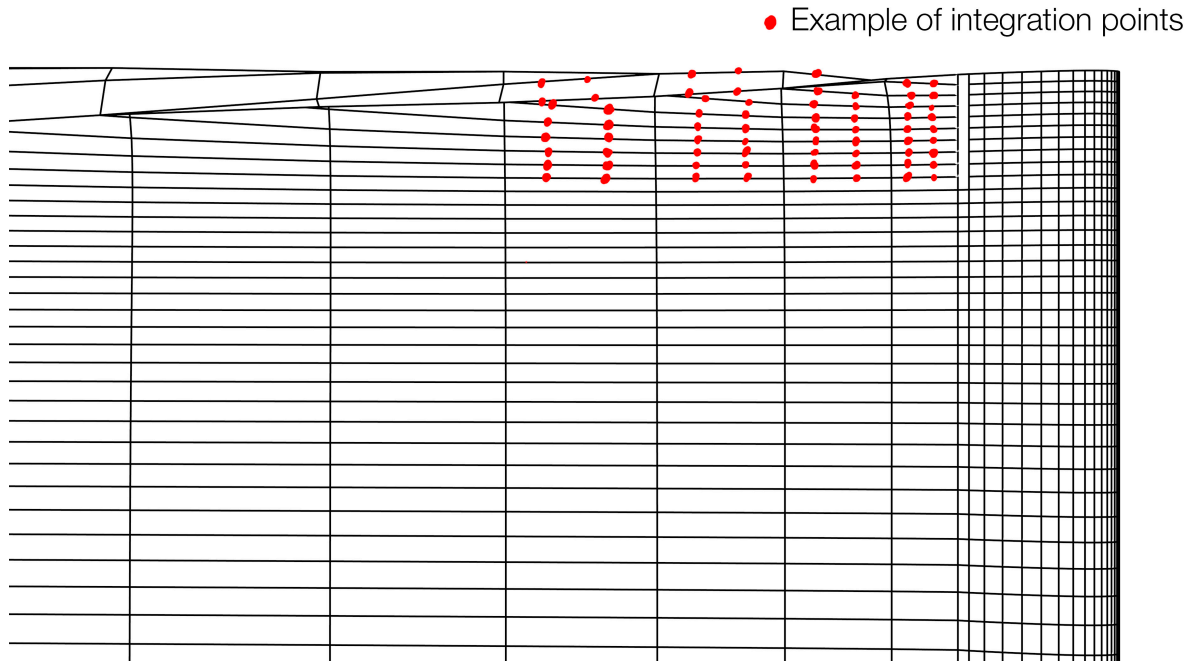


Figure 3.4: Illustration of some integration points in the wake (conceptual illustration)

The local time step,  $\Delta t$ , is defined as the distance between the two points, divided by the free stream velocity. A new value of  $p_2$  can then be calculated as follows:

$$p_{2,new} = p_1 + u\Delta t \quad (3.16)$$

The drag free wake is used as a first approximation for the wake shape, and then the integration procedure explained here is used to update the wake shape. This can be done several times, and the wake shape should converge. The overall shape of the wake seems to converge rather quickly, but the roll-up structure of the wake can end up being "unphysical" even with this approach. The same approach is used in the VSAERO code, and the theory document there also describes the problem with "unphysical" roll-up [25]. Even if unphysical roll-up can happen with too many iterations, this approach seems to find a good wake shape. In fact, after the first iteration, there are very small changes in the calculated lift and drag coefficients (see the appendix section 11.1.6 for more). All the results presented in this report are done with 3 wake shape iterations. VSAERO uses 2 wake shape iterations as default.

### 3.1.3 Evaluation of Panel Integrals

The integration of a doublet and source panel with unit strength is needed in order to evaluate the influence from panels at a point. For a panel with constant strength, the integrals to be solved has the following form:

$$\phi(x, y, z) = \iint_{\text{Panel}} \frac{1}{r(x, y, z)} dS \quad (3.16 \text{ Source})$$

$$\phi(x, y, z) = \iint_{\text{Panel}} \mathbf{n} \cdot \nabla \frac{1}{r(x, y, z)} dS \quad (3.16 \text{ Doublet})$$

The evaluation of these integrals are not a trivial task. Hess and Smith (reference [15] and [16]) solved these integrals for a flat polygon shaped panel. Their approach can be used no matter how many sides the polygon has. That is, the general idea works equally well for a triangle as for rectangle, or any other polygon. The basic idea is to divide the polygon into strips, where each strip correspond to the edges of the polygon. each strip is then evaluated by it self, by constructing a surface that goes to infinity in one coordinate direction (it does not matter whether it is the x-axis or y-axis), but is limited by the strip in the other direction. The strength of this artificial surface is set to be equal to the source/doublet strength divided by two on the side of the strip where the actual polygon is located, and the same value only negative on the other. This leads to an integral equation that is possible to solve, and by summing up all the sides of a polygon, the effect on the area outside the polygon will be canceled, while the effect on the area inside the polygon will sum up to be the correct value. This is hard to explain in a short manner, but reference [16] explains it well. The result of this type of integration is taken directly from [15] and [16]. The expressions used in this code is written below, in a manner that correspond to the way they are written in the actual code. The expression are valid for one edge of a polygon. As each polygon in the BEM code consist of four edges, the expressions written below must be executed four times, once for each edge. The nomenclature is shown in figure 3.5. In particular,  $\mathbf{p}_1$  is the first point of the edge,  $\mathbf{p}_2$  the second point,  $\mathbf{a}$  and  $\mathbf{b}$  is vectors from edge points to the control point,  $\mathbf{p}$ , while  $\mathbf{s}$  is the vector from  $\mathbf{p}_1$  to  $\mathbf{p}_2$ .  $\mathbf{l}$ ,  $\mathbf{m}$ , and  $\mathbf{n}$  are the local coordinate system vectors. That is, the vectors corresponding to the direction of the local coordinate system axis, expressed in the global coordinate system. Each polygon have a local coordinate system, where the normal of the polygon corresponds to the local z-axis. All the variables are vectors, and for instance  $\mathbf{a}.x$  is the x-component of the vector  $\mathbf{a}$ .

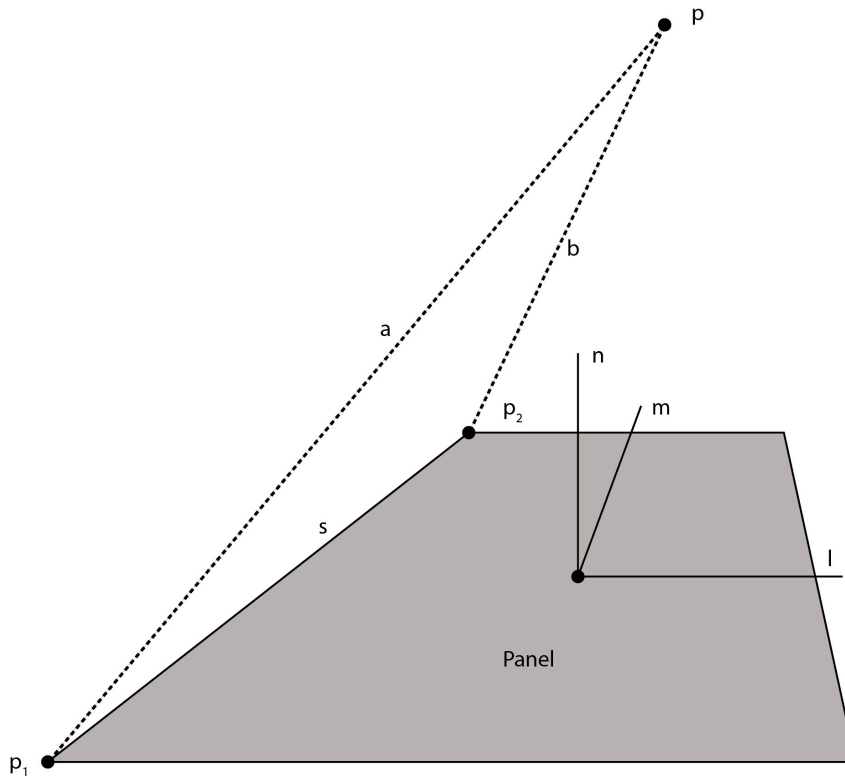


Figure 3.5: Nomenclature used while expressing the result of the integral equations for a flat polygon shaped panel

General variables used in the result:

$$S = \sqrt{\mathbf{s}.x^2 + \mathbf{s}.y^2 + \mathbf{s}.z^2} \quad (3.17)$$

$$A = \sqrt{\mathbf{a}.x^2 + \mathbf{a}.y^2 + \mathbf{a}.z^2} \quad (3.18)$$

$$B = \sqrt{\mathbf{b}.x^2 + \mathbf{b}.y^2 + \mathbf{b}.z^2} \quad (3.19)$$

$$PN = \mathbf{p} \cdot \mathbf{n} \quad (3.20)$$

$$SL = \mathbf{s} \cdot \mathbf{l} \quad (3.21)$$

$$SM = \mathbf{s} \cdot \mathbf{m} \quad (3.22)$$

$$AL = \mathbf{a} \cdot \mathbf{l} \quad (3.23)$$

$$AM = \mathbf{a} \cdot \mathbf{m} \quad (3.24)$$

$$AI = AM \cdot SL - AL \cdot SM \quad (3.25)$$

$$PA = PN^2 SL + AI \cdot AM \quad (3.26)$$

$$PB = PA - AI \cdot SM \quad (3.27)$$

$$RNUM = SM \cdot PN \cdot (B \cdot PA - A \cdot PB) \quad (3.28)$$

$$DNOM = PA \cdot PB + PN^2 A \cdot B \cdot SM^2 \quad (3.29)$$

$$C = \text{atan2}(RNUM, DNOM) \quad (3.30)$$

$$GL = \frac{1}{S} \log \left| \frac{A + B + S}{A + B - S} \right| \quad (3.31)$$

Source potential:

$$\phi = AI \cdot GL - PN \cdot C \quad (3.32)$$

Doublet potential:

$$\phi = C \quad (3.33)$$

Source velocity:

$$\mathbf{u} = GL \cdot SM \cdot \mathbf{l} - GL \cdot SL \cdot \mathbf{m} + C \cdot \mathbf{n}; \quad (3.34)$$

Doublet velocity:

$$\mathbf{u} = \frac{\mathbf{a} \times \mathbf{b} (A + B)}{A \cdot B (A \cdot B + \mathbf{a} \cdot \mathbf{b})} \quad (3.35)$$

There are a few situations where these expressions need special care while evaluating them. First of all, the inverse tangent function is written as "atan2". This is to clearly express that the inverse tangent function is a four quadrant function. Without taking all the four quadrants into account, the result will be wrong. In addition, the inverse tangent function will need special care if  $\mathbf{p}.z \rightarrow 0$ . If  $\mathbf{p}.z \rightarrow 0$  and  $\mathbf{p}$  is on the panel itself, then  $C \rightarrow -\pi$ . Otherwise,  $C \rightarrow 0$ , unless  $\mathbf{p}$  is on the edge of the panel, then  $C \rightarrow -\pi/2$ . The expression for the doublet velocity will go towards infinity if it is evaluated on the edge of a panel. A way to deal with this is to pretend there exist a very small viscosity in the fluid that limits infinite velocities, and set the induced velocity to zero if it is evaluated on the edge of a panel. This is a necessary step in order to evaluate the velocity in the wake, as this happens on the horizontal edges of the wake mesh. This is a normal "trick" although it is not technically mathematically correct, within the potential theory. It simply avoids a problem with this

method, by assuming it away. The name of this trick is "finite vortex core" (see for instance reference [18], [25])

### 3.1.4 Ship Deck Modeling

The wing sails will stand on the deck of a ship, which means that the effect of the deck must be modeled. This is done by mirroring in the BEM code. The z-axis is used as a mirroring axis, which means that the mirrored panel have the same coordinates as the original panel, only with the negative z-values. It is not necessary to transfer the entire panel to the mirrored position. Only the control point needs to be mirrored. The velocity potential from a panel on a mirrored point is equal to the velocity potential from a mirrored panel on the original point. When evaluating the induced velocity, the z-component needs to be reversed when using this approach. Using this type of modeling of the ground plane will simulate an infinite ground plane. The deck of a ship is not infinite, so this is an approximation. Even though the deck is not infinite, the water plane is practically infinite and, depending on the ship, the water plane is also close to the wings. Since the density of water is so much larger than the density of air, the effect of the water plane should be very similar to the effect of a ground plane.

### 3.1.5 Analysis of Velocity and Pressure

When the strengths of the sources and the doublets are known, along with the shape of the wake, the velocity and pressure can be found. If the velocity should be found anywhere that is not on the surface of the wing, the induced velocities must be calculated by equation 3.34 and 3.35. This is the approach used when calculating the velocities in the wake, in order to deform the wake. However, this is rather time consuming, as every panel induces velocities at every point. Luckily, there exist a faster method for the surface of the wing.

By evaluating the limit of the expressions for induced velocity when approaching a panel, the following can be found for the local velocity components (see reference [18] and [25] for more):

$$u_l = -\frac{\partial\mu}{\partial l} \quad (3.36)$$

$$u_m = -\frac{\partial\mu}{\partial m} \quad (3.37)$$

$$u_n = \sigma \quad (3.38)$$

Here,  $u_l$  is the velocity in the direction of the local coordinate axis  $l$ ,  $u_m$  in the direction of the local coordinate axis  $m$  and  $u_n$  in the direction of the local coordinate axis  $n$ . This means that the velocity can be calculated much faster on the wing surface than anywhere else in the fluid domain, as the velocity on the wing surface only depends on the local source and dipole strengths. The derivation must happen numerically, since a panel have constant values of  $\mu$  and  $\sigma$ . This is done with finite difference in the code, using the neighboring panels. Central difference is used for all panels that have four neighbors, while panels on the edges of the wing, where neighbors are only available to one side, use either forward or backward differentiation, depending on where the neighbor is located.

When the velocity is known, the pressure can be found, using Bernoulli's principle. Only the pressure coefficient is calculated, never the actual pressure. The pressure coefficient is defined, and calculated, as follows:

$$C_p = \frac{p - p_0}{\frac{1}{2}\rho U_\infty^2} = 1 - \frac{u^2 + v^2 + w^2}{U_\infty^2} \quad (3.39)$$

### 3.1.6 Force Calculation

Two methods of force calculation are implemented in this custom BEM code. First, only pressure integration was implemented. The force acting on each panel is then defined as the pressure times panel area times the negative normal vector. That is, the following is done in order to calculate the total force acting on a wing:

$$F_{\text{wing}} = - \sum_{\text{all wing panels}} p \cdot A \cdot \mathbf{n} \quad (3.40)$$

This method works fine for finding the lift on the wing, but it is very inaccurate for calculating the induced drag. This is a known problem with the method, which for instance is discussed in reference [4]. The drag force is much smaller than the lift force, so that small errors in the pressure can affect the drag force much more than the lift force. In fact, calculating the force by pressure integration will predict a small drag force for wings without lift. This is not correct, as there can be no drag without lift in a potential flow.

In order to get accurate values for the induced drag, the Kutta-Joukowski theorem for forces on a wing with a known circulation is used. This is the same method as described in reference [4], and the claim is that this method is the method that are least dependent on both panel density and wake shape. The method calculates the velocity induced by the wake, and the free stream, at the trailing edge of the wing. Then, according to Kutta-Joukowski's theorem for forces on a lifting surface, the force is calculated as follows:

$$F_{\text{wing}} = \sum_{\text{all trailing edges}} \rho \cdot \mathbf{U} \times \Gamma \quad (3.41)$$

In this equation,  $\mathbf{U}$  is the velocity at the trailing edge, neglecting the influence of the wing. If the wing is included in the velocity calculations, there will be no velocity at the trailing edge, as this is a stagnation point. The argument for using this method is that the trailing edge is the first point "on" the wing where there can be "down wash", as it is the first point where the solid wall stops.  $\Gamma$  is the circulation at the trailing edge, which is equal to the doublet strength of the wake. The direction of  $\Gamma$  is along the trailing edge. The argument is that total circulation of the wing must be shed into the wake, according to Kelvin's circulation theorem.

Every drag calculation in this report is done with the trailing edge method. See section 3.3 to see verification experiments performed to test the force values calculated on the wing.

## 3.2 Numerical Implementation

---

Based on the theory presented in section 3.1, the BEM is implemented numerically. The main programming language is Python, except for some parts that are written in C. Python was chosen as a programming language as this is a simple and user friendly programming language. The parts of the code that are written in C are the parts that uses OpenCL to execute code in parallel. This is mainly the construction of equation systems, along with calculation of induced velocities. The custom BEM code is written to be executed in the open source 3D geometry software called Blender. Blender acts as a geometry "kernel". That is, every piece of geometry that is used in the simulation is constructed and managed by Blender.

This section will go through the structure of the program, along with some discussion about the connection to Blender and the use of OpenCL. The complete BEM code can be seen in the appendix section 11.1.



### 3.2.1 Overview of the Program

The entire program is written in an object oriented manner. There are two classes: one that works as an interface between Blender and the rest of the code, called "Geometry", and one that have the actual computation methods, called "Computation". The Geometry used in the code is taken directly from Blender. Vertices, topology, normals, etc. is accessed through the Blender Application Programming Interface (API). This geometry is used as input for the computation methods. The computation methods calculates influence matrices, based on the geometry, which finally is used to solve a linear system, using the Numerical Python (NumPy) package. The "heavy lifting" in the program is done with OpenCL "kernels". Kernels are functions that can be executed in parallel with OpenCL. See section 3.2.2 for more on this.

The software is written as a set of methods/functions that can be executed independently. That is, the BEM is divided into separate tasks, where each task gets its own method. In order to simulate anything with the code, a control script must be written, that runs the necessary methods in the correct order. This approach was used, because there might be variations in the way one wants to execute the simulation. For instance, the calculation of velocity and pressure on the wing exist as a separate method. This is a function that might not be necessary to execute, since forces can be found without calculating the pressure, using the trailing edge method. A typical work flow while using the BEM code can be as follows:

1. Create the geometry that should be used in the simulation using Blender. Both wings and wakes must be constructed. The geometry must have a certain structure, in order to make the software understand how the wing and the wake is divided into strips. This is accomplished using NURBS in Blender, as the structure of NURBS objects is very appropriate for this type of simulation. See section 3.2.3 for more on this.
2. For each wake and wing to be used in the simulation, create a geometry instance. The geometry class consists of methods to transfer data from Blender to OpenCL, methods for dividing the geometry into strips, and variables and methods to store and calculate velocity and pressure. Post processing methods, that creates colormaps is also in the geometry class.
3. Send the necessary data to OpenCL by calling the right method. This is a necessary step for using OpenCL. See section 3.2.2 for more on this.
4. Calculate influence matrices based on wing geometry. Two influence matrices are needed: one for sources, and one for doublets. Each row in the influence matrix correspond to a certain control point, while each column corresponds to the influence from a certain panel.
5. Calculate the source strength, based on the free stream values, and equation 3.13.
6. Using the already existing doublet influence matrix for the wings as input, modify it to include the influence from the wake panels. That is, for each control point on the wing, calculate the influence from a wake strip, and add it to the location of the two trailing edge panels in the influence matrix.
7. Multiply the source influence matrix with the calculated source strengths to generate a vector, **b**.
8. If the doublet influence matrix is called **A**, solve the system  $\mathbf{A} \cdot \mu = \mathbf{b}$  by using the built in solver in NumPy. This solves for the values of  $\mu$ , so that the strengths of the doublet panels are known.
9. When the strengths of both the source panels and the doublet panels are known, the wake can be deformed according to the solution, and the velocity, pressure, and force on the wing can be calculated
10. When everything is known, post processing in Blender can be performed, such as color mapping the pressure values, updating the wake geometry, etc.

An example of a control script can be found in section 11.1.

### 3.2.2 OpenCL

One important feature of this BEM code is the use of OpenCL. OpenCL is a framework for writing code that can be executed on almost any parallel computation unit in a modern computer. That is, OpenCL can be executed equally well on a CPU as on a GPU. In particular, the GPU aspect of OpenCL was of a particular interest for the author. General computation on a GPU is something that is relatively new, and the author was interested in learning this type of programming. OpenCL is not the only way to write code that can be executed on a GPU, however, it is the only way to write code that can be executed on both a CPU and a GPU. OpenCL was initially developed by Apple, but is today open source and managed by the Khronos Group [46].

OpenCL works by executing what's called kernels in parallel. The number of threads to be used is automatically managed by OpenCL, which is a nice feature. The kernel code is written in C. In order to execute OpenCL code, the variables that are to be used in the calculations must be transferred to "OpenCL buffers". This is necessary as a GPU will have its own memory, and cannot access the normal memory of a computer. In order to make the same code executable on any computational device, the programmer needs to explicitly tell the software which variables that should be used by the OpenCL kernels. Transferring all the variables in a code is a bad idea, as the transfer from the normal computer memory to, for instance, the GPU memory can take quite a lot of time. This transfer time should be considered when deciding on which computational unit to use.

The communication between OpenCL and Python is done with "PyOpenCL" [19]. In order to test the speed of PyOpenCL, compared to normal Python, a numerical experiment was performed. The result of this experiment can be seen in figure 3.6. The time it takes to build an equation system, based on random source panels, are tested. PyOpenCL is executed both with the GPU and the CPU on the author's personal computer.

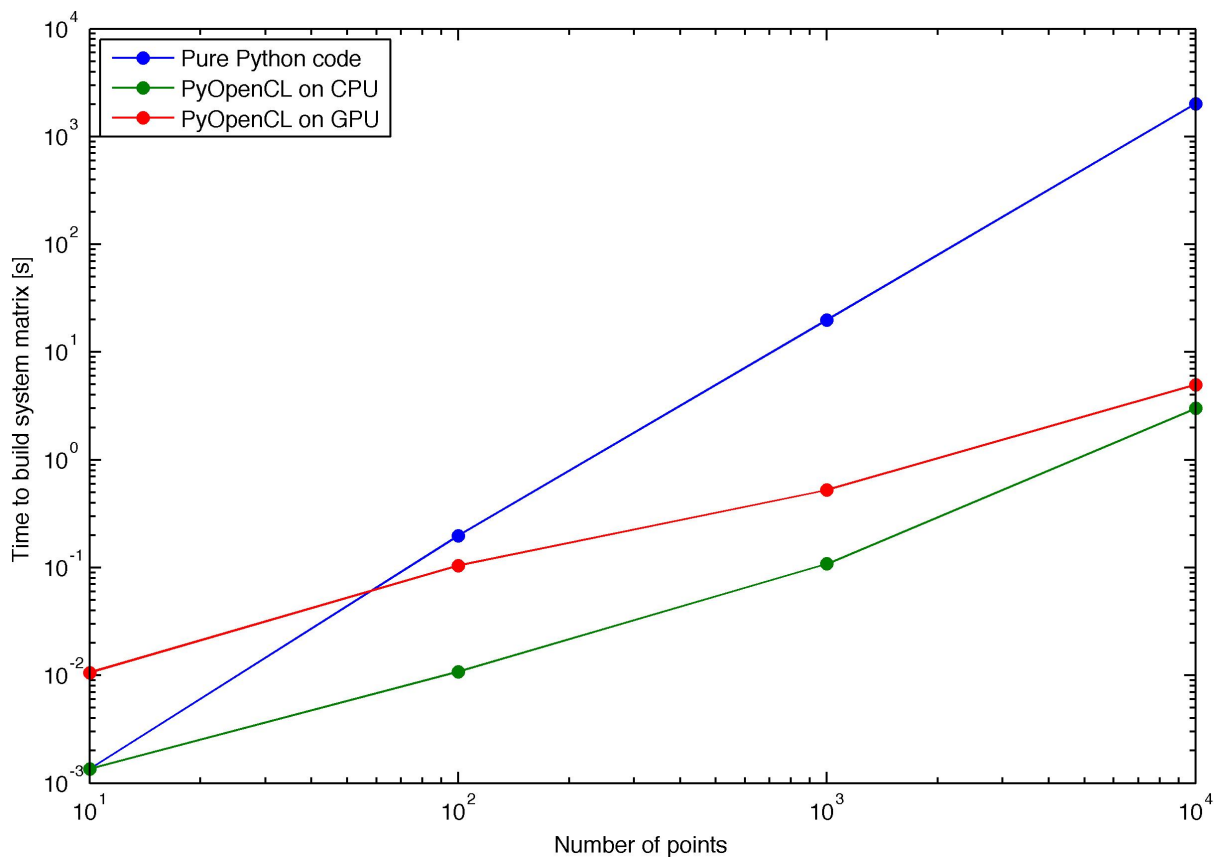


Figure 3.6: Time to build an equation system, done with different calculation methods

Notice that the axis in figure 3.6 are logarithmic. That is, the difference between pure python and PyOpenCL is several orders of magnitude for a large number of panels. When the number of panels are small, the difference is also small, which is expected. For the authors personal computer, the CPU is fastest, but this is not necessarily a general result. The GPU is the slowest for a small number of panels, which is due to the transfer time of data between the standard computer memory and the GPU memory. That is, the GPU is clearly not suitable for a small number of panels, but it is comparable to the CPU for a large number of panels.

The kernels written for this project are responsible for calculating the influence of a panel on a given point. That is, the influence matrices can be calculated in parallel, along with the induced velocities. The influence of a single panel on a certain point is completely independent of all the other panels in the simulation, which makes the tasks of calculating induced velocities and influence matrices inherently parallel. That is, no modification to the algorithm has to be done in order to execute it in parallel. It can be written in the same way as for a normal serial execution, only when executed in OpenCL it will happen much faster as it utilizes every computational core available.

Much more could be said about OpenCL, but this is a bit outside the scope of this project, so the author stops here.

### **3.2.3 Blender**

Blender is an free open source 3D software, developed by the Blender foundation [39]. It aims at being a complete tool for everything 3D: modeling, rendering, compositing, animation, special effects, etc. It also contains many simulation features, such as rigid body simulation, cloth simulation, smoke and fire simulation, fluid simulations, a complete game engine, and much more. All of which is intended for making visually realistic images and animations, not provide accurate scientific results. The goal of using Blender in this project was to see if Blender also could act as a tool to create and manage the necessary geometry for the simulations and take care of the visual post processing after the simulations are done. The hypothesis is that many of the necessary features for making visually realistic images can be used in scientific simulation as well.

The interface of Blender, together with the custom BEM code can be seen in figure 3.7. The author has found that Blender provided a good way of dealing with the geometry of the simulation. Many geometric values, such as normals and panel area, is automatically calculated by Blender. Advanced geometry models, such as Non-uniform rational B-spline (NURBS), can be used to create 3D models where the density of the panels easily can be changed. This is a practical feature, for instance when doing convergence analysis.

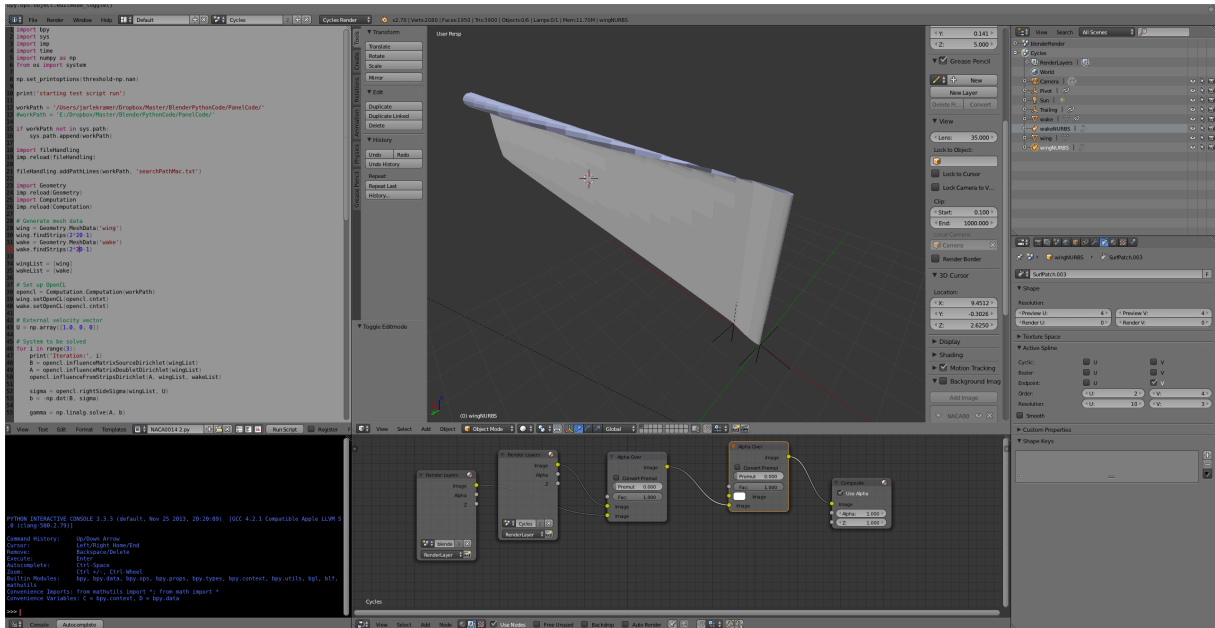


Figure 3.7: The interface used when running BEM simulations in Blender

The BEM code must be executed inside Blender in order to work. Since there are a certain demand of structure in the geometry input, due to the "strip nature" of the simulation, there are some demands on what type of geometry to be used. The author decided that the NURBS objects in Blender was suitable for this purpose, so that all wings and wakes must be constructed using NURBS as a basis.

### 3.3 Verification of BEM code

This section presents some verification experiments performed by the author in order to test whether the code was simulating flow over wings properly. In particular, the pressure distribution for "infinite" wings and cylinders are compared to the 2D versions, either based on theoretical values, or calculated by XFOIL. In addition, the lift and drag calculated by the BEM code is compared to experiments and theory. There is no way to find the potential lift-induced drag purely from experiments, but the lift from a potential code should correspond to experiments for small angles of attack. The conclusion from these tests are that the BEM code is implemented correctly. The pressure distribution around both foils and cylinders seems correct, with slight errors for large pressure peaks. This behavior is believed to be due to the relatively low order of the BEM code, rather than an implementation error. Even if there are some problems with large pressure peaks, the lift and drag seems to fit very well with both experiments and theory. It is particularly important that the lift values are correct, which seems to be the case.

#### 3.3.1 Near-Infinite Cylinder

The results from the BEM code, used to simulate a *near-infinite* cylinder is compared to theoretical values for 2D, or infinitely long, cylinders. The length of the actual cylinder used in the simulation was determined by increasing the length until there was no visible changes in the pressure values close to the center of the cylinder. See figure 3.8 for a visual explanation of the geometry used.

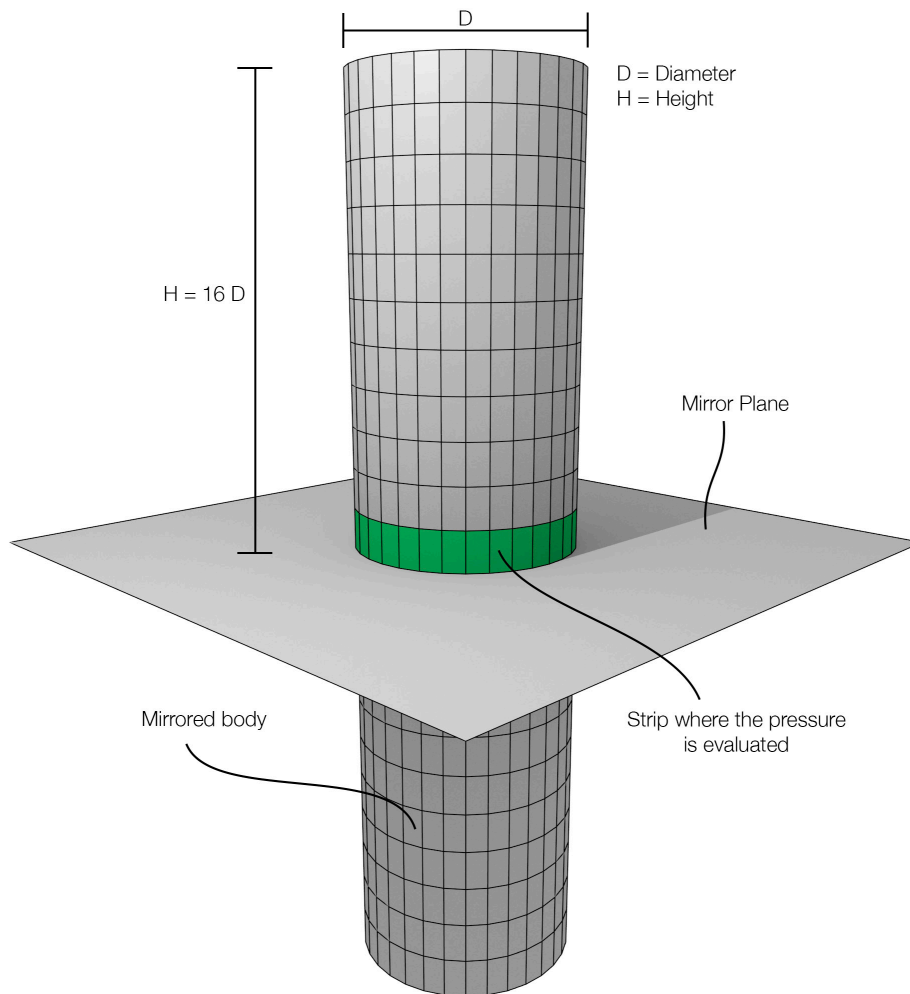


Figure 3.8: Geometry of the cylinder used in the verification test

The pressure was evaluated at the strip in the horizontal plane that is closest to the mirror plane. These pressure values should be least effected by the finite length of the cylinder. The theoretical values for the pressure coefficient for a 2D cylinder is given as follows:

$$C_p = 1 - 4 \sin(\theta) \quad (3.42)$$

This equation, compared to the data from the BEM code can be seen in figure 3.9

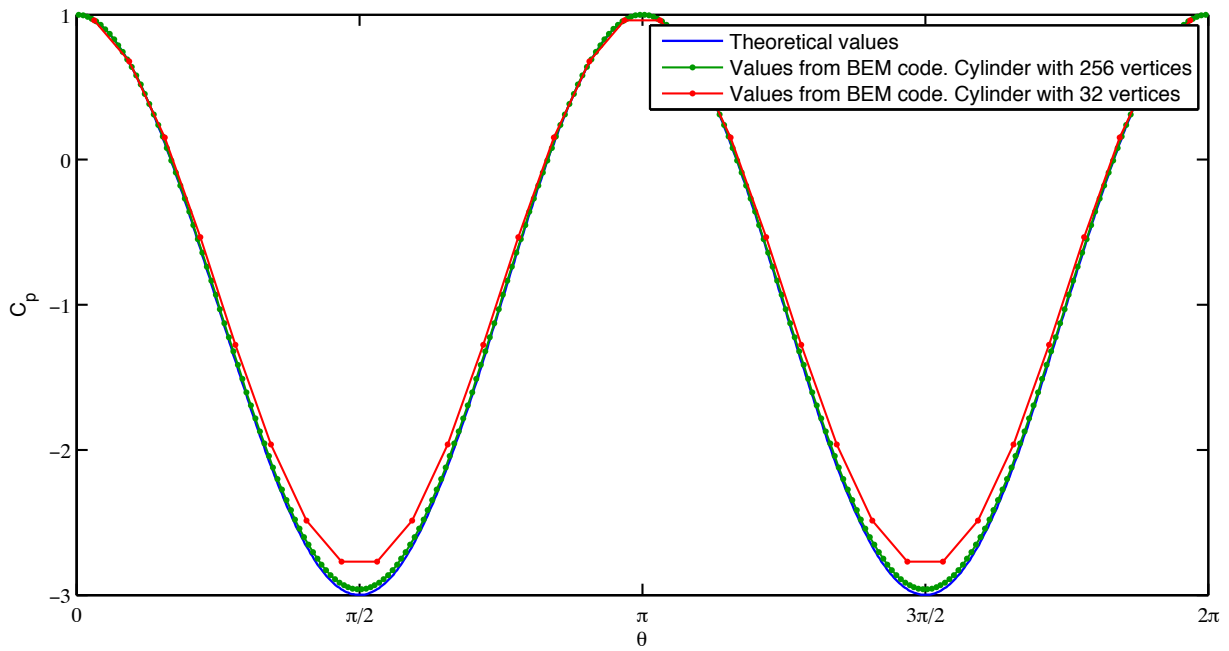


Figure 3.9: Pressure coefficient for a near-infinite cylinder. Theoretical and numerical values

### 3.3.2 Near-Infinite Wing

Same approach as for the "infinite" cylinder were used on wing profiles. One symmetric foil with and angle of attack, and one cambered foil is tested. The cambered foil, NASA LS - 0417 have two pressure distributions: one at a strip close to the symmetry plane, strip 0, and one at the end of the wing, strip 36. It is seen that strip 36 have significantly different pressure distribution compared to the 2D values, but this is to be expected. This is in fact the reason for reduction in lift for a 3D wing. The pressure distribution close to the symmetry plane fits well with the 2D version from XFOIL

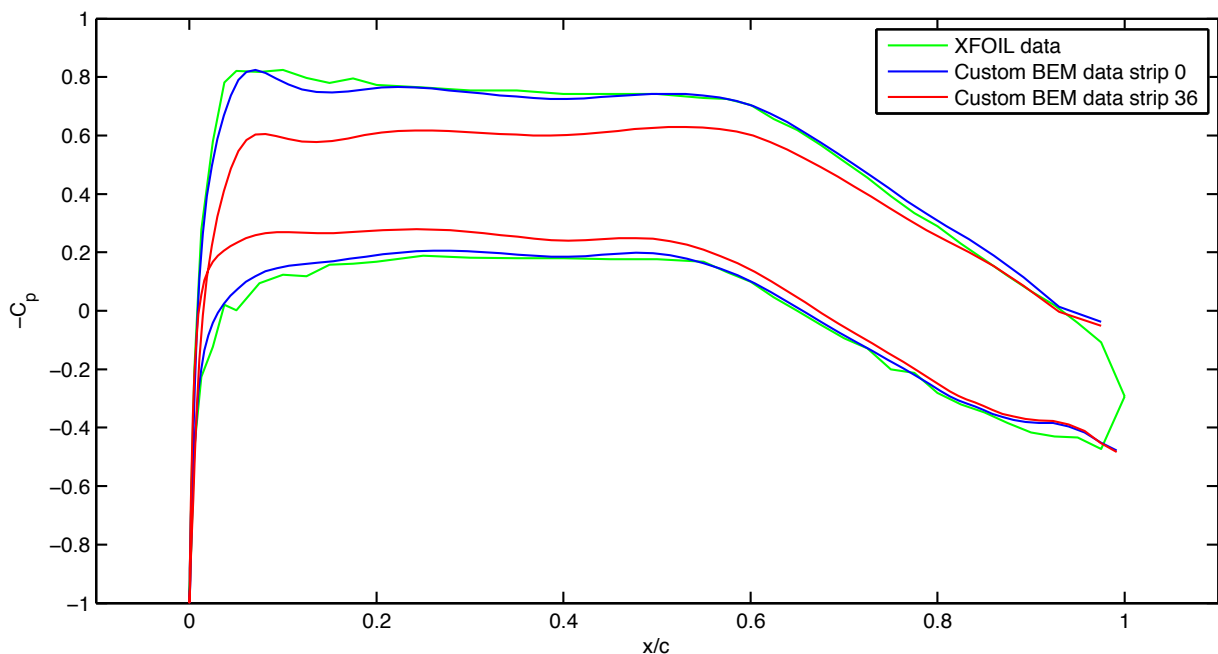


Figure 3.10: Pressure coefficient for a near-infinite wing with foil profile NASA LS - 0417

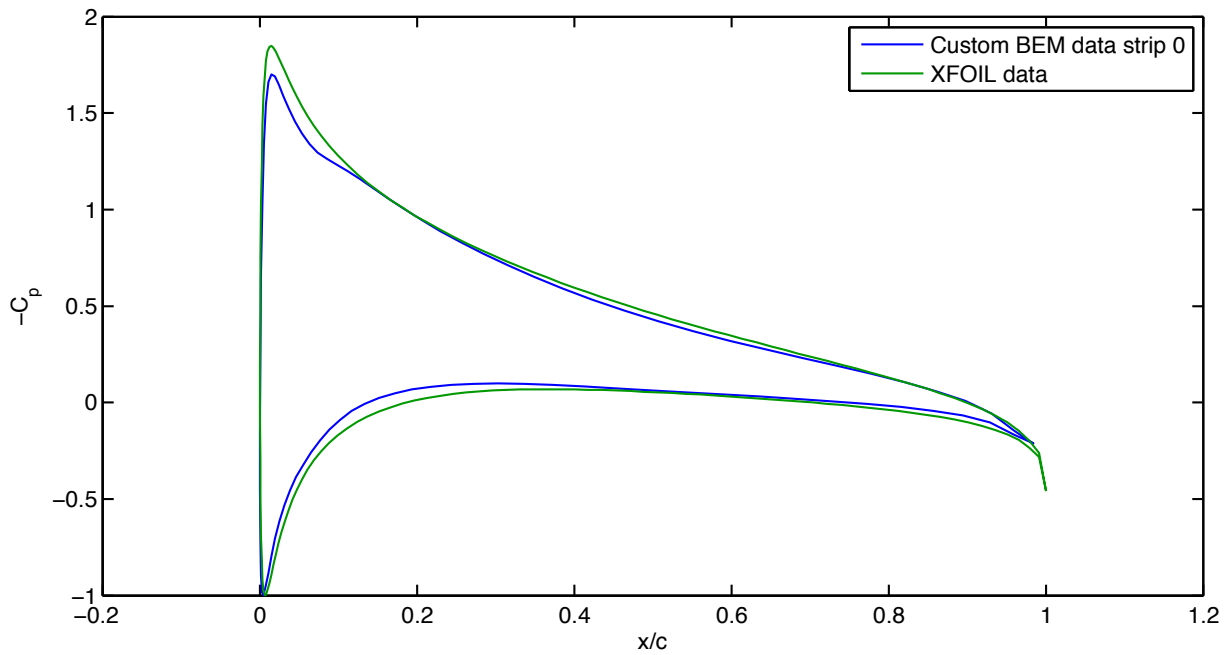


Figure 3.11: Pressure coefficient for a near-infinite wing with foil profile NACA0014 at 10 deg angle of attack

### 3.3.3 Validation of Lift by Comparison with Experimental Data

In order to test the lift, experimental data, based on experiments performed by Prandtl, for rectangular wings with different aspect ratio, were used. The data can be found in reference [32]. The data in reference [32] were originally for a cambered wing. In order to compare the data in reference [32] with a symmetric wing at an angle of attack, the lift at zero angle of attack was subtracted from the data. Two aspect ratios are tested,  $Asp = 3$  and  $6$ . Both aspect ratios seems to fit well with the experimental data, however, in the process of performing these experiments it was discovered that the wing with  $Asp = 6$  needed more strips to simulate accurate values than the wing with  $Asp = 3$ . This is expected to be due to the fact that the largest change in spanwise circulation is happening at the end of the wing, so that it is necessary to keep the "strip density" rather high at the end of the wing. It is possible to have more strips at the end of the wing, then close to the symmetry plane, but the tests performed in this experiment were done with strips that were equal in size. The number of strips are 15 for the  $Asp = 3$  wing, while it is 40 for the  $Asp = 6$  wing. The number of panels in each strip were 36 for both cases.

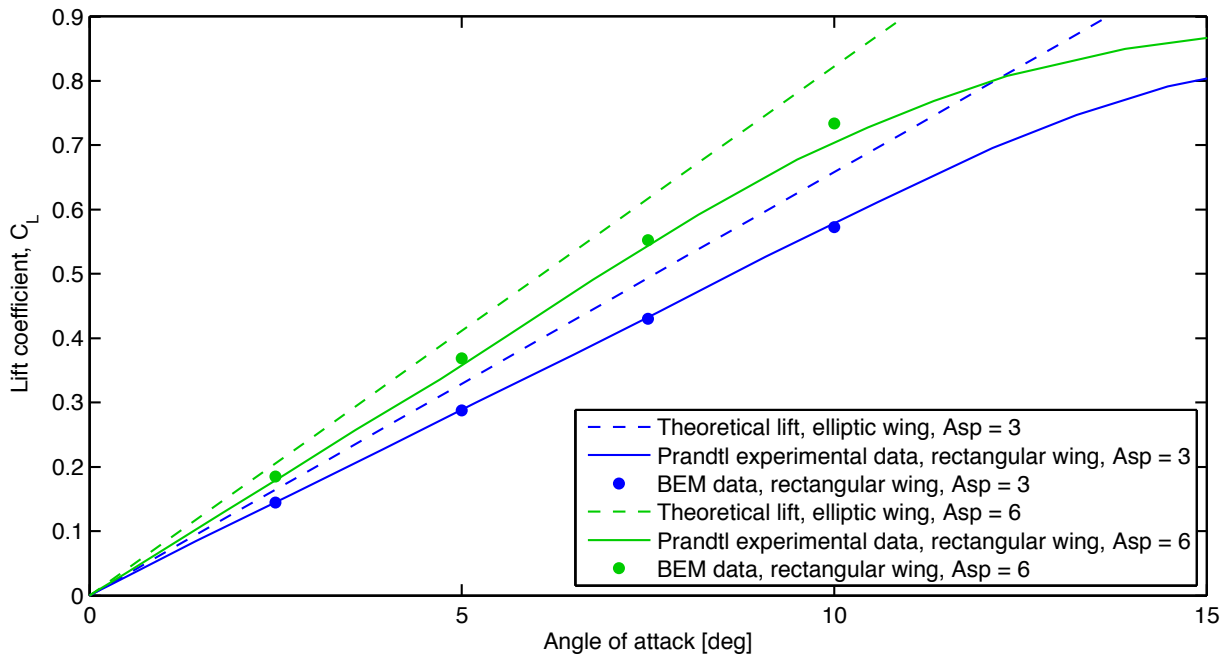


Figure 3.12: Lift coefficient for rectangular wings with different Asp. Data from BEM and source [31]

### 3.3.4 Validation of Lift-Induced Drag

Lift-induced drag can not be tested experimentally, however, reference [34] presents theoretical values for rectangular wings. The theoretical values are compared to the result from the BEM code and can be seen in figure 3.13

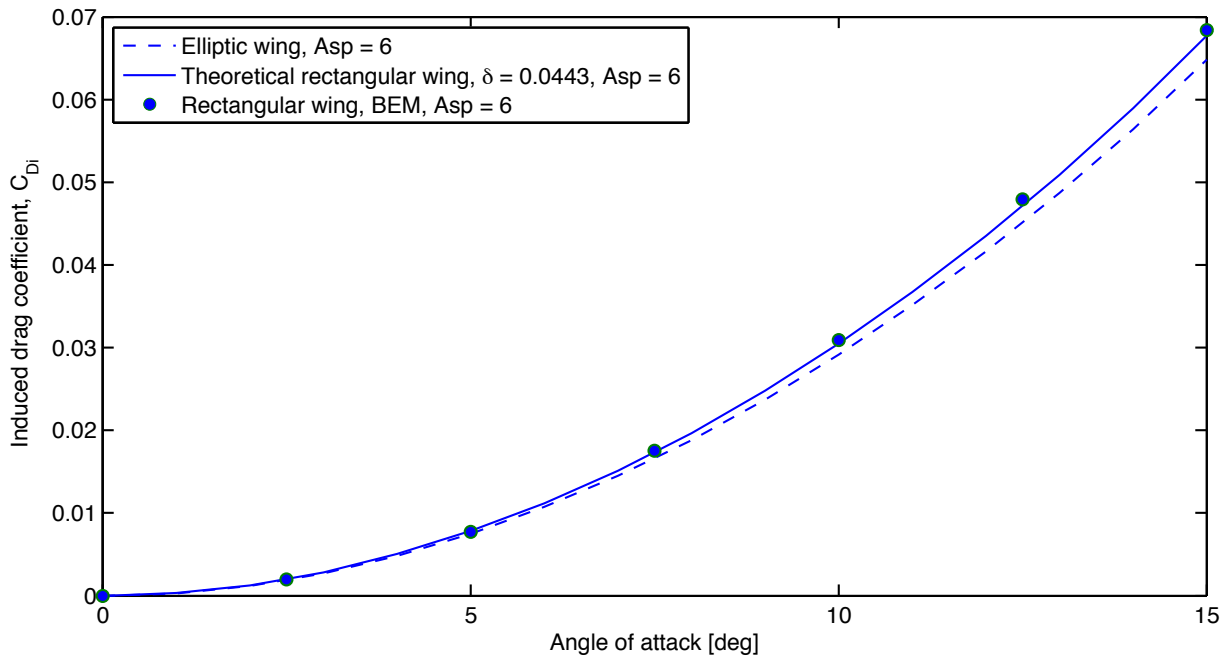


Figure 3.13: Induced drag coefficient for rectangular wing with Asp = 6, compared with theoretical values from source [34]



## Chapter 4

# CFD using STAR CCM+

Computational Fluid Dynamics (CFD) was used as a modeling tool in this project in order to capture the effect of letting a ship go with a yaw angle. This was necessary, as simpler methods for predicting ship resistance don't include yaw effects. Ship hulls have very low aspect ratio, in addition to being thick bodies, so that viscous effects may play a significant role for both the added resistance due to lift, and the lift forces them self. Potential theory is therefore probably not suitable for simulating the lifting properties of a ship hull.

The commercial software STAR CCM+, made by the company CD-Adapco, is used for the simulation. The goal was to find a setup that allowed relatively fast simulation, but that still captured the important effects of interest. In an attempt to accomplish this, several articles were studied. In particular, the articles presented in the "Gothenburg 2010 CFD Workshop" was used extensively, along with tutorials made by CD-Adapco that specifically give guidance on how to use STAR CCM+ for predicting ship resistance. The Gothenburg 2010 CFD workshop included several institutions and companies doing the same type of CFD simulation on ships. The predictions from CFD was then compared with Experimental Fluid Dynamics Data (EFD Data). An overview of the results for ship resistance from the Gothenburg 2010 CFD workshop can be found in reference [23]. Based on the statistics presented in reference [23], the average error for ship resistance predictions using CFD was 2% of the EFD Data, with a standard deviation of 1.3 %. Although many of the simulations performed had quite a high number of grid points (the highest number for simulations using the same turbulence model as in this project was around 9 million grid points), reference [10] suggest that a lower number of grid points can achieve accurate results as well, if care is taken regarding where to distribute the grid points. Reference [10] achieved an error of just 0.48% using "only" 1.2 million cells. However, all of the simulations performed in the Gothenburg 2010 CFD workshop was for ships going without yaw-angle.

In order to find a specific simulation setup to work with, one of the tutorials regarding marine ship resistance prediction (no reference for this tutorial, as it is located in a closed "customer portal", that only is available for people with license for STAR CCM+, but the tutorial seems to be based on reference [10], as this article is written by CD-Adapco employees, and covers the exact same topic) from CD-Adapco was used as a starting point, before certain parameters was changed based on either suggestions from different articles, or based on convergence analysis done by the author.

In the beginning of the project, only the chemical tanker was considered as a test ship, and this model was the only one that was simulated with CFD. At the very end of the project, EFD Data, including yaw effects, was discovered for the Series 60 ship model. This data was useful, not only as a second set of force data for a ship going with a yaw angle, but as an opportunity to validate the CFD setup against experimental data. The same "logic" that had been used to simulate the chemical tanker was applied to the series 60 model. That is, length and time scales in the simulation setup was scaled from the chemical tanker model to the scale of the series 60 model, before some test simulations was performed for different speeds and different yaw angles.

This revealed a problem. For zero yaw angles, the result from the CFD test matches the results from the EFD Data, but for large yaw angles, the results differs quite a lot. The CFD predicts too high resistance, and too low side force for a ship going with a yaw angle. As this discovery was made at a rather late point in the project, it was not possible to fix it due to time issues. Some discussion will be made regarding why this

error might occur, however, the author has not had the opportunity to test this theory properly. Only some early theories were tested, such as varying the number of cells, changing the turbulence model, and altering the time step, but non of these changes affected the result from the CFD simulation. The next step would be to change the domain size, and the author have a theory of why this might be the error. See section 4.5 for more on this.

Based on the final results, it seems that the forces predicted by CFD does behave somewhat correctly (non-linear lift from the hull, non-linear lift induced drag), but the result for high yaw angles are probably a bit off. They are still used in the final model in order to predict the importance of hydrodynamic effects from a wing sail, but the results should be considered as a rough approximation, not highly accurate force values. If the potential error in the results for the chemical tanker matches the nature of the error seen for Series 60, the importance of hydrodynamic effects should be overestimated. That is, since the resistance might be too high, and the lift too low, the result from the final model might overestimate the necessary yaw angle, and added resistance due to yaw. As can be seen in the final result chapter, chapter 8, this error can be considered "conservative". Higher lift and lower drag should not alter the final conclusion, only strengthen it.

This chapter will go through the general theory that the CFD code uses to simulate fluid flow around a ship hull, give an overview of the simulation setup used, along with the argument behind them, before presenting the final raw results from the CFD simulation, where the result from the Series 60 model is compared to the EFD Data.

## 4.1 Theory

---

This section will go through the theory behind the CFD simulation done in this project. As STAR CCM+ is a commercial software, using fairly standard CFD methods, and the author had nothing to do with the development of this code, the theory section will be brief.

STAR CCM+ uses the Finite Volume Method (FVM) to discretize and solve the integral formulation of the Navier-Stokes equations. That is, the computational domain is divided into many cells, or volumes, and the integral formulation of the Navier-Stokes equation is approximated over each cell in a discrete manner. Points on the cell faces is used to approximate surface integrals, while points in the middle of the cell are used to approximate volume integrals. The unsteady integral formulation for the Navier-Stokes equation, assuming incompressible, Newtonian-fluid flow, can be written as follows:

$$\iint_S \mathbf{U} \cdot \mathbf{n} dS = 0 \quad (4.1)$$

$$\frac{\partial}{\partial t} \iiint_V \mathbf{U} dV + \iint_S (\mathbf{U} \cdot \mathbf{n}) \mathbf{U} dS = -\frac{1}{\rho} \iint_S p \mathbf{n} dS + \iiint_V \mathbf{g} dV + \nu \iint_S \nabla \mathbf{U} \cdot \mathbf{n} dS \quad (4.2)$$

$\mathbf{U}$  is the velocity vector,  $\mathbf{n}$  is the normal vector to the surface,  $\mathbf{g}$  is the acceleration of gravity vector,  $\nu$  is the kinematic viscosity,  $\rho$  is the density and  $p$  is the pressure. These are the equations to be solved by STAR CCM+. The first one is the conservation of mass equation, while the second one is the momentum equation. When these equations are discretized, a non-linear, unsteady, system of equations is the result. The non-linearity and the velocity-pressure coupling must be dealt with, and there exist several methods for doing this. STAR CCM+ have several options. For this project the following has been used:

The equations are solved using a *segregated*, or *decoupled*, flow solver. That is, each equation are solved separately, and coupled together using whats called a predictor-corrector approach. The predictor-corrector approach used specifically is the standard method called Semi-Implicit Method for Pressure Linked Equations (SIMPLE). One of the key general principles of the SIMPLE method is to construct a "pressure correction equation". This can be done by taking the divergence of the momentum equation. For more details on this, see reference [13]. The overall steps for the SIMPLE method is as follows:

1. Set boundary conditions
2. Compute the gradients of velocity and pressure
3. Solve the discretized momentum equation in order to find a new velocity field
4. Compute mass fluxes at the faces of each cell
5. Solve the pressure correction equation, in order to find  $p'$
6. Update the pressure field,  $p^{n+1} = p^n + \omega p'$ , where  $\omega$  is an under relaxation factor
7. Update the boundary pressure corrections
8. Correct the mass fluxes
9. Correct the velocity field
10. start over, until solution are found

The number of iterations that are performed for each time step, called the inner iterations, can be adjusted by the user. For this project, the number was set to 10, based on recommendations from the ship resistance tutorial to CD-Adapco.

Time stepping must also be performed. This happens with a first order implicit method in this project. The reason for using an unsteady solver in the first place, rather than a steady solver, is that the simulation includes a free surface. The shape of the free surface is not known a priori and must be found, and the only way to do this in STAR CCM+ is by using an unsteady simulation. That is, there is no way to enforce the *radiation* condition, the demand that the waves must travel in the correct direction, using a steady solver. Implicit time-stepping is known to be a good approach when one are actually searching for a steady state solution, as the final result from an implicit time-stepping method is less sensitive to the size of the time step [13].

The presence of the free surface is treated with whats called the Volume Of Fluid (VOF) method. The VOF method is conceptually rather simple. The entire fluid domain is discretized, and each cell is given fluid properties, such as density and viscosity, based on whats called a *volume fraction*. If two types of fluids are simulated in the same domain, the volume fraction is just a number, giving the ratio of how much fluid there is of one type. For instance, the volume fraction,  $C$ , could give the volume ratio of water divided by air. Then the density of that cell would be  $\rho = C \cdot \rho_{\text{water}} + (1/C) \cdot \rho_{\text{air}}$ . The viscosity, and any other fluid properties, will be calculated in the same manner. The initial values of  $C$  must be set in the entire fluid domain. Then, as the simulation progresses, the change of  $C$  in a cell will be calculated according to a transport equation. The VOF method is recommended to be used for marine simulations by CD-Adapco. A down side of the method is that changes from water to air happens somewhat gradually. In order to have a "sharp" interface between water and air, very small cells has to be used in the region of the free surface. See section 4.2.3 for more on this.

The last thing that needs to be simulated is turbulence. For this project, it is done with Reynolds-Averaged Navier-Stokes (RANS) equations. The Navier-Stokes equations are averaged over a hypothetical turbulence-time-scale, which results in an extra term in the Navier-Stokes equation that is known as Reynolds stress. This Reynolds stress term can be modeled based on physical principles, and some empirical coefficients. The specific turbulence model used for this project is the "Realizable two-layer k-epsilon model". The k-epsilon model is perhaps the most famous turbulence model. It was chosen, as this was the most common turbulence model used for predicting ship resistance in the Gothenburg 2010 CFD workshop, based on reference [23]. The "realizable" part of the turbulence model is referring to the fact that there is a correction in the k-epsilon model in order to limit unphysical large growth of turbulent kinetic energy, which are known to be a problem, according to the STAR CCM+ documentation. The "two-layer" part of the turbulence model is referring the fact that regions close to a wall is treated a bit differently than regions far away.

## 4.2 Parameters Used

---

This section will go through the parameters used in the simulation setup. In particular, the ship model, the size of the domain, the mesh design, and the time step will be discussed. A simulation model in STAR CCM+ involves many more parameters than this. The parameters not mentioned was set according to the tutorial made by CD-Adapco, or just kept at default values. For instance, there are several empirical parameters that can be specified for the turbulence model. These parameters were not touched by the author, because, in general, that should not be necessary. STAR CCM+ also include a lot of parameters that affect how the non-linear solver works. These parameters should mostly affect the speed of the simulation, not the results, and it is expected that the default values are close to the appropriate values. The parameters presented in this section is considered to be the most important.

### 4.2.1 CFD Ship Model

The ship models used in the simulations were in model scale. Theoretically, it is possible to do full scale tests with CFD, but in general, this would take longer time as there would have to be more cells in the fluid domain. It is known that small turbulent structures can have an effect on the large scale flow. The smallest turbulent flow structures is dependent on the Reynolds number of the flow, and the higher the Reynolds number, the smaller will the smallest turbulent length scale be (see for instance reference [38], chapter 1, for a discussion about turbulent length scales). Even though RANS is used to model turbulence in this project, the general tendency will still be that higher Reynolds number demands more cells in order to give accurate results. So in order to lower the cell count in the simulation, ships in model scale was tested. There exist methods for scaling the results to full scale from model scale, developed for towing tank test, and the same methods will be used on the CFD results. See section 5.4 for more on this.

The dimensions of the CFD models can be seen in table 4.1. The Series 60 model was scaled to the same scale as the model used in the experiments that generated the EFD Data, so that direct comparison of the result were possible. The chemical tanker was scaled such that the width of the model is 1 m, because this gave a "normal" model scale, and 1 m is a round nice number.

Table 4.1: Data for the CFD ship models, at model scale

	Chemical tanker model	Series 60 model
scale	1:29.5	1:40
$L_{wl}$ [m]	5.765	3.048
$B_{wl}$ [m]	1	0.406
$D$ [m]	0.339	0.163
$\nabla$ [m <sup>3</sup> ]	1.486	0.121
$S$ [m <sup>2</sup> ]	7.885	1.579
$U_m$ [m/s]	1.42	1.22
$\rho$ [kg/m <sup>3</sup> ]	997.561	997.561
$\nu$ [m <sup>2</sup> /s]	8.91E-07	8.91E-07
Re [-]	9.19E+06	4.17E+06

### 4.2.2 Domain Size

The domain size for the calculation was determined based on average results from the Gothenburg 2010 CFD workshop. This was done as an attempt to reduce the number of convergence tests. The exact mesh design is often hard to quantify when reading articles about CFD, as the cells might be distributed throughout the domain in a complex matter. Domain size on the other hand is often very easy to quantify. The external shape of the simulation domain are usually simple geometries, such as cylinders, cubes or spheres and the dimensions are usually given. This makes it easy to copy the general domain design used by others, and

the author thought this was an excellent opportunity to make a shortcut in finding an appropriate simulation setup. This was most likely a mistake. The general domain dimensions will be given in this section, and it is expected that these domain dimensions is suitable for ship simulation without yaw angle. However, it is most likely not suitable for a simulation that includes yaw angles, at least not large ones. See section 4.5 for more discussion about this.

The articles covering ship resistance prediction from the Gothenburg 2010 CFD workshop were systematically read through, and simulation domain length-scales were stored, whenever they were given explicitly. This should give an overview of "normal" domain designs, when it comes to ship resistance prediction. The result of this analysis can be seen in table 4.2. The external geometry of the domain is assumed to be box-shaped, as seen in figure 4.1. The final values used in this project is also stored in table 4.2.

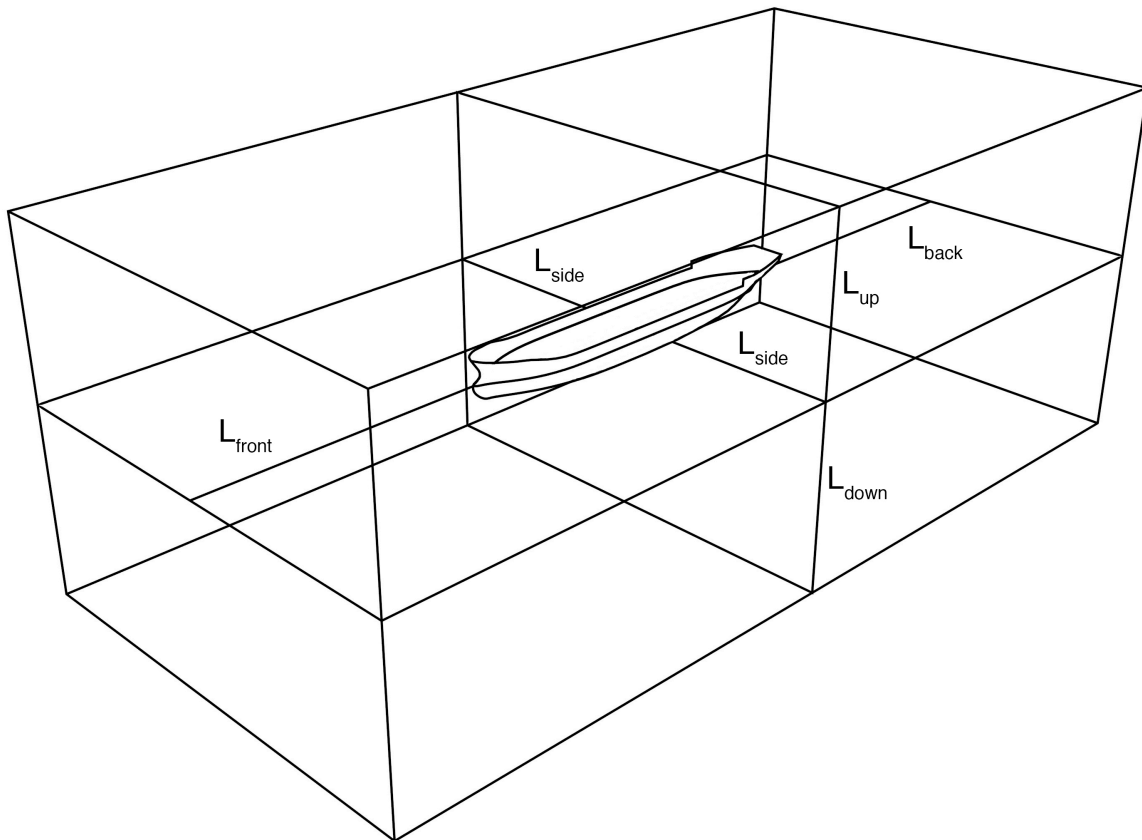


Figure 4.1: Illustration of domain dimensions

Table 4.2: Simulation domain length scales, reference values and values used in this project. All values are divided by the ship length used in the corresponding simulation

Article name [reference number]	$L_{front}$	$L_{back}$	$L_{side}$	$L_{down}$	$L_{up}$
Simulation of Flow Around KCS-Hull [10]	2.47	2.47	2.47	2.47	1.24
URANS Computations of a DTMB 5415 [17]	1	3	2	1.5	0.75
Verification and Validation for Unsteady Length Computation [5]	1	2.3	1.8	1.5	0.5
Viscous free surface calculations for the KCS hull [29]	2	3.5	1	1	0.1
CFD Simulation Of KCS Sailing in Regular Head Waves [36]	0.9	2.5	1.5	1.5	-
Prediction of Ship Resistance and Propulsion Performance Using Multi-Block Structural Grid [49]	1	3	1	0.87	0.028
This project	2	2.5	2	2	1

### 4.2.3 Mesh

The number of mesh cells were decided based on convergence analysis. STAR CCM+ have a several built in mesh generators. For this project, the "trimmed" mesh generator are used along with "prism layers". The trimmed mesher is a mesh algorithm that generates mostly hexahedral cells (polyhedrons with six faces, such as a cube, parallelepiped, rhombohedron, etc). The prism layer is a way to generate orthogonal prismatic cells next to wall surfaces or boundaries. This makes it so that the mesh follows the curvature of the ship hull closely in layers. The number and thickness of the prism layers can be adjusted by the user. 6 prism layers were used in this project, due to recommendations from the tutorial made by CD-Adapco, based on reference [10].

The local size of mesh cells can be set by the user, as a percentage of a mesh "base size". That is, there is a length scale that determines the general size of each mesh cell, but based on local refinement, the local cell size can vary. Each boundary surface have a custom cell size, along with a few "volumetric controls". Volumetric controls are a way of telling STAR CCM+ to adjust the cell size locally inside a volume that is specified by the user. For this project, volumetric controls were used to increase the cell density close to the free surface, behind the ship in the kelvin-wake, and close to the bow and stern of the ship. These refinements were based on the authors guess that there would be large changes in the flow in these areas. In general, areas with large gradients of velocity and pressure should have more cells than areas where the flow is changing very slowly. The bow and the stern of the ship will create changes in the flow due to large changes in the ship geometry. The kelvin-wake behind the ship will experience large changes in velocity due to the ship-generated waves. The free surface must have a high cell density in order to model a sharp transition from water to air, which is a necessity due to the use of the VOF method.

The convergence analysis were then done in order to find an appropriate value of the base size. Adjusting the base size will adjust all local cell sizes, as every local refinement is given as a percentage of the base size. The local refinement, as percentage of the base size, can be viewed in table 4.3. The free surface have three layers of local refinement, in order to have very fine control over the mesh in this region. The cell size is only adjusted in the z-direction, as the refinement is only there to give smooth transition between water and air in the vertical direction.

Table 4.3: Local refinement for special areas in the simulation domain

Area name	Local refinement in percentage of base size
External boundaries	1600
Ship hull	25
Ship deck	100
Free surface, fine	12.5 in z-direction only
Free surface, medium	25 in z-direction only
Free surface, coarse	50 in z-direction only
Bow and stern	25
Kelvin wake	100

The convergence analysis where then done for two yaw angles: 0 degrees and 8 degrees. Based on this, it was determined that the chemical tanker should have a base size of 0.1 m. The base size for the Series 60 is scaled according to the length of the ship, so the base size is 0.05. Figure shows the mesh for the chemical tanker, with 6 degree yaw angle, from the top, side and front respectively.

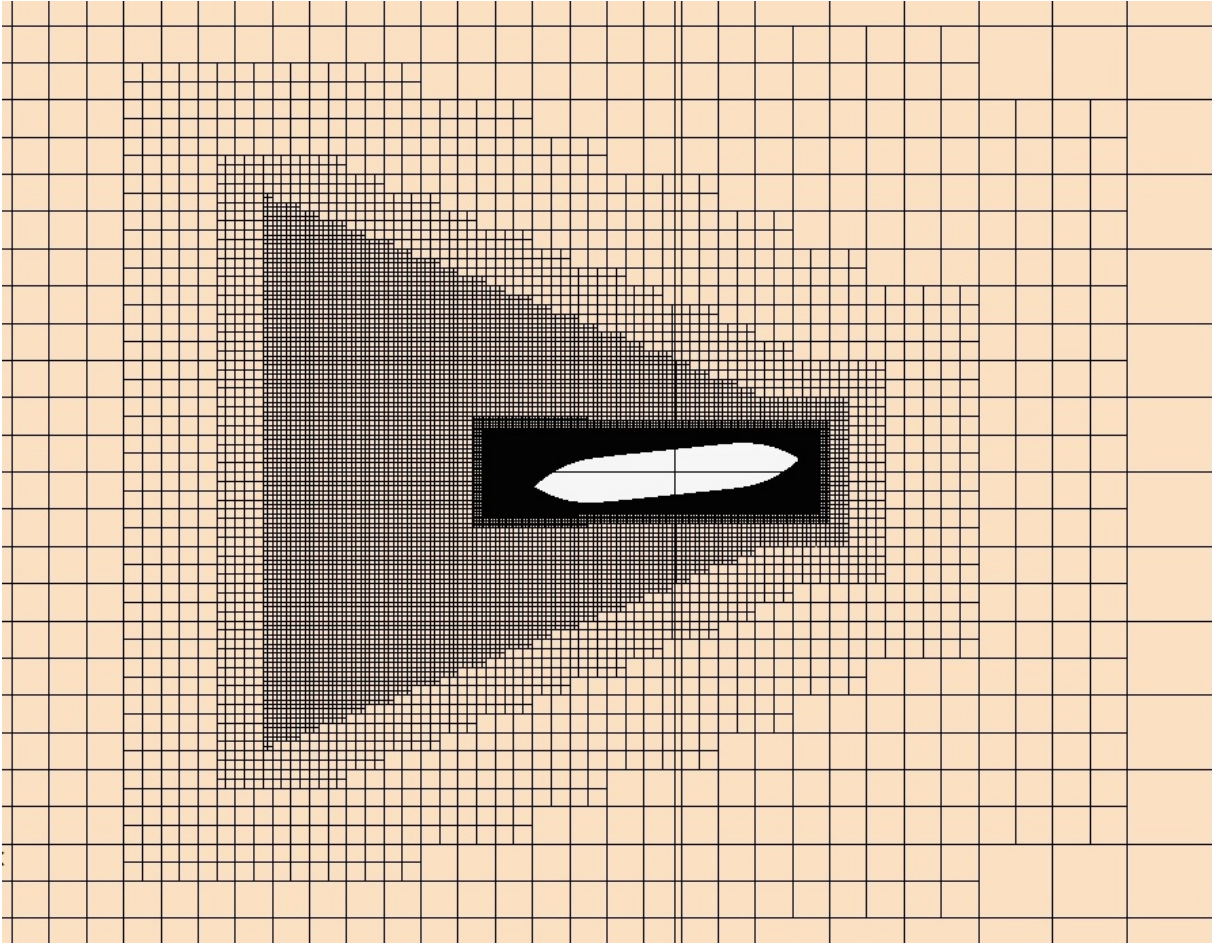


Figure 4.2: The mesh used to simulate the chemical tanker at 6 deg yaw angle, viewed from the top

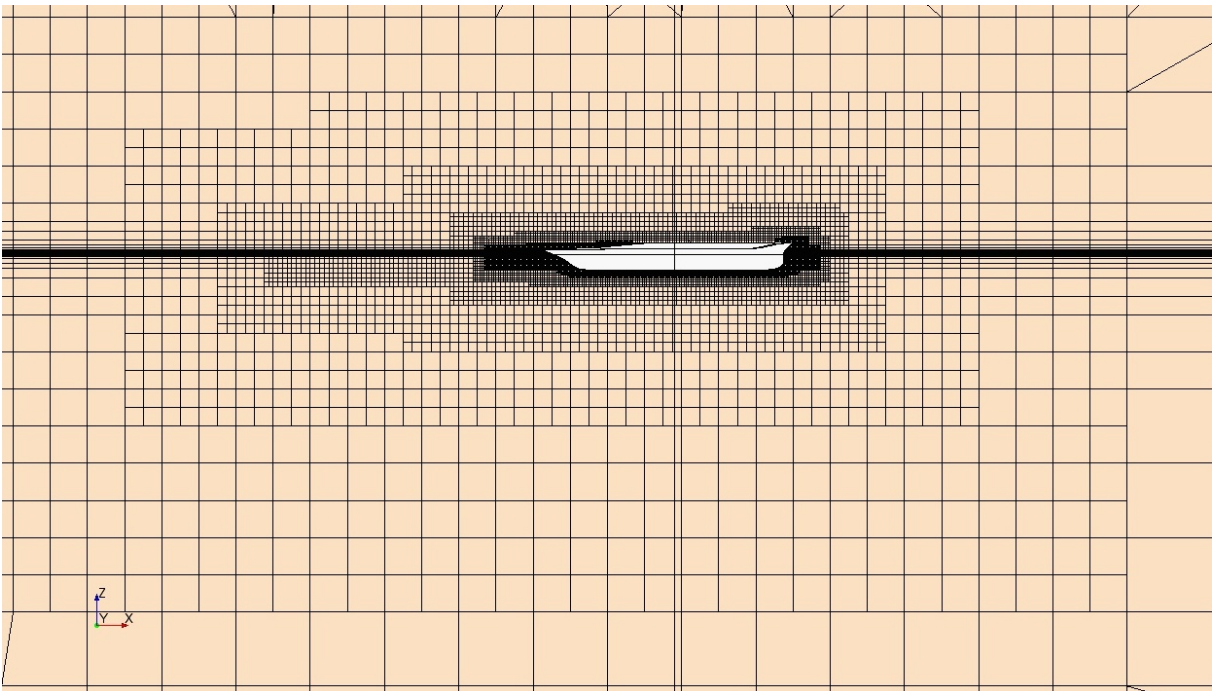


Figure 4.3: The mesh used to simulate the chemical tanker at 6 deg yaw angle, viewed from the side

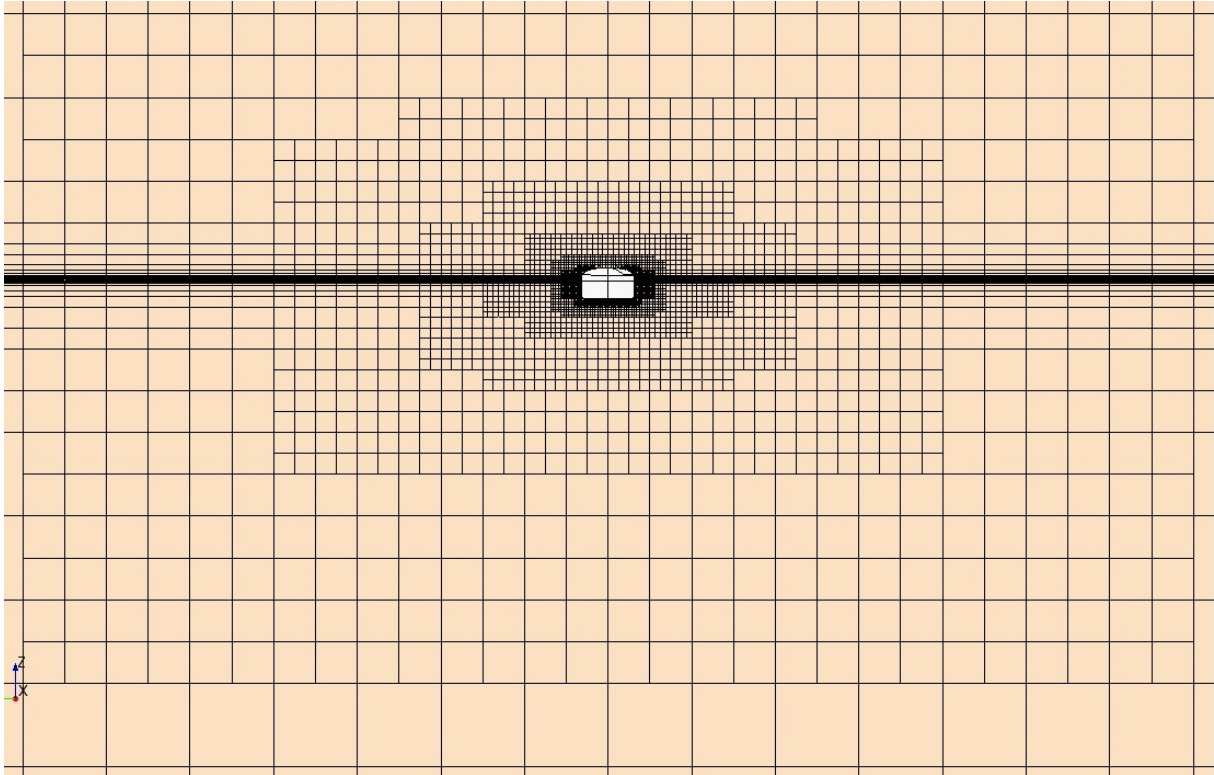


Figure 4.4: The mesh used to simulate the chemical tanker at 6 deg yaw angle, viewed from the front

The results from the convergence test can be seen in table 4.4 and 4.5. 8 degrees yaw angle were used as the main convergence test, as this was considered to be the "worst" case, numerically. This was the highest yaw angle to be simulated. The reason for using time step equal to 0.01 for the mesh convergence test, was that the importance of the time step was already known at this point. Before performing these tests, another set of tests were performed with a much higher time step (0.05 s). This clearly led to weird behaviour, so the time step was revised, and the mesh convergence test were performed all over again. See section 4.2.4 for more on this. The results presented in this report are from the "usable" convergence test, as the first one failed to give any insight into the importance of the mesh size, due to time step problems. The "Medium" mesh gives results that are very close to the "Very Fine" and the "Extremely Fine" mesh for both 8 degrees yaw and 0 degrees yaw. The "Medium" drag result is 98.9% of the "Extremely Fine" drag result for yaw angles equal to 8 degrees, and the side force of the "Medium" result is 97.9%. It might seem weird that the result for the "Fine" mesh is further from the "Extremely Fine" result than the "Medium" mesh, but this is normal, according to reference [10]. When local refinements are used in the mesh design, as is the case for this project, this type of behavior can happen.



Table 4.4: Results from mesh convergence test with yaw angle equal to 8 degrees

Grid Name	Coarse	Medium	Fine	Very Fine	Extremely Fine
Yaw Angle [deg]	8	8	8	8	8
Speed full scale [knots]	15	15	15	15	15
Speed model scale [m/s]	1.42	1.42	1.42	1.42	1.42
Base size [m]	0.125	0.1	0.08	0.064	0.0512
Number of Cells [-]	709444	1071243	1772343	2905080	5020656
Time step [s]	0.01	0.01	0.01	0.01	0.01
Drag Total [N]	42.837	42.687	44.541	42.861	43.170
Drag Pressure [N]	23.583	23.589	23.621	23.967	24.202
Drag Shear [N]	19.254	19.099	20.921	18.895	18.968
Side Force Total [N]	68.075	69.279	72.738	69.820	69.508
Side Force Pressure [N]	69.674	70.800	73.459	71.214	70,850
Side Force Shear [N]	-1.599	-1.521	-0.721	-1.394	-1.342

When simulating a ship without yaw, only half the ship can be simulated. Symmetry boundary condition is applied at the center line of the ship, which effectively models the entire ship, with half the number of cells. This is the reason for the small number of cells in table 4.5

Table 4.5: Results from mesh convergence test without yaw angle

Grid Name	Coarse	Medium	Very Fine
Yaw Angle [deg]	0	0	0
Speed full scale [knots]	15	15	15
Speed model scale [m/s]	1.42	1.42	1.42
Base size [m]	0.125	0.1	0.064
Number of Cells [-]	284013	425703	984217
Time step [s]	0.01	0.01	0.01
Drag Total [N]	36.133	30.552	30.635
Drag Pressure [N]	24.056	22.532	22,813
Drag Shear [N]	12.076	8.021	7,822

#### 4.2.4 Time stepping

The time step was discovered to be an important parameter for simulating flow around a ship hull. As the point of these simulations is to find a steady state solution, the time step might seem like a secondary parameter. As the author was reading about CFD in articles, this also seemed to be the case. Time steps were only given, without any convergence tests, while convergence tests for the mesh was always given. The number of cells seems to be a more important factor. At first, the time step of the simulation was set to be 0.05, which was decided based on reference [10], where the time step is 0.04. This was for a slightly larger model, with similar Froude number, so the time step might be correct, but no convergence test for the time step are presented.

When doing convergence test for the mesh size with the first time step, it was discovered that the waves on the free surface disappeared for small base sizes. This seemed very unphysical, so a time step test was initiated, before further mesh testing were done. The result from the time step convergence test can be seen in figure 4.5. It is done without yaw angle, as the importance of the time step is believed to be connected to the waves on the free surface, primarily.

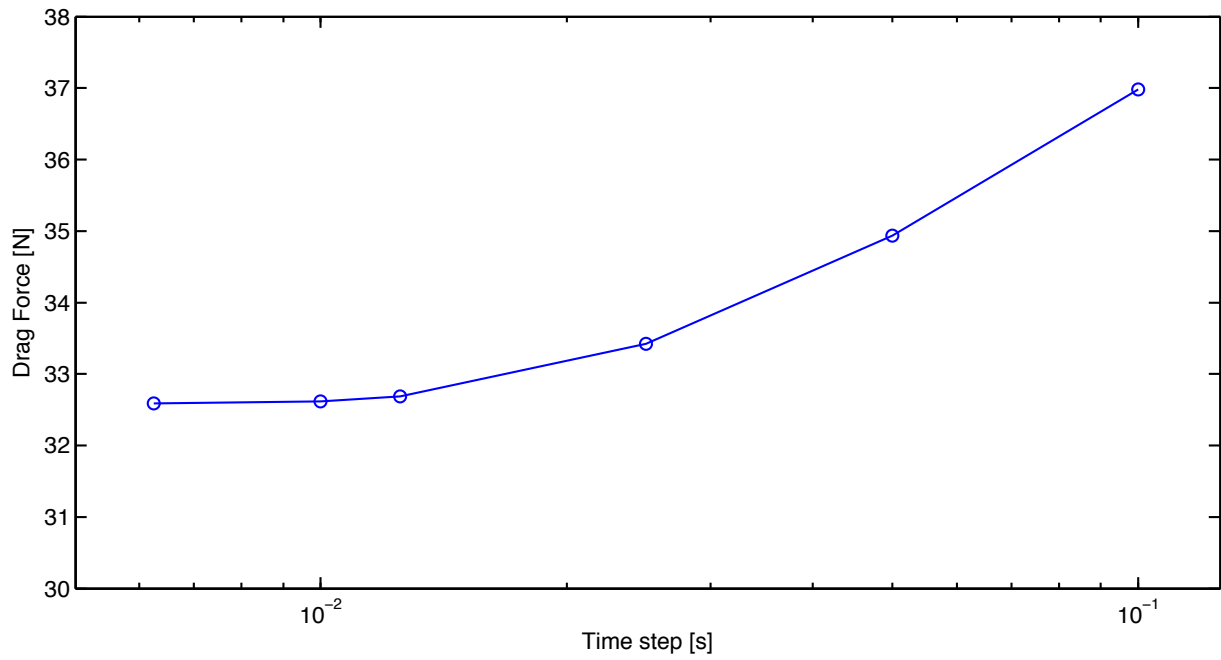


Figure 4.5: Drag force value as a function of time step size for the chemical tanker. Performed with base size = 0.1, and model scale speed = 1.42 m/s

Based on figure 4.5, a time step of 0.01 seconds were decided to be appropriate. Figure 4.6 and 4.7 show the importance of the the time step for the wave pattern. The wave pattern shown in figure 4.6 has the general Kelvin wave structure that is to be expected from a moving ship, while figure 4.7 clearly has a very unphysical wave pattern. Figure 4.7 looks nothing like the reality. A detailed analysis of the wave pattern has not been done in this project, but it was discovered that a plot of the wave pattern was a good way of detecting problematic time steps, as a too low time step generates "unphysical" waves.

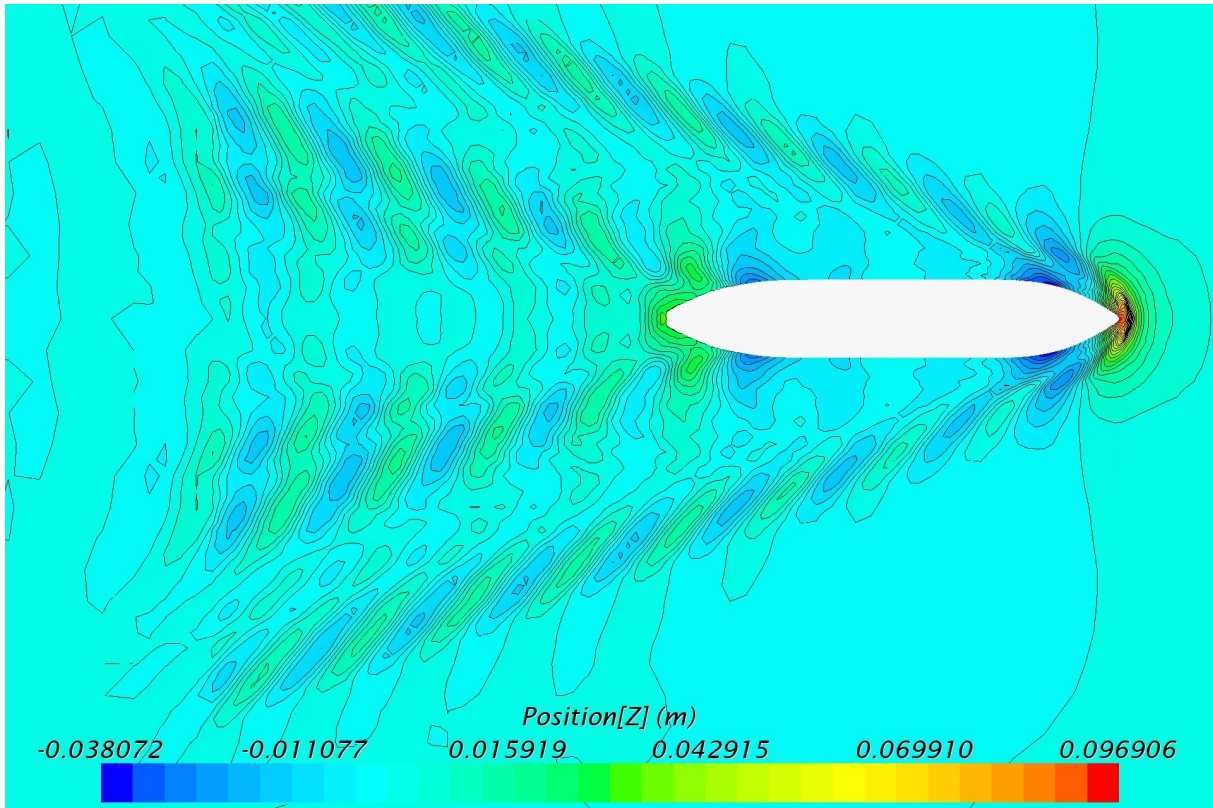


Figure 4.6: Wave pattern for the chemical tanker with time step = 0.01 s

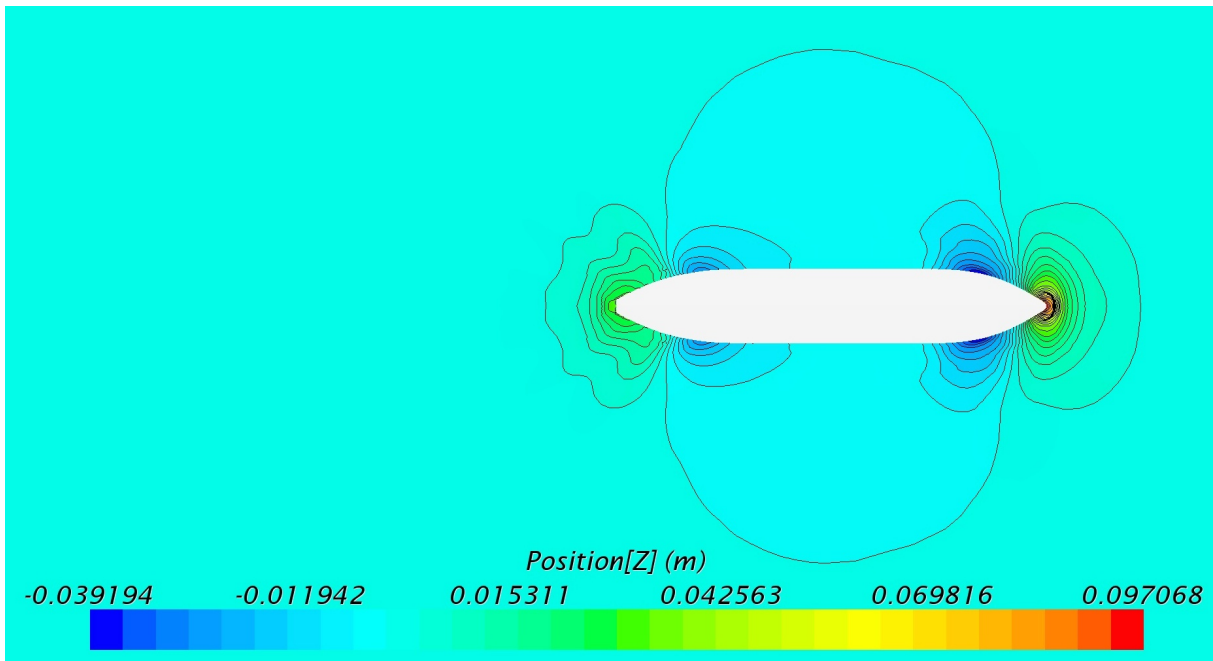


Figure 4.7: Wave pattern for the chemical tanker with time step = 0.1 s

### 4.3 Results from CFD

After the simulation parameters were decided, the actual tests were performed. Drag and side force for three speeds, and five yaw angles were done, on total 15 simulation runs. The three ship speeds corresponds to a full scale speed of 10, 12.5 and 15 knots, while the yaw angles are 0, 2, 4, 6 and 8 degrees. The raw result, presented as force acting on the ship hull, calculated by STAR CCM+ can be seen in figure 4.8, 4.9 and 4.10

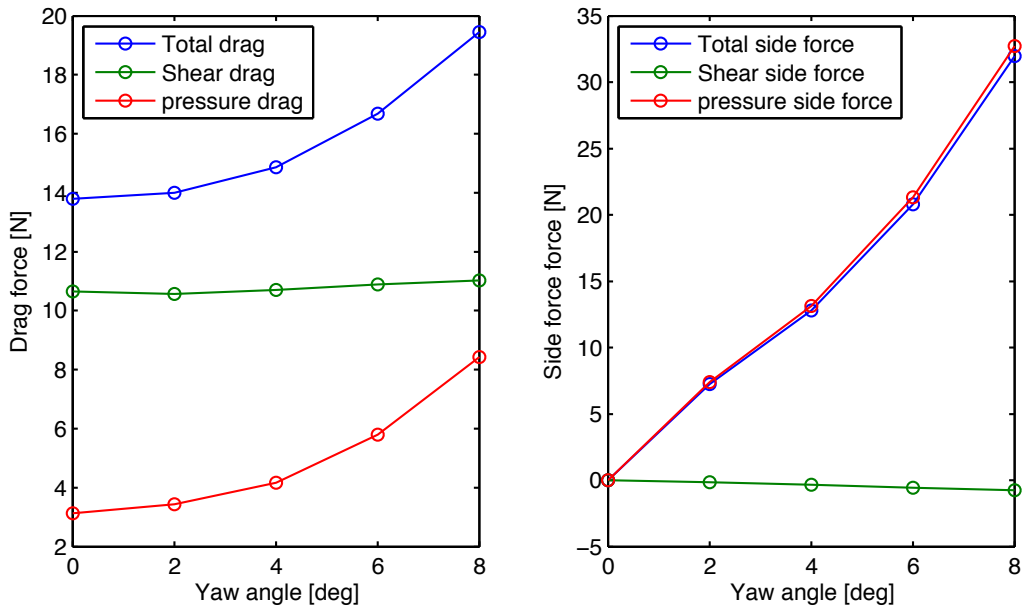


Figure 4.8: Resulting forces from CFD experiment at  $U_m = 0.95$  m/s which corresponds to  $U_S = 10$  knots

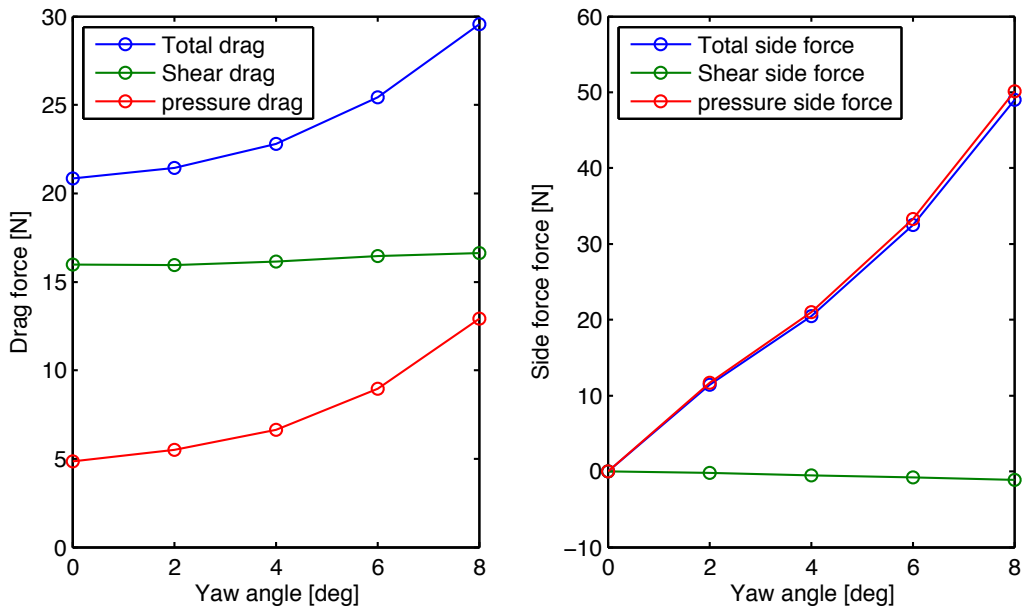


Figure 4.9: Resulting forces from CFD experiment at  $U_m = 1.18$  m/s which corresponds to  $U_S = 12.5$  knots

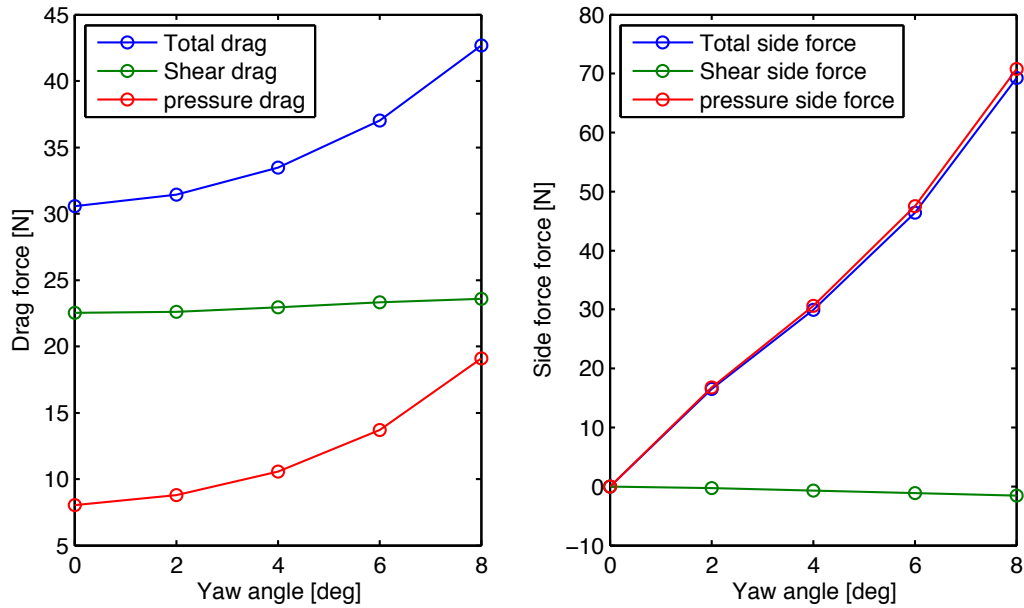


Figure 4.10: Resulting forces from CFD experiment at  $U_m = 1.42$  m/s which corresponds to  $U_S = 15$  knots

## 4.4 Comparison With Experiments

---

At a late point in the project, EFD Data data for the Series 60 geometry were discovered. It was decided that this was a good opportunity to test the simulation setup that were used in this project. The same "logic" was applied to the Series 60 model. That is, the domain size, and base size were scaled according to the length scale of the Series 60 model, which is slightly smaller than the chemical tanker. This resulted in a base size of 0.05, and simulations for yaw angles equal to 0 and 7.5 degrees were performed. The result of these experiments can be seen in figure 4.11 and 4.12. See section 4.5 for a discussion about the result.

The graphs show the forces as coefficients, where  $C_T$  is the drag coefficient and  $C_S$  is the side force coefficient.

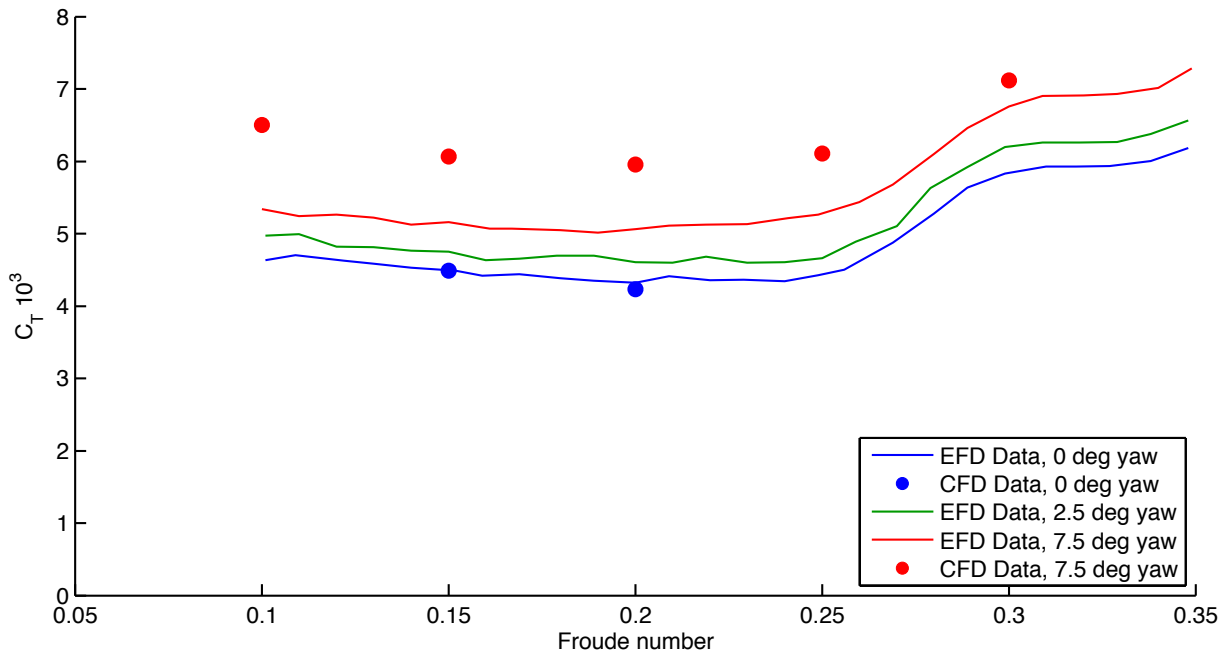


Figure 4.11: Validation test of drag force on Series 60, for different yaw angles, compared to EFD data

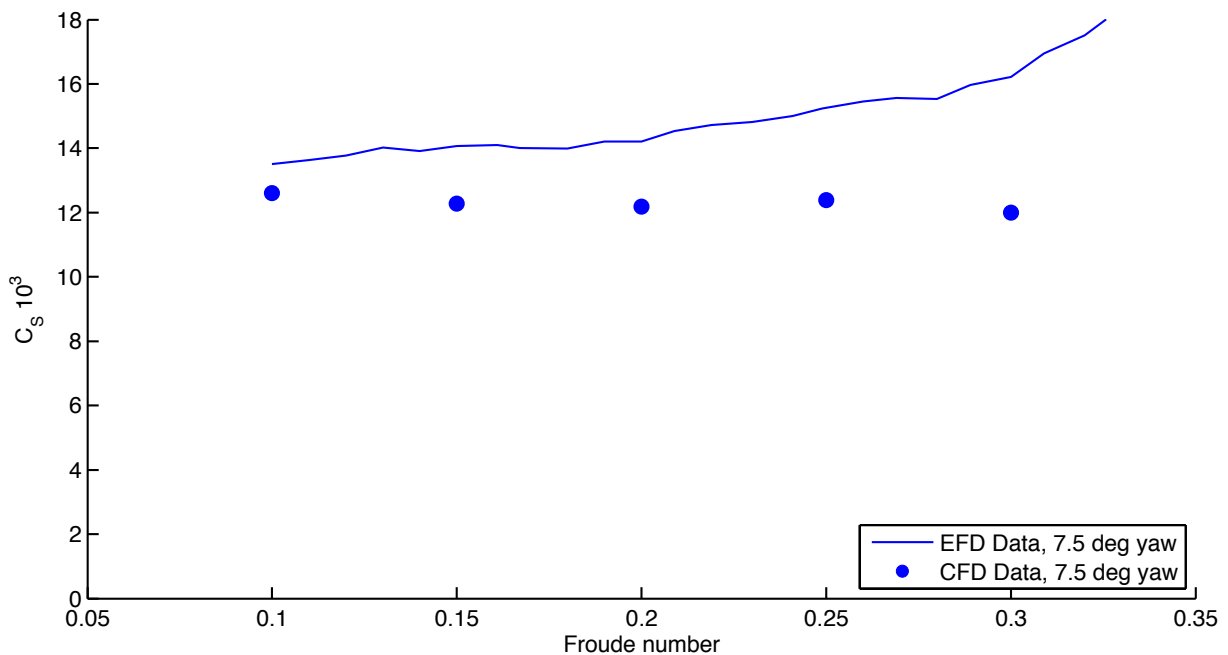


Figure 4.12: Validation test of lift force on Series 60, for different yaw angles, compared to EFD data

## 4.5 Discussion Regarding the Accuracy of the Results

For the yaw angle equal to 7.5 degrees, the result from the Series 60 tests clearly differ from the EFD Data. The drag is too high, and the lift is too low. The result for 0 degree yaw angle fits much better. This is problematic, as the main point of these tests were to see the effect of having a yaw angle on a ship. A few tests were performed in order to find the error. The number of cells in the mesh were increased, the turbulence model was changes, from k-epsilon to the so called k-omega model, the number of prism layers were increased, from 6 to 10, and the time step were decreased from 0.01 to 0.001. Nothing had any

significant effect on the result, and after doing all these tests, there were no time left, so the author had to give up. Also, some tests were done for a smaller yaw angle, in order to see the magnitude of the error for a "milder" case, but unfortunately, the simulation failed. That is, sometimes, there seems to be errors in the execution of the simulations, which makes the result "blow up". The force values never converge, even after many time steps, and the waves on the free surface are clearly wrong. This has happened a few times during this project (exclusively while running STAR CCM+ on an "old" supercomputer called "kongull", which might have something to do with the error, but the author is very uncertain of why this happens, and hesitates to speculate on this), and normally, the solution is to run the exact same simulation all over again, in order to get a completely different result. But again, due to time issues, this was impossible to complete before the delivery of this report.

There is therefore an error in the simulation, of which the importance at smaller angles of attack is uncertain, but it is clearly there for large yaw angles. The authors theory is that this has to do with the size of the simulation domain. The size of the simulation domain were decided based on a literature review, however, all the articles that were used to find an appropriate simulation domain simulated a ship without yaw angle. The result without yaw angle fits well with the EFD Data, so the result from the literature review is not "wrong" it is just for a simulation case that is not exactly as performed in this project.

The simulation domain for a lifting surface is something that should have been investigated better before performing these experiments. For instance, the company that makes STAR CCM+, CD-Adapco, has a section in their online "user portal" where they have recommendations for different simulation cases. The recommended simulation domain for a wing is "around 8–10 body lengths or wing spans, whichever is larger, from the body". This is quite a lot more than 2.5 ship lengths which is the case used in this project. It is known that the wake of a lifting surface is important for the lifting characteristics. For instance, the result from the Boundary Element Method (BEM) simulations clearly shows that a long wake is necessary. The author was aware of the importance of the potential wake throughout the project, but failed to realize that this should have been done for the the ship simulations as well. The authors guess is therefore that the too short simulation domain affects the lifting properties of the ship hull, which makes the result inaccurate. It should be less important for smaller angles of attack. After all, the lift force will be smaller, which means that less vorticity is shed from the ship hull, which again should die out faster in a viscous fluid. This theory is also based on the fact that the side force for 2 degrees seems to be a bit different than the rest of the yaw angles. That is, if one looks at the CFD result figures, and imagines a smooth line fitted to the data, the side force for 2 degrees seems to be a bit too high. Since it is known that the side force at high angles actually are too low, the case might be that the side force for 2 degrees are more or less correct, but at 4 degrees, the effect of the too short simulation domain can be seen. This is just a guess. The author needs to explore this at a later point.

Luckily, the available EFD Data can be used directly to calculate the final results, and the errors from the CFD simulation seems to point in the "right" direction in terms of the final conclusion. That is, the fact that the CFD predicts higher drag and less lift means that the importance of hydrodynamic effects will be overestimated, not underestimated.

## Chapter 5

# Other Methods

This chapter contains a brief explanation of the methods used in this project that were not large enough to get a separate chapter. In particular, all the external experiments which produced data that have been used in this project will be referenced and shortly explained. In addition, a short explanation of the software XFOIL will be given, as this software is the source of all the viscous drag coefficients used in the final wing sail model. The mathematical theory behind initial stability will be given, along with stability parameters for the two different ship models, as this is used to quantify the importance of stability in the final results. The scaling of resistance data from model scale to full scale is described. Lastly, the way everything is connected into a final model that simulates a ship moving with wing sails are explained.

### 5.1 External Experiments

---

In total, three sources of external EFD Data were used in this project. Two of them were used to check the effect of heel on ship resistance, while the third was used to check the effect of yaw. The result from these experiments are presented here, while the discussion of the consequence of these results is presented in chapter 7

#### 5.1.1 EFD Data from the University of Iowa

The "IHR—Hydroscience and Engineering at the University of Iowa" have an excellent web site, providing EFD Data completely for free, along with documentation of how this EFD Data were acquired. EFD Data for several different types of tests, along with two different types of ship geometries are provided. The intention is that this data can be used by researchers working with CFD in order to compare their results against experiments. The web site can be found in reference [40]. Unfortunately, this data were discovered a bit late in the project, so the data were only used for comparison with CFD after the simulations were supposed to be finished. A better use of this data would be to use it directly while looking for a suitable simulation setup. See chapter 4 for more on this. However, as the data is on the exact same form as the result from the CFD experiments performed in this project, the author has decided to use the raw data from these experiments when calculating the final results as well. As the Series 60 geometry is significantly different from the chemical tanker geometry, this could potentially provide and insight into the importance of ship hull geometry.

The EFD Data is collected for the series 60 geometry in a towing tank for a range of Froude numbers and several yaw angles, namely 0, 2.5, 5, 7.5 and 10 degrees. The article documenting the experiments can be found in reference [24]. The ship model has the dimensions as seen in table 5.1



Table 5.1: Model scale data, Series 60

Series 60 model	
Scale [-]	1:40
$L_{wl}$ [m]	3.048
$B_{wl}$ [m]	0.406
$D$ [m]	0.163
$\nabla$ [m <sup>3</sup> ]	0.121
$S$ [m <sup>2</sup> ]	1.579

The results used in this report is given in figure 5.1 and 5.2. Only the drag (expressed by the drag coefficient,  $C_T$ ) and side force (expressed by the side force coefficient,  $C_S$ ) are used from the EFD Data, even though more data is available, such as sinkage and trim. As can be seen in the figures, there are some "kinks" and "bumps" in the data. Reference [24] reports of an uncertainty of 0.5% for  $C_T$  and 0.1% for  $C_S$ , based on repeated test performed for Froude number equal to 0.316.

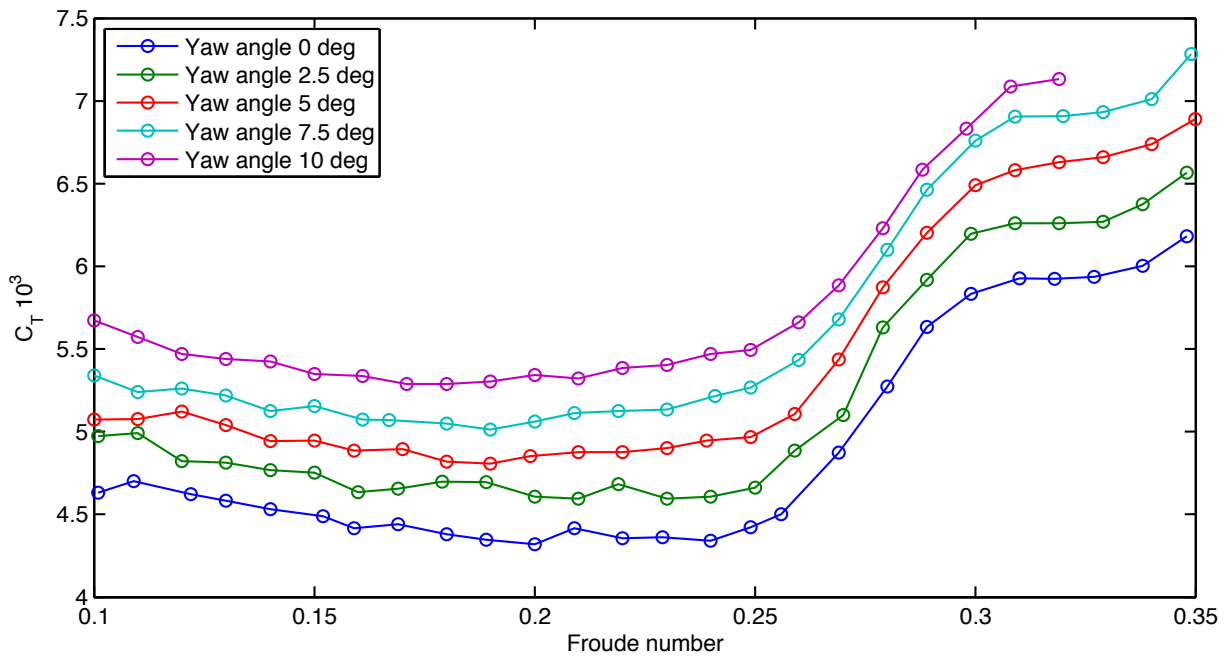


Figure 5.1: Drag coefficient,  $C_T$ , for Series 60 based on experimental results, for different yaw angles

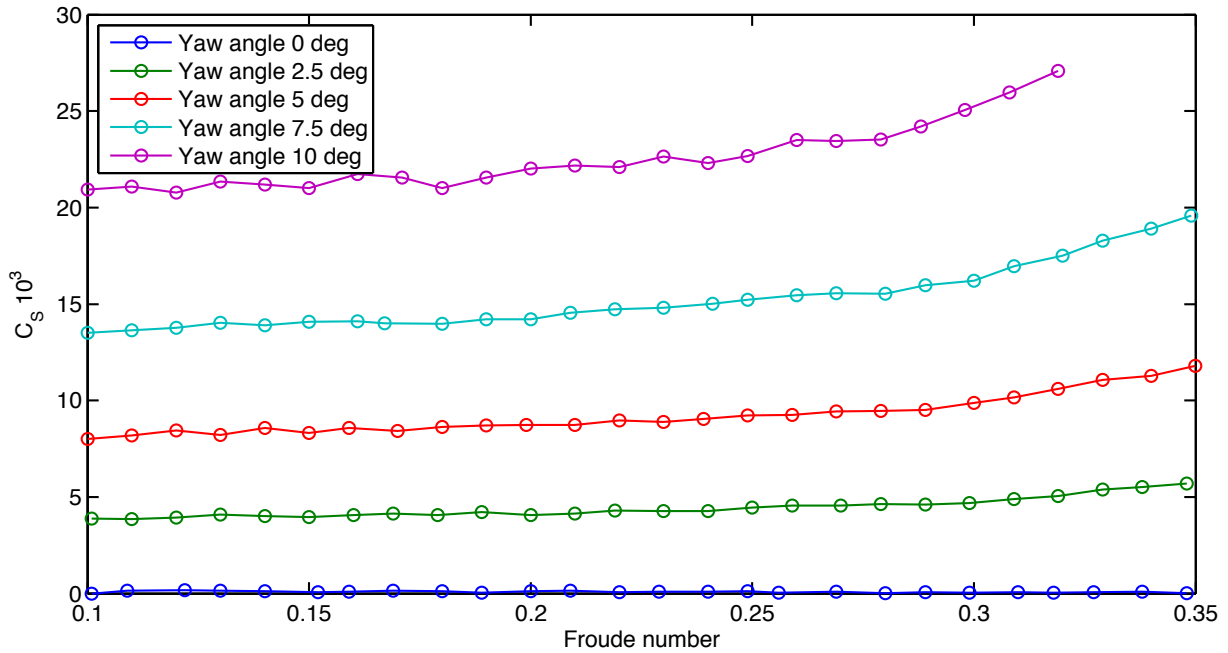


Figure 5.2: Side force coefficient,  $C_S$ , for Series 60 based on experimental results, for different yaw angles

### 5.1.2 Cargo Ship Heel Experiments

A class mate of the author, Steffen Hasfjord, were nice enough to perform a few experiments at The Norwegian Marine Technology Research Institute (MARINTEK) towing tank facilities in order to answer the authors question about the importance of heel for a cargo ship. Hasfjord was already performing towing tank test for a ship hull that is similar to the chemical tanker used in this project. The specific hull geometry were a Rolls-Royce 8000 Deadweight tonnage (DWT) chemical tanker that Rolls-Royce frequently uses for research purposes. Hasfjord's master thesis, explaining his use of the experiments, along with details about the experimental setup, can be found in reference [14]. In addition to his own experiments, which included finding the ship resistance, he performed a few extra test with a heel angles, specifically for this project. The ship model data can be seen in table 5.2

Table 5.2: Ship model data for Rolls-Royce chemical tanker, used to find the importance of heel for a cargo ship

	Full scale	Model scale
Scale [-]	1	1:16.570
$L_{wl}$ [m]	116.918	7.142
$L_{pp}$ [m]	113.2	7.056
$B_{wl}$ [m]	19	1.147
$D$ [m]	7.2	0.435
$\nabla$ [m <sup>3</sup> ]	11521.2	2.532
$S$ [m <sup>2</sup> ]	3227.27	11.754

The heel angle of the ship was created by adding weights to one side of the ship, so that a heeling moment were created, and then measuring the resulting heel angle. Four tests where performed: one first test without heel angle, one with a "large" heel angle, one with a "small" heel angle, and then one extra test without heel angle after the three first ones in order to test for any changes in the experimental setup that might create uncertainties in the resulting force measurements. The result for these experiments can be seen in table 5.3

Table 5.3: Result of heel experiments, done by Steffen Hasfjord, performed with Rolls-Royce chemical tanker

Experiment number	1	2	3	4
Carriage speed [m/s]	1.7671	1.7674	1.7675	1.7672
Heel angle [deg]	0	8.8	5.3	0
Measured Resistance [N]	74.5169	74.9897	74.7708	75.0483

### 5.1.3 Geitbåt Experiments

While doing this project, there was a project at the department of marine technology, at Norwegian University of Science and Technology (NTNU), investigating two different traditional Norwegian sailing boats, made with different construction techniques. The two boats are called "Geitbåt" and "Møringbåt" in Norwegian. The goal was to determine whether or not the boat with a more advanced construction technique (Geitbåt) provided better sailing performance than the boat constructed with a more simple construction technique. As a part of this project, the boats were towed in the towing tank facilities at MARINTEK, with different heel and yaw angles. As the boats are relatively small, they were tested at full scale. The sailing performance of the two boats were then determined by constructing a Velocity Prediction Program (VPP), with the use of the model test data. As this was happening at the same time as this project, and many of the same test were performed for the two traditional boats, the author asked to get access to the heel data in order to check the effect of heel on a traditional sailing boat. The results of this test is not yet published at the time of writing this report, but reference [33] contains the title and author of the article. The data for the two boats, at different heel angles are given, with permission, in table 5.4 and 5.5.

Table 5.4: Full scale resistance [N] for "Geitbåt", for different speeds and heeling angles

Boat speed [knots]	Heel angle [deg]			
	0	5	10	15
4	59.1	60.73	61.2	59.2
5	108.2	107.3	108.1	113.4
6	226.4	232.8	233.8	239.9

Table 5.5: Full scale resistance [N] for "Møringbåt", for different speeds and heeling angles

Boat speed [knots]	Heel angle [deg]			
	0	5	10	15
4	67.09	67.27	67.88	70.19
5	135.8	134.7	141.4	143.2
6	297	294.5	300.2	306.2

## 5.2 Viscous Modeling of Wing Profiles using XFOil

XFOIL is a 2D open source panel method that includes integral boundary layer theory in order to predict viscous effects on 2D wing profiles. The original version of XFOIL was released in 1986 by professor Mark Drela, however, some improvements have been implemented in the software since then. An article describing the theory behind XFOIL can be found in reference [7]. Mark Drela has developed XFOIL specifically for "low-speed" airfoils. In particular, XFOIL is supposed to have a robust and accurate method for predicting the viscous effect on airfoils in the Reynolds number range where the flow is switching between laminar and

turbulent flows. An article discussing the details about the integral boundary layer implementation developed by Mark Drela, and implemented in XFOIL, can be found in reference [8].

In order to test the viscous formulation, an experiment were performed by the author, originally for a different project, presented in reference [20]. There exist different friction lines for a flat plate. The analytical Blasius friction line for laminar flow, and the empirical ITTC-57 friction line for turbulent flow are compared to the viscous drag from XFOIL in figure 5.3. The foil in question is NACA 0012, and an empirical shape factor is used along with the friction line in order to model the form drag of the foil. That is, the increase in drag for the foil due to the fact that it is not a flat plate, but a relatively thick solid body. The shape factor is given in reference [12], as follows:

$$k = 2 \left( 1 + (t/c) + 60(t/c)^4 \right) \tag{5.1}$$

In this equation,  $t$  is the thickness of the foil, and  $c$  is the chord length. This shape factor is used along with the friction factor,  $C_f$ , from each of the friction lines in order to estimate the viscous drag coefficient,  $C_{Dv}$ , as follows:

$$C_{Dv} = k \cdot C_f \tag{5.2}$$

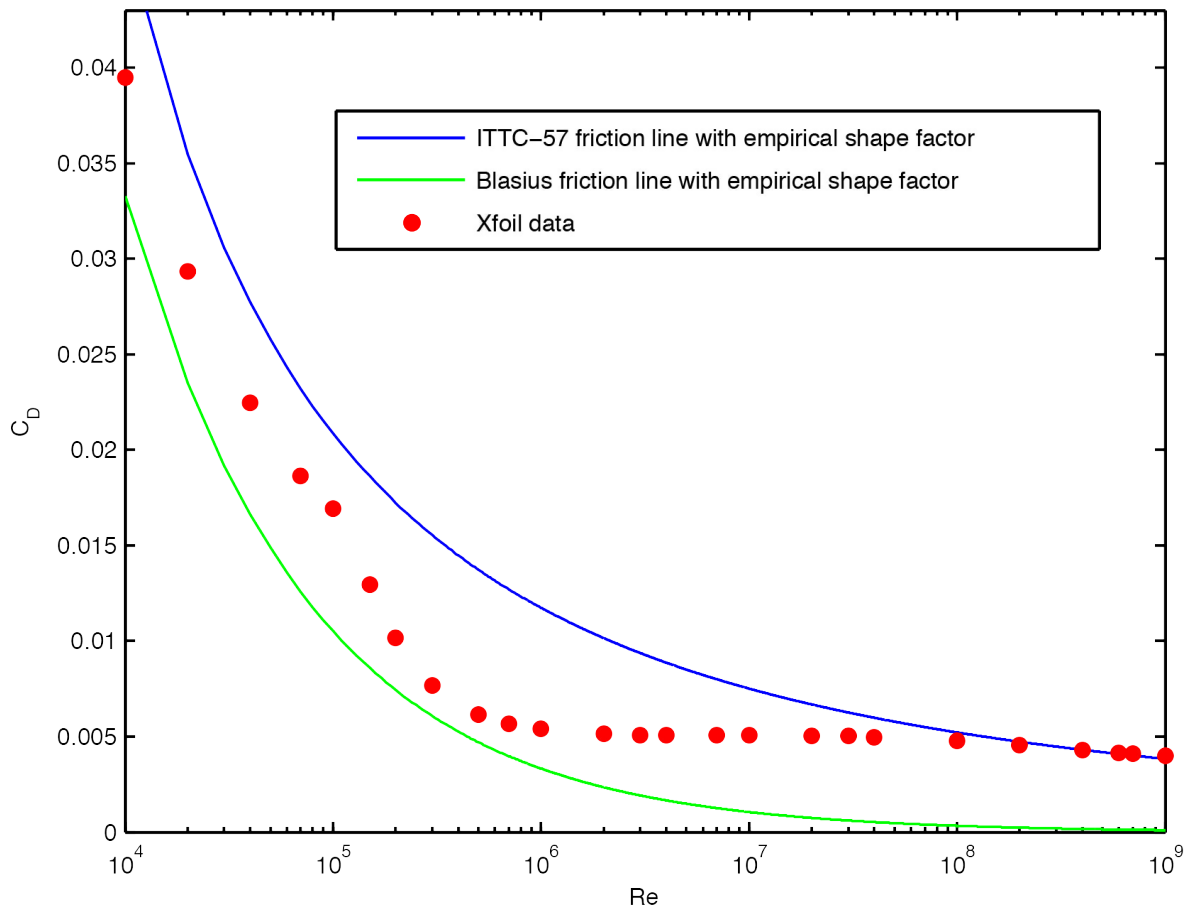


Figure 5.3: Viscous drag predicted by XFOIL compared to empiric shape factor for foils, and different friction lines. Performed with NACA 0012 as geoemtry

Based on this experiment, and the articles from Mark Drela, the author has concluded that XFOIL probably predicts accurate viscous drag for two-dimensional airfoils.

## 5.3 Stability Calculations

---

Stability is calculated by assuming initial stability theory. That is, relatively small angles of attack are assumed. Initial stability is, according to reference [1], usually valid for heeling angles up to around 10 degrees. Stability of a ship is not only dependent on the ship hull geometry, it is also dependent on how the weight of the ship is distributed. That is, the center of gravity has a large influence on the stability of the ship. As the center of gravity will be highly dependent on how the ship is loaded with cargo, what type of cargo the ship is transporting, and how the hull and machinery is constructed and placed, it was decided to quantify the importance of stability by calculating the required distance from the keel of the ship to the center of gravity, KG. This is a length scale that is understandable. By comparing the value KG to the draft of the ship, it is possible to make a judgement on how difficult it is to achieve the required value of KG. In this section, the mathematical theory of initial stability will be given, which is based on reference [1]. The required value of KG is calculated by finding the value of KG that creates a restoring moment for the ship that is equal to the heeling moment from the wing. The stability parameter data for the two ships will also be given.

The required value of KG is dependent on the maximum allowable heeling angle. What this angle should be, is a hard question to answer. Stability is a big topic, where safety considerations are a big part. This is a bit outside the scope of the project, which is mainly focused on resistance. Based on the literature review (chapter 2) and the heel experiments performed in this project, it seems that heeling angles that are less than 10 degrees are non-problematic from a resistance point of view. In order to be somewhat conservative, a maximum allowable heeling angle is set to be 5 degrees. The maximum required value of KG is calculated based on this. The value of KG will not take into account any safety considerations, only the demand that the heeling moment from the wing sail shall not create larger heeling angles than 5 degrees.

### 5.3.1 Mathematical Theory of Stability

Initial stability theory gives a relationship between geometric variables for a ship hull and the distance from the center of gravity, to the rotation axis, known as the metacenter. This value is known as GM. Based on the value of GM, the restoring moment arm, GZ, can be calculated. When GZ is known, the restoring moment is calculated by taking the weight of the ship multiplied with the restoring moment arm. The initial stability equations, taken directly from reference [1], is as follows:

$$GM = KB + BM - KG \quad (5.3)$$

$$BM = \frac{I_{yy}}{\nabla} \quad (5.4)$$

$$GZ = GM \sin(\theta) \quad (5.5)$$

$$M_x = g\rho\nabla GZ \quad (5.6)$$

The requirement for KG is that the heeling moment from the wing must not exceed the restoring moment from the ship. The side force from the wing is named  $F_y$ , and it is assumed that the force is acting at the middle of the wing. That is, the height of the wings is 40 m, so the distance from the deck to the attack point of the force is 20 m. Since the moment balance is taken around the center of gravity, the total arm of the side force from the wing will be equal to  $H/2 + D - KG$ . The resulting requirement for the value of KG can be seen below:

$$\text{Static restoring moment} \geq \text{Heeling moment from wing} \quad (5.7)$$

$$g\rho\nabla GZ \geq F_y \left( \frac{H}{2} + D - KG \right) \quad (5.8)$$

$$g\rho\nabla GM \sin \theta \geq F_y \left( \frac{H}{2} + D - KG \right) \quad (5.9)$$

$$g\rho\nabla (KB + BM - KG) \sin \theta \geq F_y \left( \frac{H}{2} + D - KG \right) \quad (5.10)$$

$$g\rho\nabla (KB + BM) \sin \theta - F_y \left( \frac{H}{2} + D \right) \geq KG (g\rho\nabla \sin \theta - F_y) \quad (5.11)$$

$$KG \leq \frac{g\rho\nabla (KB + BM) \sin \theta - F_y \left( \frac{H}{2} + D \right)}{(g\rho\nabla \sin \theta - F_y)} \quad (5.12)$$

### 5.3.2 Ship stability data

The ship stability data is presented in table 5.6. The values for the chemical tanker is taken from the software FreeShip, since the geometry of the chemical tanker originates from this software, and FreeShip have functions that calculates initial stability parameters. The values for Series 60 is calculated with the 3D modeling software Rhino 3D, as this software contains methods to calculate the necessary values based on CAD models.

Table 5.6: Stability data for the ship models used in this project

	FreeShip Chemical Tanker	Series 60 Container Ship
$D$ [m]	10	6.5
$\nabla$ [m <sup>3</sup> ]	38147	7744
KB [m]	5.215	3.5
$I_{yy}$ [m <sup>4</sup> ]	266172	23191
BM [m]	6.978	2.995

## 5.4 Going from Model Scale to Full Scale

All the resistance values in this project are originally from model scale ships. Since the resistance in full scale is different from the resistance in model scale, some scaling must be applied. The scaling is done by assuming the ITTC57 friction line, and only friction resistance is scaled. Air resistance and appendage resistance is neglected, and non of the ship models have any transom stern so this is not modeled. The friction coefficient based on the ITTC57 friction line is given as follows:

$$C_F = \frac{0.075}{(\log_{10} Re - 2)^2} \quad (5.13)$$

As the ship is not a flat plate, a form factor is needed. For the Series 60, MARINTEK's empirical formula for a form factor is used, taken from reference [37]

$$k = 0.6\Phi + 75\Phi^3 \quad (5.14)$$

$$\Phi = \frac{C_B}{L} \sqrt{2B \cdot T} \quad (5.15)$$

This gives the result  $k = 0.0705$  for Series 60. The friction resistance for the chemical tanker is calculated by assuming a constant relationship between viscous and friction line.

In order to model roughness on a full scale ship hull, a roughness factor is used. This is based on empirical formula, taken from reference [37]:

$$\Delta C_F = \left[ 110 (H \cdot U_s)^{0.21} - 403 \right] C_F^2 \quad (5.16)$$

$$H = 150 \quad (5.17)$$

The residual drag coefficient,  $C_R$ , is assumed to be equal to the pressure drag coefficient for the CFD data, while  $C_R$  is calculated by subtracting frictional resistance from the total resistance from the EFD Data. The total resistance is then calculated as follows:

$$C_T = C_R + (1 + k) (C_F + \Delta C_F) \quad (5.18)$$

The side force is not scaled. That is, the side force coefficient,  $C_S$  is assumed to be the same in both model scale and full scale. This is based on the fact that the side force is mainly from pressure, and for yaw angles well below stall, viscous effects are known to have little effect on lifting surfaces.

## 5.5 Final Coupled Model

---

When the resistance is found as a function of speed and yaw angle, either from CFD or from EFD Data, and the wing model is constructed (see chapter 6), then everything must be put together, in order to produce the final results, seen in chapter 8. Three things are plotted for both ships: effective thrust from wing sails, with and without taking into account the hydrodynamic effects of yaw, the yaw angles experienced by the ships, under the influence of the wing sails, and finally, the required values of KG in order to not exceed 5 degrees heeling angle. Resistance is only considered to be a function of speed and yaw angle, not heeling angle (see chapter 7 for more on this), as heeling angles seems to have little effect on the resistance. The final results are really just a balance between side force from the wings and side force from the hull. Two different approaches are used while looking at the thrust from the wing sails: thrust at constant speed and constant wind velocity, for different true wind directions, and speed as a function of wind direction and wind speed.

All the Matlab scripts that are used to generate the plots shown in chapter 8 can be seen in section 11.2

### 5.5.1 Effective Thrust Prediction

The effective thrust is found by first finding the angle of attack for the wing model, that produces the highest amount of thrust. This is done by finding the highest possible thrust coefficient, as a function of apparent wind direction. For a given speed, and a given true wind direction, the apparent wind direction is found, and the corresponding maximum thrust coefficient is used to calculate the thrust from the sails. The side force from the sails are found by using the side force coefficient that corresponds to the maximum thrust, and the balance between the sails and the hull are found by numerically solving for the yaw angle that produces the necessary side force. This is done by using built in interpolation functions in Matlab

The added resistance due to yaw is defined as the resistance at the yaw angle that generates enough side force to balance the side force from the wing sails, minus the resistance at zero yaw angle. The effective thrust is then defined as the thrust from the wing sail, minus the added resistance. The effective thrust, divided by the resistance for zero yaw angle will then be a way to measure the effect of the wing sails. If this

value is one, the wing sails will provide enough thrust to push the ship forward at the given speed alone. If the value is less than one, an engine has to help.

The yaw angle that creates enough side force to balance the wing sail is stored for plotting. This will be a function of wind direction, in the same way that as the thrust. The necessary value of KG is also calculated and stored for plotting. This is a function of the side force, and thereby also a function of wind direction.

### **5.5.2 Velocity Prediction**

The prediction of ship velocity, with a given wind speed, is slightly more computational demanding, as more iterations has to be performed in order to find a balance. As a first guess of the speed of the ship, the resistance and thrust at zero yaw angle, and zero ship speed is calculated. This value for the ship speed is wrong, which makes the value for the apparent wind direction wrong, which makes the thrust wrong. The resistance is off course also wrong, as yaw angles are not yet considered. However, this works as a good first guess. The balance is found by simply iterating until the change in predicted ship velocity from one iteration to the next is small (less than 0.1% for the specific plots used in chapter 8). An updated value of the ship velocity makes for a better guess for the apparent wind direction, which makes for a better guess for the thrust, etc. In the end, this approach gives convergence for the ship velocity, and a ship velocity corresponding to balance in wing thrust and ship resistance, and side force from ship and wing is found.



## Chapter 6

# The Physics of Wing Sails and How to Model it

Wing sails are, as the name suggest, very similar to normal wings. They are both lifting surfaces, which is defined as a body that can create a force normal to the incoming flow direction, i.e. *lift*. They are also similar to normal sails, as they create useful force from the wind when some component of the total force points in the ships direction of travel, i.e. *thrust*. The thrust is dependent on the speed and direction of the apparent wind. The apparent wind is defined as the sum of the true wind velocity and the incoming velocity due to the motion of the ship. The apparent wind hits the sail at an angle relative to the direction of travel, and lift will be generated normal to the apparent wind, while drag is generated parallel to the apparent wind. The nomenaure used to describe wing sails are the same as the one used to describe normal wings. An overview of some expressions are given in figure 6.1.

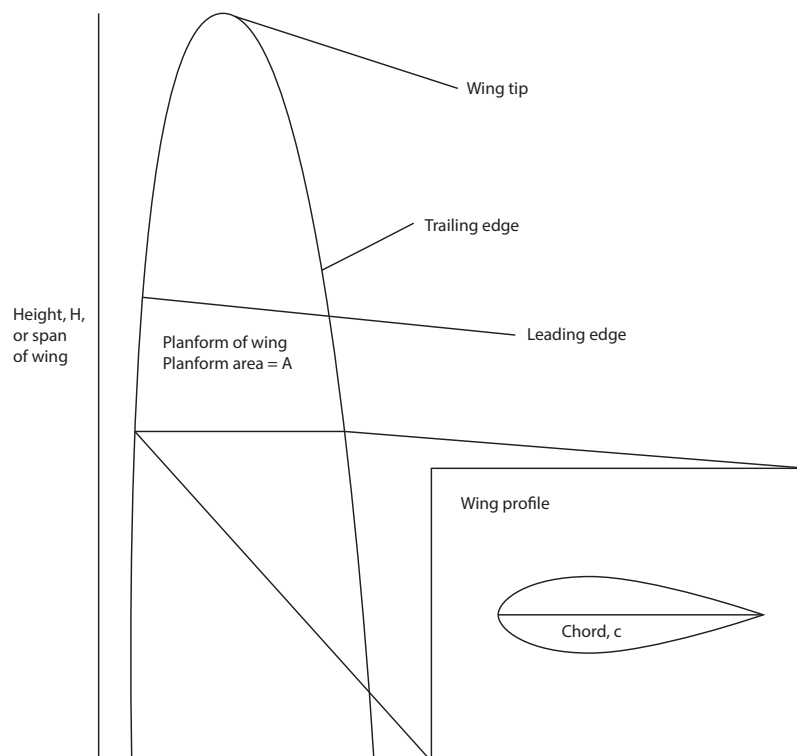


Figure 6.1: Nomenaure used to describe wing sails

In this chapter, there will first be a brief overview of the forces that act on a wing sail, and how these forces are modeled in this project. Then there will be a discussion about the differences between normal airplane wings and wing sails. Although the similarities between airplane wings and wing sails are striking, there are at least one fundamental difference, that should have some effect on the design of the wing sail. After this discussion section, some simple numerical experiments are performed in order to pick a foil profile to use in the wing sail simulation in this project. This will not be an extensive study of the shape of the foil profile, but a few candidates are tested. In addition to finding a foil profile, the 3D effects on the wings must be determined, as this is known to be of great importance. Quantifying the reduction in lift, and increase in drag due to 3D effects are done with the custom Boundary Element Method (BEM) code for both rectangular wings alone, and 8 wings in a row. The wings for both the chemical tanker and Series 60 are tested. The dimensions of the wings were determined in the introduction, and can be seen in table 1.2.

## 6.1 Forces from a wing sail

---

When air flows over a wing, forces are created. The conventional way of looking at these forces are to divide them in two, namely *lift* and *drag*. Drag acts in the same direction as the incoming flow, while lift acts perpendicular to the incoming flow. This is the most natural way of looking at wing forces when one are talking about airplanes: lift is the useful force that keeps the plane from falling to the ground, while drag is the negative force, that creates the need for big jet engines. For a wing sail, it is more natural to divide the forces into *thrust* and *side force*. Thrust pushes the ship forward, while the side force tries to push to ship sideways. Thrust and side force consists of both lift and drag, where the relationship between lift, drag, thrust and side force are dependent on the apparent wind direction.

### 6.1.1 Lift

Lift, although inherently a phenomenon that only exist due to viscosity, is modeled pretty accurately by potential theory, at least for small angles of attack (a fact that is covered in many text books, for instance reference [2]). By enforcing the *Kutta condition* at the trailing edge of a wing, the viscous effect that creates lift can be captured in an accurate way. The Kutta condition says that the flow must leave the trailing edge of a wing *smoothly*. Or, to say it in a an equivalent way, the rear stagnation point must be at the trailing edge. This is the only solution to a potential flow around a body with a sharp trailing edge, that does not produce infinite velocities at the trailing edge. Although there is no mechanism for stopping infinite velocities in potential theory alone, viscosity in the real world will always make sure this does not happen (even though very large velocities can happen for short periods of time, such as in the startup period of flow around wings [2]).

For small angles of attack, the flow around a wing will be linearly proportional to the angle of attack. That is, as long as the flow is attached to the wing, and separation happens at, or very close to, the trailing edge, potential theory predicts accurate lift, which is linear as a function of angle of attack. For large angles of attack, separation occurs a different place than the trailing edge, and the linear behavior stops.

Three-dimensional effects have large impact on the lift forces from a wing. The lift force happens because one side of the wing have low pressure, while the other have high pressure. At the tip of the wing, the flow from the high pressure side will flow around the tip to the low pressure side. This lowers the pressure on the high pressure side, and increases the pressure on the low pressure side, which in effect reduces the lift. Another way of looking at it is by considering the fact that this flow around the wing tip will create vortices. In potential theory, a vortex will induce velocities, which happens to be such that they reduce the effective angle of attack for the wing.

Lift is quantified with lift coefficients,  $C_L$ . Viscous effects on lift can be approximated by looking at the viscous effect on the two dimensional lift coefficient,  $C_{L,2D}$ . Neglecting viscous effects will predict a two-dimensional lift coefficient equal to  $2\pi\alpha$ , where  $\alpha$  is the angle of attack for the wing. In this project, viscous effects on lift will be predicted on the two-dimensional *foil profile* of the wing, using XFOIL, while the BEM

code will be used to predict three-dimensional effects. That is, the following expressions apply:

$$C_L = \frac{\text{Lift}}{\frac{1}{2}\rho AU^2} = C_{L,2D,viscous} \cdot \frac{C_{L,BEM}}{2\pi\alpha} \quad (6.1)$$

### 6.1.2 Drag

Drag on a wing can be divided in two: viscous drag and potential drag. As with everything existing in a moving fluid, viscosity will create drag forces on a wing, with or without lift. Since a wing is a slender body, where the flow is attached, this viscous drag force will mainly be due to friction between the flow of air and the wing surface.

Three-dimensional potential theory also predicts drag, however, only for a wing that creates lift. This is often called *lift-induced drag* or *vortex drag* as the drag forces stems from the induced velocities that are created by the vortices that are shed from wing in the process of creating lift. The act of creating lift, creates vortices at the wing tips, which alters the velocity field at the wing itself, which again has the effect of "tilting" the lift in the direction of the incoming velocity. This has the effect that the 3D force vector has a component in the direction of the incoming flow, i.e. drag.

A common model for lift-induced drag is to assume that it is proportional to the lift coefficient squared. This is based on theoretical results for an elliptic wing, the planar wing shape that gives the least amount of lift-induced drag. For an elliptic wing, the lift induced drag coefficient can be written as follows:

$$C_{Di,elliptic} = \frac{\text{Drag from potential theory}}{\frac{1}{2}\rho AU^2} = \frac{C_L^2}{\pi Asp} \quad (6.2)$$

For a non-elliptic wing, the general behavior will be the same, i.e. proportional to the lift coefficient squared, but slightly higher. This is often expressed as follows:

$$C_{Di,non-elliptic} = \frac{C_L^2}{\pi e Asp} = \frac{C_L^2}{\pi Asp} (1 + \delta) \quad (6.3)$$

The value of  $\delta$  is often small. For instance, for a rectangular wing, with  $Asp = 10$ ,  $\delta = 0.08$ , based on data from reference [34]. The factor  $e$  is known as the span efficiency factor. The span efficiency factor is only including potential effects. The viscous nature of drag is also such that it is approximately proportional to the lift coefficient squared. If viscous effects are included in  $e$ , it is often called the *Oswald efficiency number* [47].

For this project, viscous effects are modeled with XFOIL by using integral boundary layer theory, and the data for a certain wing profile, for a certain angle of attack, is found by interpolating the raw data from XFOIL, rather than finding a viscous value of  $e$ .  $\delta$  is found by using the BEM code. As  $\delta$  is rather small, the exact value for  $\delta$  is not very important. Accurate values for the lift coefficient have a much bigger influence on the drag values than accurate values of  $\delta$  (see chapter 3 for validation of the lift values calculated by the BEM code). The complete drag model for wings in this project can be written as follows:

$$C_D = \frac{\text{Drag}}{\frac{1}{2}\rho AU^2} = C_{Dv}(\text{from xfoil}, \alpha) + \frac{C_L^2}{\pi Asp} (1 + \delta_{\text{from BEM}}) \quad (6.4)$$

Here,  $C_{Dv}$  is the viscous drag coefficient, and  $\alpha$  is the angle of attack for the wing.

### 6.1.3 Thrust and Side Force

The thrust and side force are given when the lift and drag is known. Thrust and side force are only a transformation of lift and drag into a new coordinate system. When talking about lift and drag, the coordinate system has the x-axis parallel with the incoming flow, and the y-axis normal to the incoming flow. When talking about thrust and side force, the x-axis is pointed in the direction of travel for the ship. That is, towards the bow. The y-axis is pointed to the port/left side of the ship. See figure 6.2 for a visual explanation.

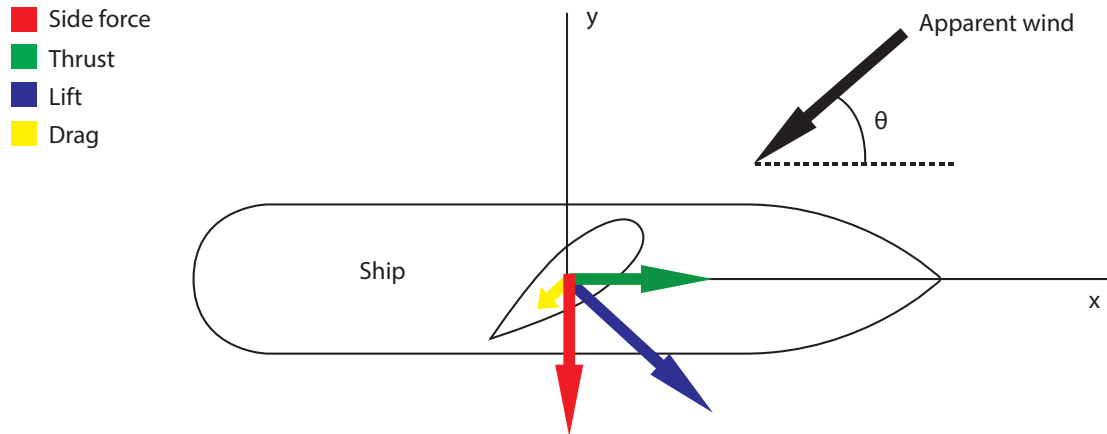


Figure 6.2: Illustration of the relationship between the different forces acting on a wing sail

In mathematics, the relationship between lift, drag, thrust and side force will be as follows:

$$\text{Thrust} = \text{Lift} \cdot \sin \theta - \text{Drag} \cdot \cos \theta \quad (6.5)$$

$$\text{Side Force} = -\text{Lift} \cdot \cos \theta - \text{Drag} \cdot \sin \theta \quad (6.6)$$

The thrust is symmetric about the x-axis, while the side force is anti-symmetric, so in this report, thrust and side force are only calculated for  $\theta$  values between 0 and 180 degrees.

In order to make the thrust and side force non-dimensional, coefficients are used.  $C_x$  are the thrust coefficient, while  $C_y$  is the side force coefficient. They are defined as follows:

$$C_x = \frac{\text{Thrust}}{\frac{1}{2}\rho AU^2} = C_L \sin \theta - C_D \cos \theta \quad (6.7)$$

$$C_y = \frac{\text{Side force}}{\frac{1}{2}\rho AU^2} = -C_L \cos \theta - C_D \sin \theta \quad (6.8)$$

$$(6.9)$$

The thrust from a wing sail will be dependent on the angle of the apparent wind. The relationship between apparent wind and true wind can be seen in figure 6.3.

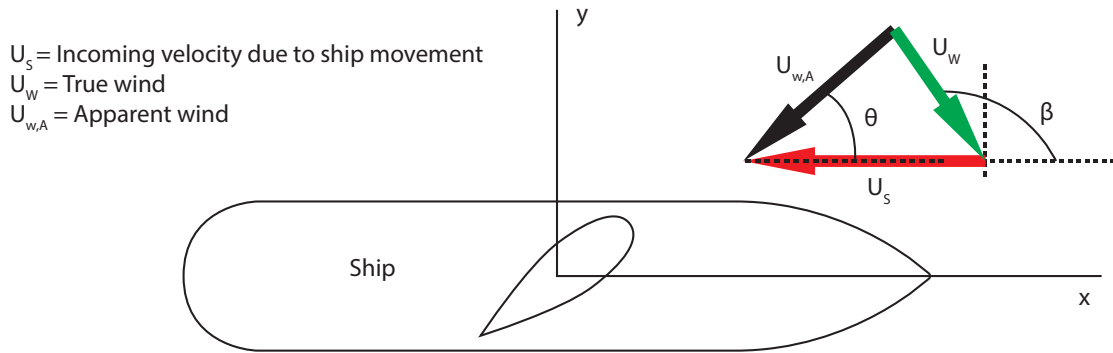


Figure 6.3: Relationship between wind, ship velocity and apparent wind

The velocity components, and angle, of apparent wind, referring to figure 6.3, are given as follows:

$$u_{w,A} = -U_w \cos \beta - U_s \quad (6.10)$$

$$v_{w,A} = -U_w \sin \beta \quad (6.11)$$

$$\theta = \tan^{-1} \left( \frac{v_{w,A}}{u_{w,A}} \right) \quad (6.12)$$

## 6.2 Wing Sails vs Conventional Wings

While the purpose of a normal wing is to use the lift to counter the pull of earth's gravity on an airplane, the purpose of a wing sail is to push the ship forward as fast as possible. This creates one important difference, which has a consequence when it comes to the design of the wing.

A normal airplane wing must be designed in such a way that it can create *enough* lift to keep the airplane flying. Drag is purely a negative effect that increases the required propulsive power. This means that an optimal design of a normal airplane wing could be defined as the wing that creates *enough* lift, with the *minimum* amount of drag (This might not be true for more special airplanes, such as fighter jets, acrobatic planes etc, but is fitting for a passenger/cargo airplane). The amount of lift that can be considered *enough* is of course dependent on the weight of the airplane, which also is dependent on the size of the wing, but it will be a finite amount. There exists a limit to how much lift there is necessary to create for a normal airplane wing. Since the nature of lift is such that the very act of lifting also creates drag, it's not necessarily beneficial to use high-lift-coefficient wings. Doubling the lift coefficient reduces the necessary wing area to half its original value, which also, roughly speaking, reduces the viscous drag to half its original value. However, since the lift-induced drag is proportional to the lift coefficient squared, doubling the lift coefficient will actually multiply the lift-induced drag coefficient by four. The wing area is cut in half, but this only makes it so that the lift induced drag will be doubled, instead of multiplied by four. Because of this, there exists an optimal wing area, and thereby an optimal lift coefficient, for a certain wing design at a certain speed. Often this results in an optimal lift coefficient that is rather low. This can for instance be seen if one studies the mid-span foil profile for the passenger airplane Boeing 737, which can be seen in figure 6.4.

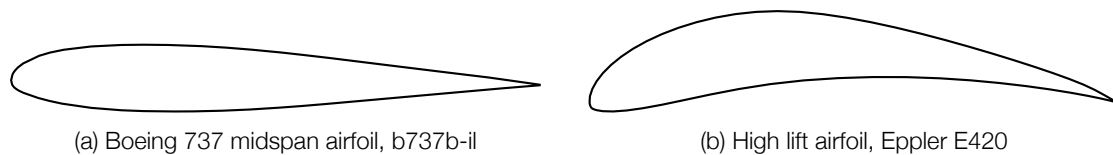


Figure 6.4: Example of airplane wing profiles. source: [42]

Compared to the high-lift foil profile Eppler E420, this foil profile has very little camber, and thereby, very low lift coefficient (0.1391 according to XFOIL). If the lift coefficient was higher, there could be a smaller wing area, and thereby lower viscous drag, but a higher lift coefficient would also increase the induced drag, and overall there is probably no benefit in doing this (since Boeing has decided to use this specific foil). So the goal is just to create enough lift with minimum amount of drag, and there is a limit to how much lift that needs to be created. As high lift coefficients creates high lift-induced drag, foil profiles for airplanes will many times have low lift coefficients. That is, they are *NOT* high lift-wings.

For a wing sail, the story will be quite different. This has to do with the fact that drag itself has very little negative effect. For instance, if the apparent wind direction,  $\theta$ , is 90 degrees, which means that the apparent wind is coming directly from the side of the ship, the drag will not have any negative effect on the thrust at all. If  $\theta$  is larger than 90 degrees, the drag actually gives positive thrust. The only time drag gives negative thrust is when  $-90 < \theta < 90$ , that is, when the apparent wind is coming from slightly ahead of the ship.

But even in this range, there are areas where drag have little to say. For  $|\theta| > 45$ ,  $\cos\theta < \sin\theta$ , which means that lift is given more weight than drag in the thrust coefficient equation, equation 6.7. In addition, lift is in generally higher than drag, and gives more contribution anyway. All in all, this gives a situation where very high angles of attack is favorable for many of the apparent wind directions. In order to test this, viscous data from XFOIL was collected, for the foil profile NACA 0014, at Reynolds number equal to 6 million (see section 6.3 for more on this). This data was then used, together with elliptic wing theory for three-dimensional lift and lift-induced drag, to build a wing model where the optimal angle of attack can be calculated. The optimization is done with brute force in Matlab. That is, many angles are tested, with resolution of 0.1 degrees, and the angle of attack with the highest thrust coefficient is stored as the optimal angle of attack for a given apparent wind direction. The result, for different aspect ratios can be seen in figure 6.5

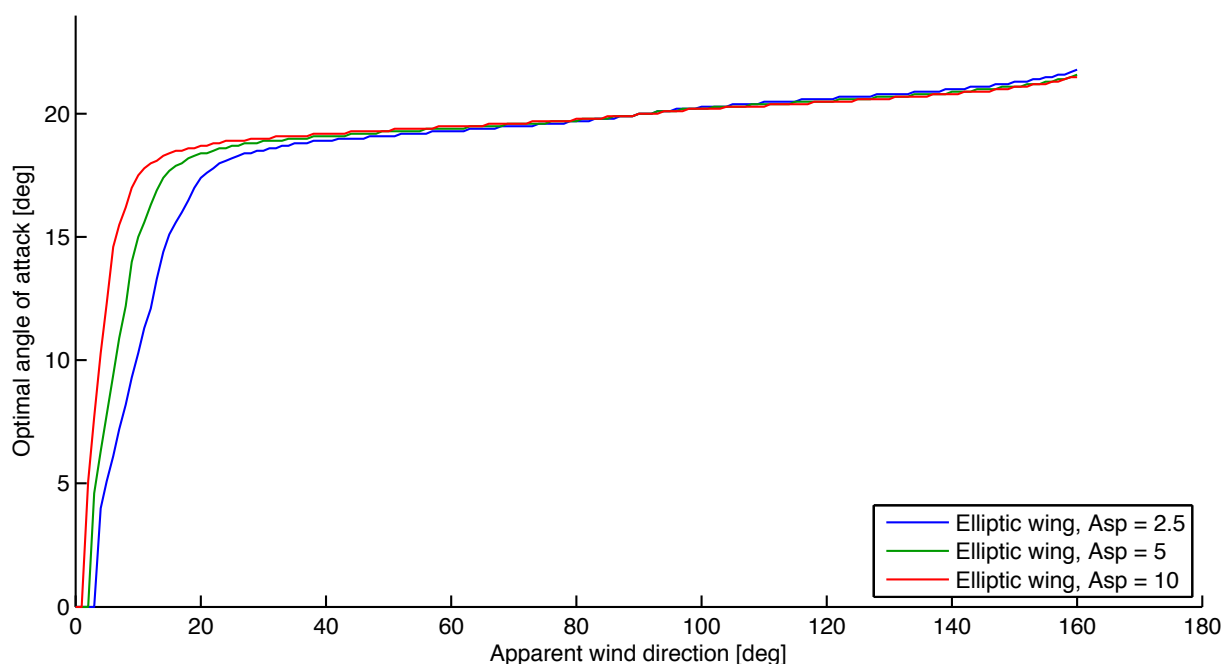


Figure 6.5: Optimal angle of attack for a wing sail, calculated based on elliptic wing theory, using viscous data from XFOIL and different aspect ratios

The result is that angle of attack at around 20 degrees is optimal for almost all apparent wind directions, except for  $\theta < 20$ . For very small values of  $\theta$  the optimal angle of attack drops quickly, which also means that there is little thrust to be gained from a wing sail in these apparent wind directions. In general, it seems that if one wishes to increase the thrust coefficient, one should try to increase the lift of the sail, rather than decrease the drag. High lift devices, such as flaps and leading edge slots are probably a good idea. High lift devices are not tested in this project, but based on the arguments made so far, it seems that this should be investigated at a later time.

It is also worth nothing that operating at such high angles of attack means that the wing are operating very close to stall. It is known that stall can lead to drastic change in both drag and lift. If the wind direction for some reason suddenly change, the actual angel of attack for the wing might be pushed into a stalling angle. If the stalling characteristics of the wing is "bad", i.e. stall causes sudden large drop in lift, and large increase in drag, it could be a large practical problem. The speed of the ship might suddenly drop, or alternatively, the engine must suddenly increase the power output in order to compensate for the loss of thrust from the wing sail. Stalling characteristics of a wing is therefore also something that should be investigated at a later time.

## 6.3 Foil Profile

---

The purpose of this project is not to find an optimal design of a wing sail, but in order to have a wing model that is realistic, some design choices have to be made. This section goes through some simple experiments, performed with the computer software XFOIL, that are meant to give an idea of how the foil profile for a wing sail should look. 2D lift and drag data is extracted from XFOIL, using the viscous integral boundary layer theory in order to predict the effect of viscosity on lift and viscous drag. The 2D data are then used along with elliptic wing theory in order to predict 3D effects, such as reduction in lift, and lift-induced drag.

The forces from the complete wing model is then transformed into thrust and side force, according to equation 6.5. The thrust is evaluated for all angles of attack below the stalling angle of the profile, and the optimal angle of attack for a given wind direction is stored.

In order to keep the experiment simple, most of the variables influencing the problem is set to be constant, except for the actual foil geometry. The variables are meant to be realistic values, and changing them is not expected to give large variations in the results. The necessary variables, used to build the complete wing model, is shown in table

Table 6.1: Non-geometry variables used in the foil experiment

Variable name	Value
Wing effective aspect ratio	10
Reynolds number	$6 \cdot 10^6$

These numbers are based on the dimensions of the wing sail given in table 1.2. An effective aspect ratio of 10 corresponds to a physical aspect ratio of 5, using symmetry to model the deck of the ship. The Reynolds number is based on an apparent wind velocity of 10 m/s.

### 6.3.1 4-Digit NACA Profile with Varying Thickness

First, the classical symmetric NACA 4-digit profiles where tested with varying thickness. The thinnest foil tested have a chord to thickness ratio of 0.1, while the thickest foil have a ratio of 0.2. The difference in thickness can be observed in figure 6.6. The geometry of the foils where generated with XFOIL's built in NACA profile generator.

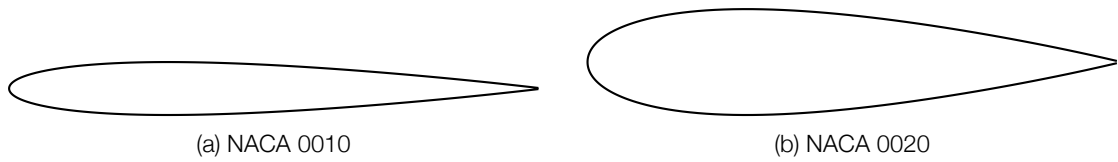


Figure 6.6: Thinest and thickest NACA 4-digit foils used

Two-dimensional lift and drag data for NACA 4-digit foil profiles with varying thickness can be observed in figure 6.7. NACA 0012 and 0014 have the highest lift coefficient, with NACA 0012 being slightly higher, but NACA 0014 have a more gradual stall characteristic (the lift drops more slowly after the maximum lift coefficient is reached).

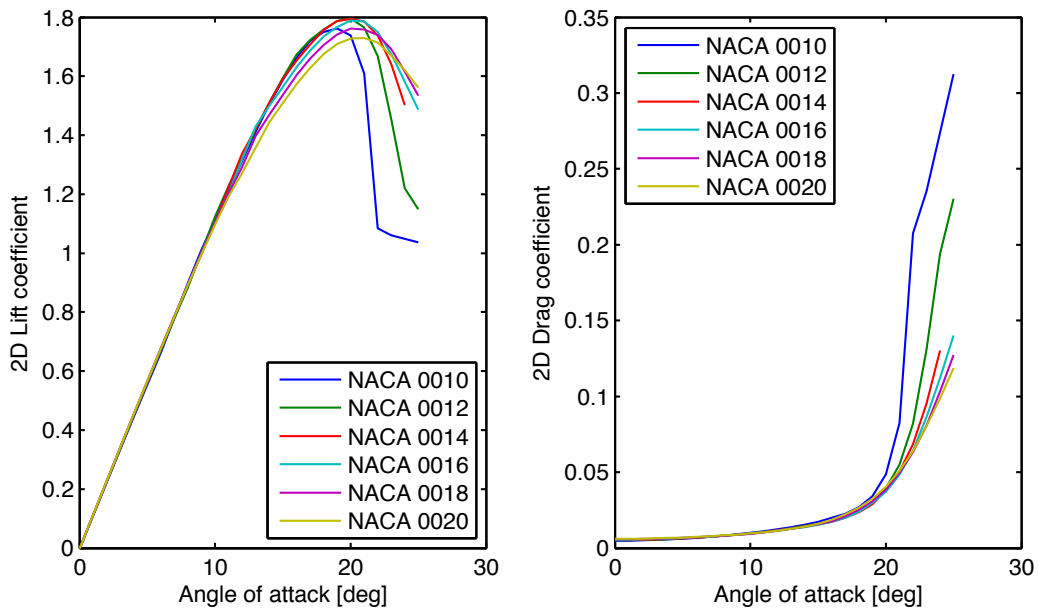


Figure 6.7: Lift and drag coefficients for different NACA 4-digit foil profiles

The thrust coefficient for different wind directions are then calculated based on the data, and plotted in figure 6.8



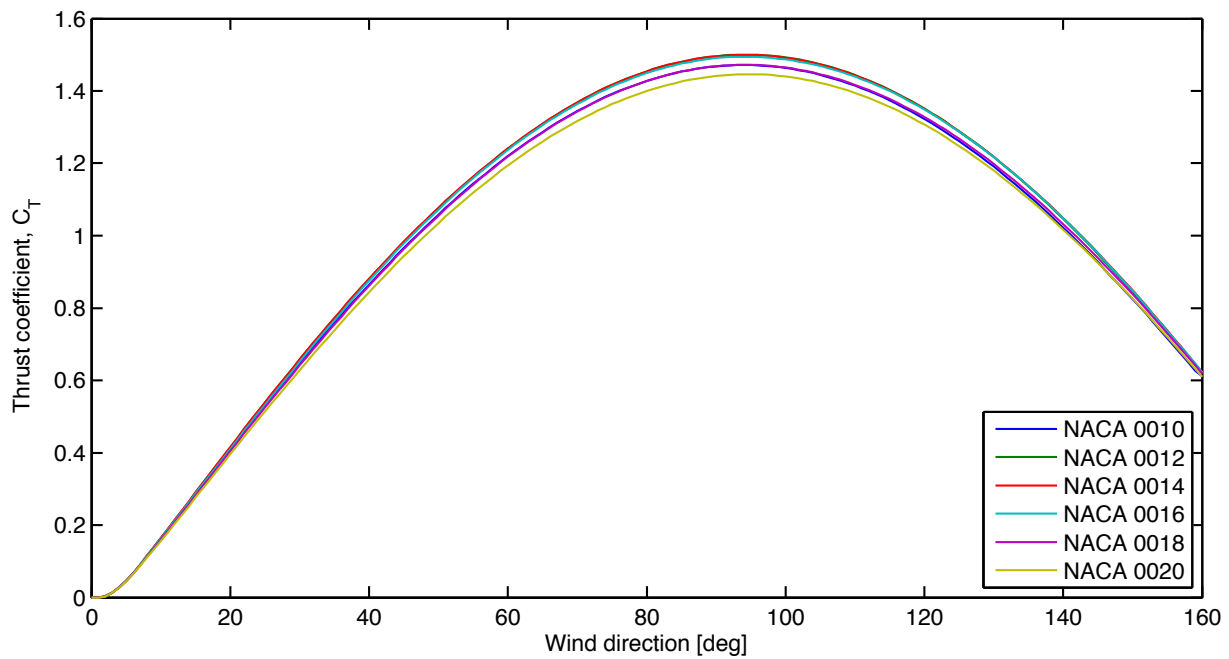


Figure 6.8: Thrust coefficient for different NACA 4-digit foil profiles

The result is that thickness have very little effect, but the two foil profiles with highest lift gives the highest thrust coefficient as well. Based on figure 6.8, NACA 0014 is picked as the best NACA foil.

### 6.3.2 Other Foil Profiles

Some other foils where also tested. In particular, a few low-reynolds foils where collected from reference [42]. They were picked based on personal interest from the author, while looking through all the symmetrical foils in the foil database in reference [42]. For one, Eppler, Joukowski and Wortmann are rather famous names when it comes to foil design. They all have many foils available in foil databases such as reference [42]. Another interesting feature is that they all have a slightly cusped trailing edge, which the NACA profiles do not have. The Atlantis foil is of particular interest as it is a foil specifically designed in order to be used for a wing sail. Details about the Atlantis project can be found in reference [9]. It must be mentioned that the Atlantis foil is specifically designed in order to be optimal for a much lower Reynolds number than 6 million (200 000 - 250 000), and low Reynolds number effects was a particular topic of interest for the authors. The bad results seen in figure 6.10 and 6.11 should therefore be considered with this fact in mind. That is, the Atlantis foil seems to be very bad at  $Re = 6$  million, but based on results from reference [9], the results for lower Reynolds numbers are quite different.

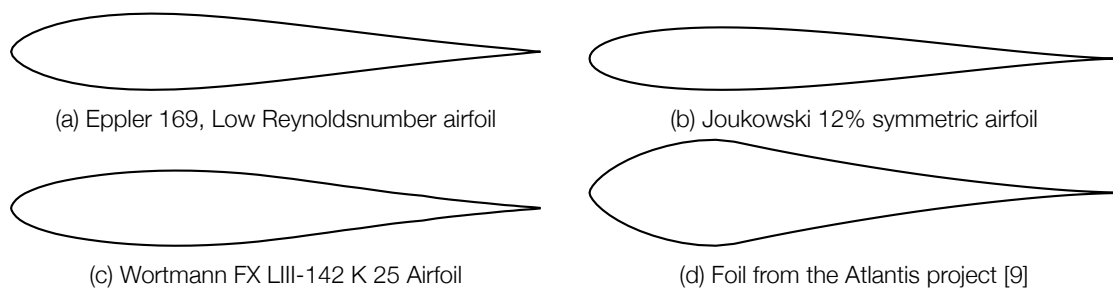


Figure 6.9: Non-NACA foils tested

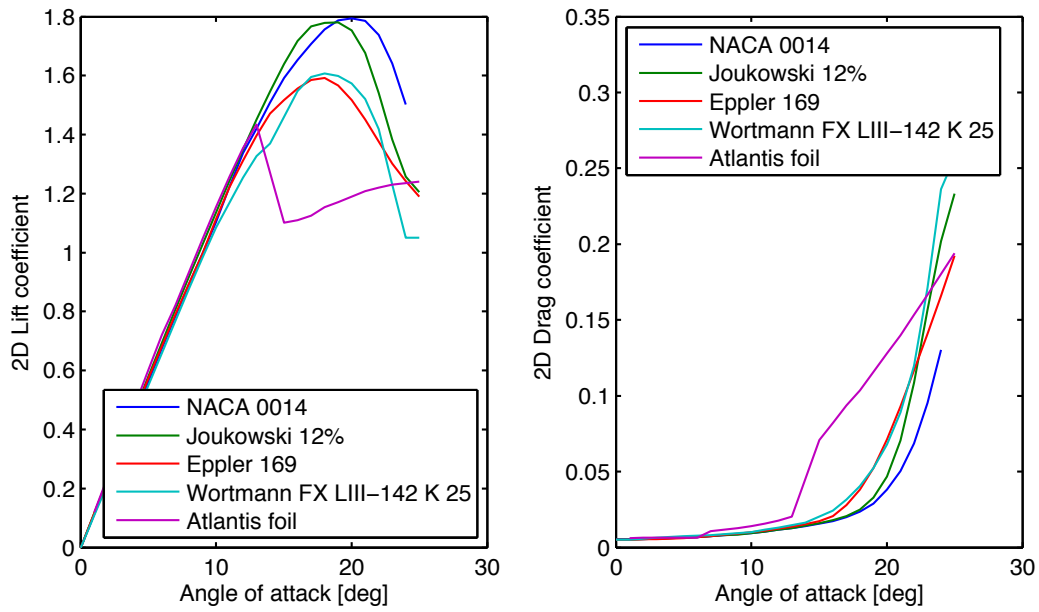


Figure 6.10: Lift and drag coefficients for different foil profiles

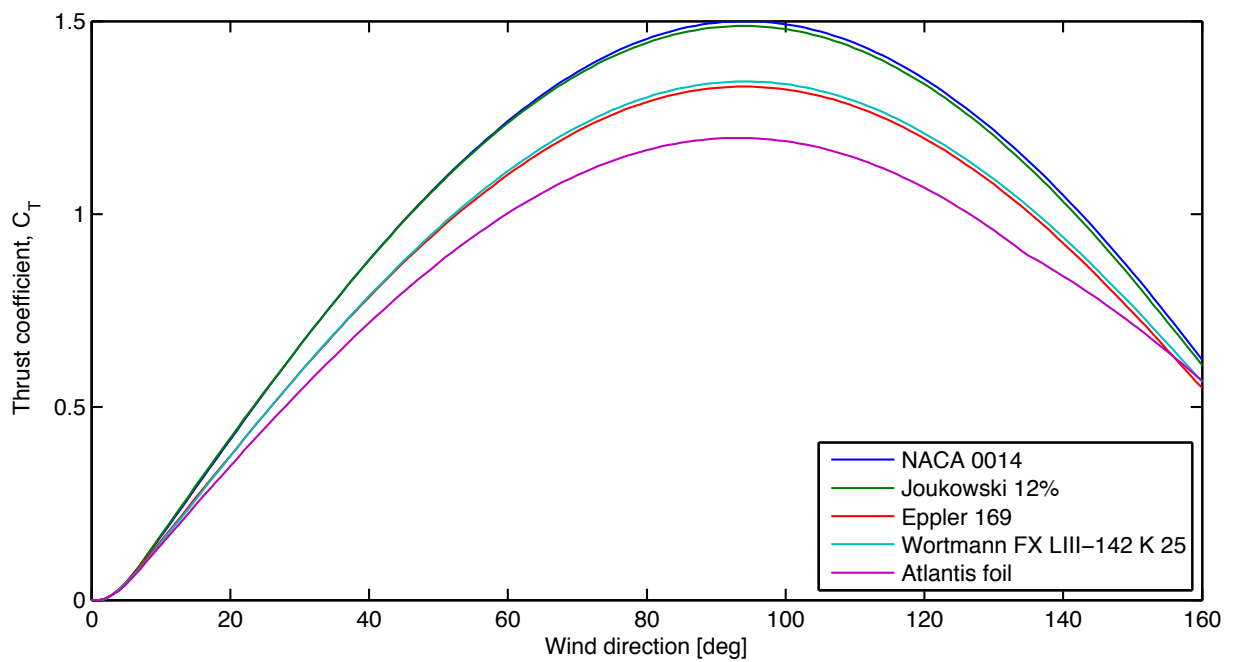


Figure 6.11: Thrust coefficient for different foil profiles

The results for this test was that non of the more special foils were able to achive better results than the NACA 0014 profile. Based on this, NACA 0014 has been used as a test foil in the rest of this project.

## 6.4 3D Effects

Three-dimensional effects are known to be very important for wing sails. Three-dimensional lift is significantly lower than Two-dimensional lift, and three-dimensional lift generates lift-induced drag. Two thing were tested

while using the custom BEM code developed for this project: The effect of having a rectangular wing, rather than having an elliptic wing, and interactions effects between 8 wing sails standing in a row.

### 6.4.1 Rectangular Wing

A rectangular wing is known to be worse than an elliptic wing. The lift is reduced, and the lift-induced drag is increased compared to an elliptic wing. The custom BEM code was used, together with viscous data from XFOIL for the NACA 0014 profile, in order to generate the plots shown in figure 6.13. The BEM simulation was performed with 36 panels in each chord-strip, 10 strips for the chemical tanker wings, and 20 strips for the Series 60 wing. The dimensions of the wings are given in table 1.2. The effective aspect ratio is twice the physical aspect ratio of the wings, in order to model the presence of the deck. Figure 6.12 shows the discretized panels for the Series 60 wing, along with the deformed wake.

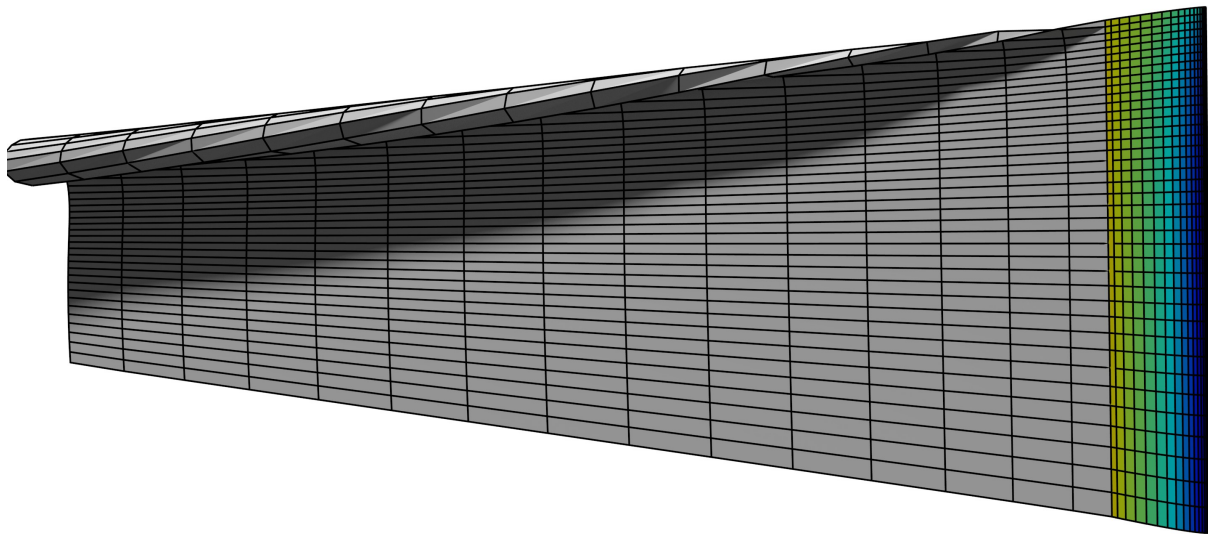


Figure 6.12: Illustration of the model used in the BEM simulation for single rectangular wing with physical aspect ratio = 5. The wake is deformed according to the final solution, and the colors on the wing corresponds to pressure values

The result of these simulations can be seen in figure 6.13. The lift for the chemical tanker wing are 65 % of 2D lift, while the lift for Series 60 are 78 %.  $\delta$  values for the two wings are 0.05 and 0.08 for the two wings respectively. The Series 60 wing show better performance than the chemical tanker wing, as expected. This is explained by the higher aspect ratio for the chemical tanker wing. figure 6.13 also shows the raw data from XFOIL for the two-dimensional foil profile, and values calculated as if the wing where elliptic. This is done for comparison. Although the difference in induced drag is small when one is comparing an elliptic wing with a rectangular wing, the difference in lift is significant. As lift is the most important property for a wing sail, this will also affect the thrust coefficient in a significant way.

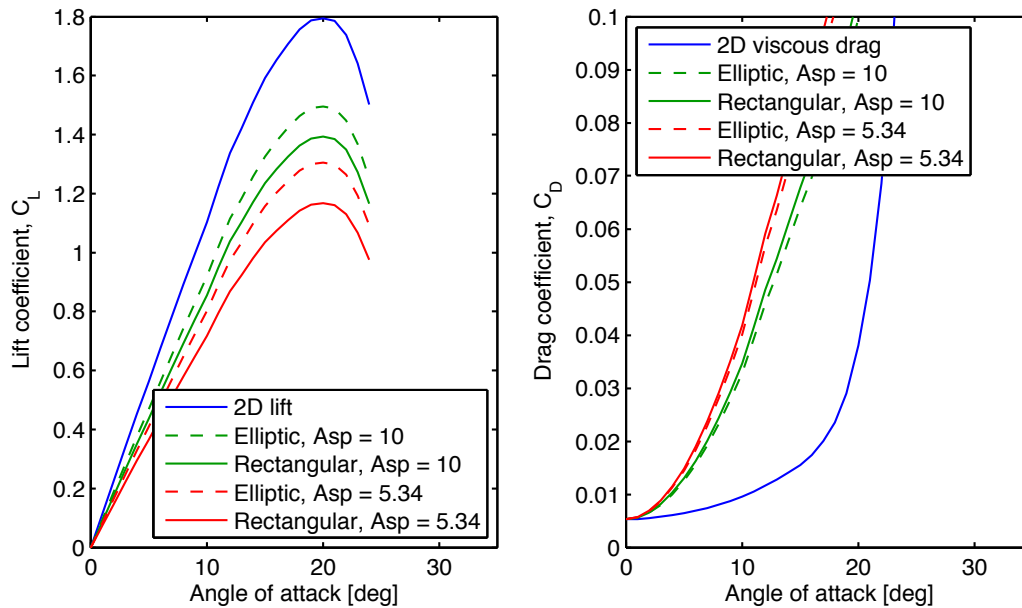


Figure 6.13: Lift and drag data for single wing, calculated with different methods

#### 6.4.2 Interaction Between Wing Sails in a Row

Interaction effects for wings standing in a row are important. In the same way that a single wing will experience induced velocities due to its own wake, several wings standing close to each other will experience induced velocities due to each others wakes. Before starting this project, it was suspected that this interaction effect could potentially be positive. This was because of a slightly confusing way interaction effects for *biplanes* (that is, planes with two wings on top of each other), often are described. For instance, reference [21] discusses different non-planar wing concepts (such as winglets, box wings, wing-tip fences, etc) that are intended to reduce the lift-induced drag. The general message of reference [21] is that non-planar wings are a good thing, as the non-planar wing concepts generally increases the span efficiency. A biplane is reported to have a span efficiency of 1.36, if the wings are located 0.2 span lengths away from each other. This sounds like a good thing. However, as reference [21] are talking about planes, and not wing sails, they can look at the interaction effects a bit differently: in reference [21], all the non-planar wings tested have the same span, and the same amount of lift. That is, the area of the non-planar wing is adjusted so that the amount of lift is constant. The span efficiency is then defined as the amount of lift-induced drag generated for a planar wing, divided by the lift-induced drag for a non-planar wing.

But if the total area of a biplane is at a similar size as for the planar wing (which it should be, depending on how much the lift is affected by interaction effects), the physical aspect ratio of each wing in the biplane, will be roughly twice the physical aspect ratio of the planar wing, as the chord length for two wings can be half the chord length of one wing, without changing the total wing area. So when reference [21] are reporting an effective aspect ratio of 1.36, it would correspond to each wing in the biplane only having an effective aspect ratio of  $1/1.36 = 0.74$ , if one is using the model in this report. That is, the interaction effect is actually negative for a biplane. The lift-induced drag coefficient for each wing is higher than it would be if each wing was standing alone, but from an airplanes perspective, the goal is to minimize the total drag, while keeping the lift constant, so that a biplane configuration actually could be positive after all.

Other reports, discussing interaction effects for actual wing sails, agree with this. The interaction effect is, in general, negative. For instance, reference [27] have tested wings experimentally and with Computational Fluid Dynamics (CFD). If the wings are kept at a constant angle of attack, three wings standing together will in general have a reduced thrust coefficient.

However, reference [27] also reports on another interaction effect that actually is positive. When several

wings are standing together, it seems that the stall angle increases. The optimal angle of attack for several wings are higher than the optimal angle of attack for just one wing. For instance, the optimal angle of attack for a single wing, based on experiments in reference [27], is 17 degrees. But if the wind is coming at an angle of 90 degrees, straight from the side, the optimal angle of attack for three wings are 18, 24, and 26 for the three wings respectively. That is, the optimal angle of attack increases, and it increases differently for the different wings. The total effect of this is that the thrust coefficient for wings standing together is only reduced by 4 %. For certain wind directions, such as  $\theta = 120$  the total interaction effect is positive, increasing the thrust coefficient with 15 %. The total interaction effect, including viscous effects, are therefore dependent on the wind direction.

The effect of increase in stall angle is not modeled in this project, due to limitations in the simulation software used. The custom BEM code cannot model the viscous phenomena that makes it so that higher stall angles are achieved. However, it can model the negative effect where the induced velocities are affecting the lift and drag of the wings. This was done for 8 wings in a row, for different wind directions. Both the chemical tanker wings and the Series 60 wings were modeled. The result of this can be seen in figure 6.15 and 6.16. An illustration, showing the simulation model used in the BEM code can be seen in figure 6.14

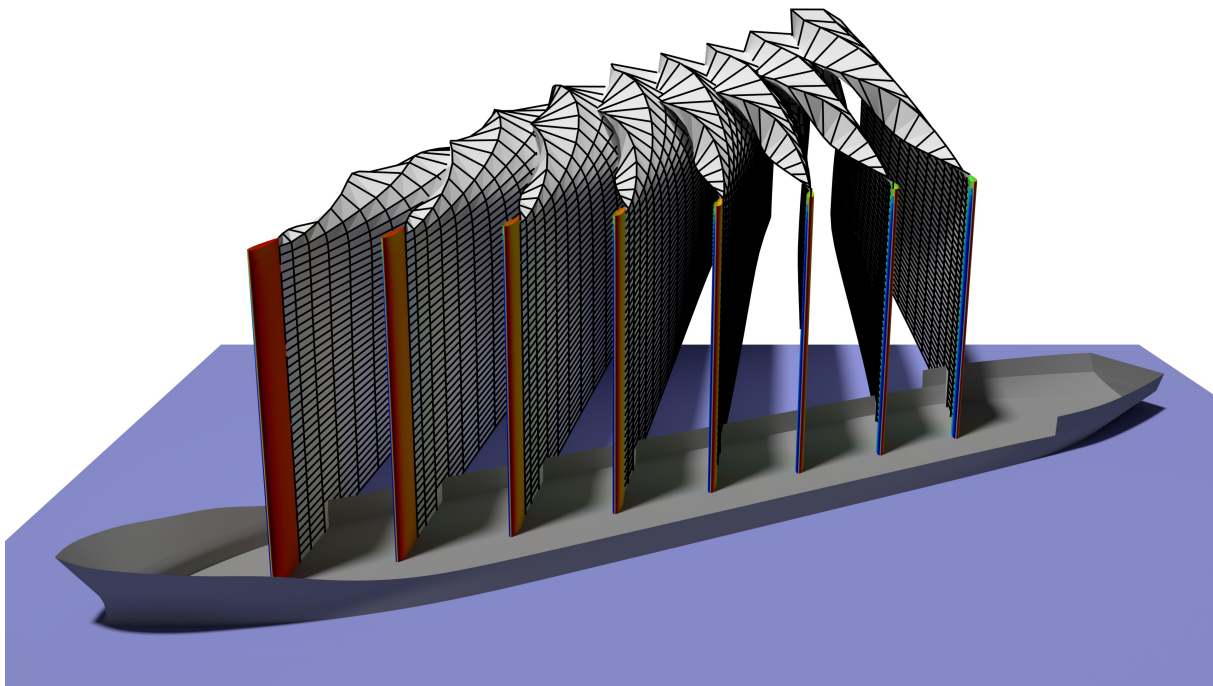


Figure 6.14: Illustration of the simulation model used in the BEM code in order to estimate interaction effects between wing sails standing in a row. The ship hull is included in the illustration, but was not directly a part of the simulation

The different wind directions were simulated by rotating the entire row of wings together, while keeping them facing the incoming velocity at a constant angle of attack. The effect of a different wind direction is to shift the wings relative to each other. The distance between the wings were kept constant, by assuming that the rotational axis is at 25% of the chord length from the nose of the wing.

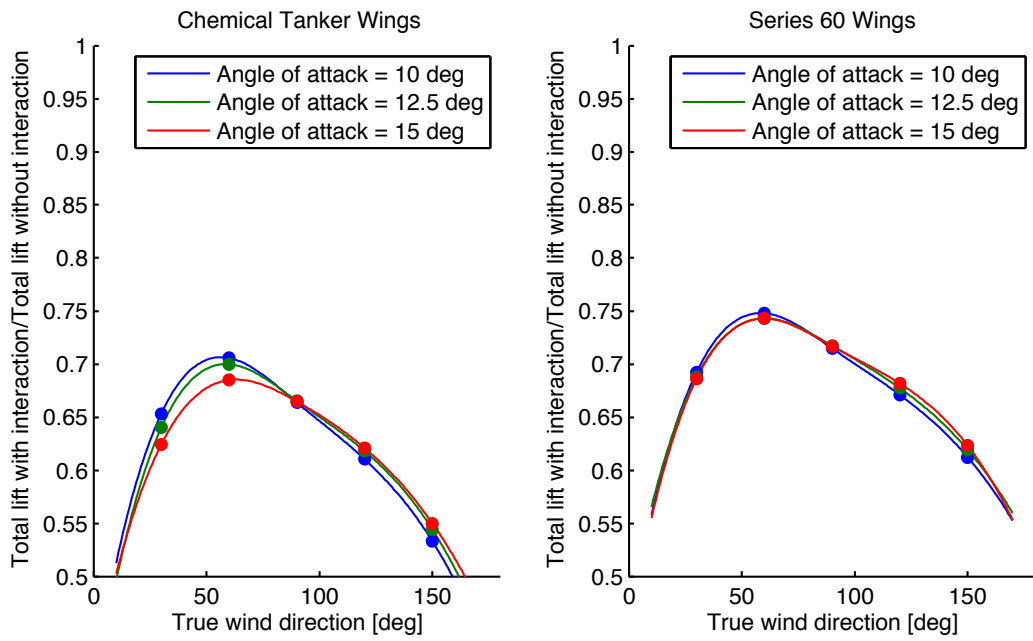


Figure 6.15: Interaction effects on lift for 8 sails in a row

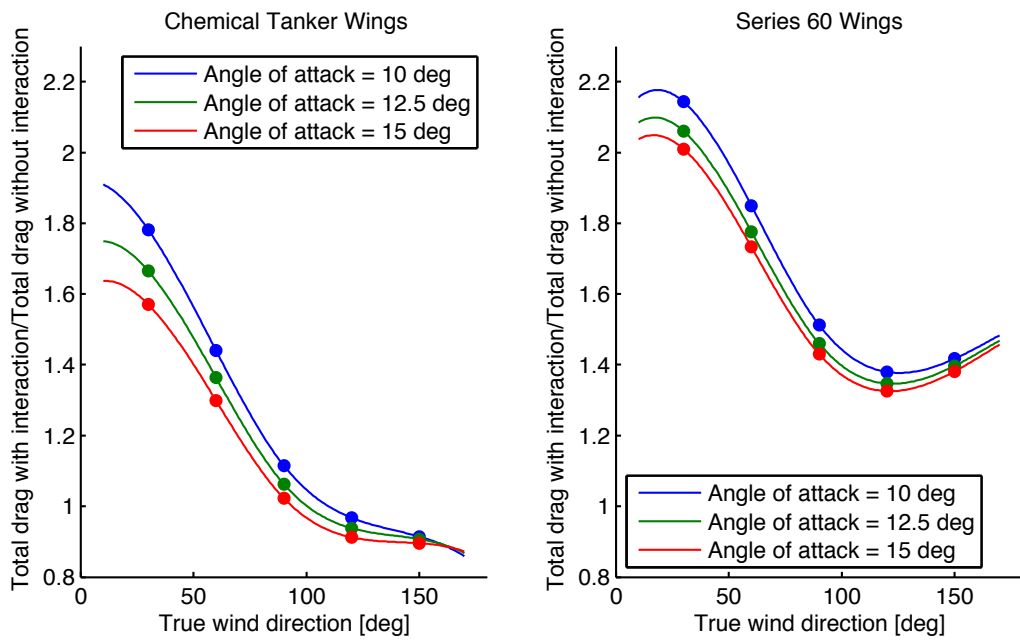


Figure 6.16: Interaction effects on drag for 8 sails in a row

As can be seen in figure 6.15 and 6.16, the predicted interaction effects by the BEM code is strong. The figures shows the forces calculated for wings in a row divided by the forces from a single wing alone. The lift is significantly reduced, and the drag is in general increased. This will reduce the thrust from the wings. The effect seems to be somewhat dependent on the angle of attack, but not very dependent.

## 6.5 Complete Wing Sail Model

---

Even though the viscous effect of higher stall angle has not been modeled properly in this project, the potential interaction effects is included in the final results, as a way of showing the importance of this physical phenomena. The viscous drag and lift is modeled using XFOIL, for the 2D foil profile NACA 0014.

In short, the complete wing model is the 2D data from XFOIL, where the lift is reduced due to the fact that it is a rectangular wing, and lift induced drag is included. The interaction effects are included as a function of wind direction. The interaction effect between the data points in figure 6.15 and 6.16 are found by spline interpolation in Matlab. This wing model is used to calculate the maximum thrust, and corresponding side force as a function of apparent wind direction and speed.

## Chapter 7

# Hydrodynamic Effects

When the wing sails are pushing the boat forward, there is also a significant side force and heeling moment. These effects from the sails must be balanced by the hull in order to have a steady motion in one direction. In this chapter, the scaled resistance results are presented, as a function of speed and yaw angle. Based on the literature review in chapter 2 and the experimental data that are available, the effect of heel is discussed.

### 7.1 Calm Water Resistance without Yaw

---

The scaled resistance without yaw for the two test ships are presented in this section. This resistance is used as a reference resistance later in this report. The result can be seen in figure 7.1. It is seen that the chemical tanker have significantly higher resistance, but this is also expected, since it is a larger ship.

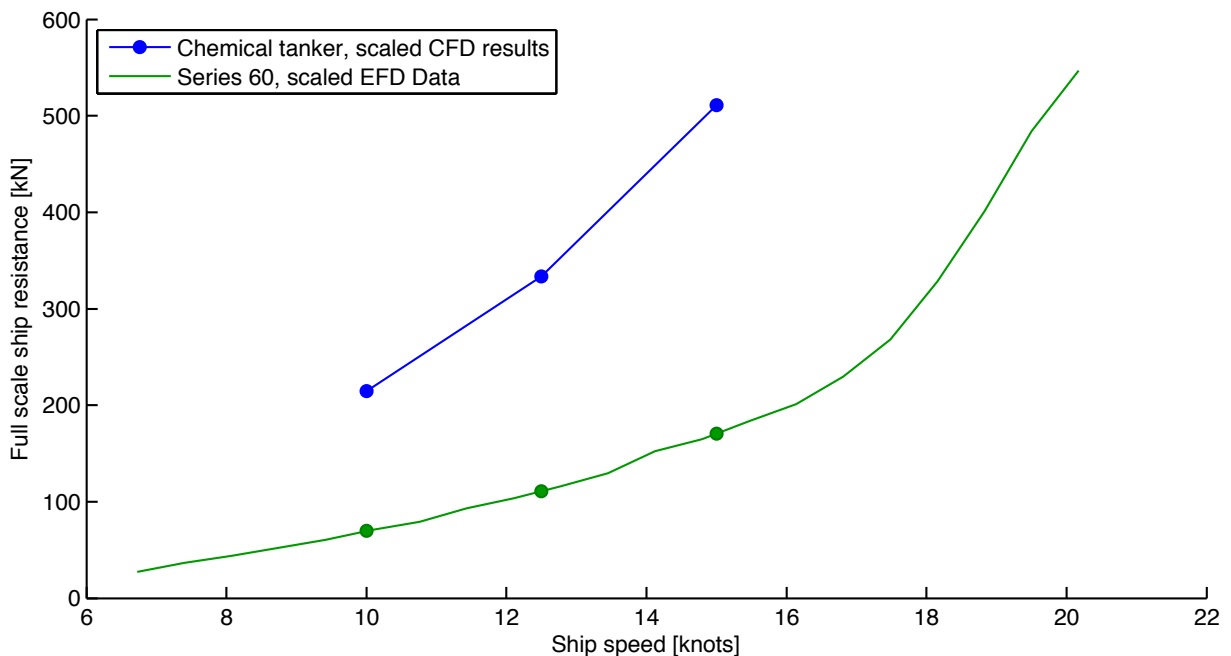


Figure 7.1: Full scale calm water resistance, at 0 deg yaw, for the two test ships, as a function of ship speed in knots



## 7.2 Effect of Heel

---

The effect of heel was one of the first things the author tried to test with CFD, as the exact effect of heeling a cargo ship was completely unknown. The result from one of these early CFD tests predicted that the resistance of the chemical tanker, at 10 degree heel was equal to 33.67 N. This simulation was done at such an early stage, that the time step in this simulation was too high, and it therefore predicts too high resistance as well, but a simulation performed without heel, with the same (wrong) time step predicted a resistance of 33.54 N. That's a difference of less than 1%, so the author was worried that the simulation was executed wrong (which it actually was, but this is also beside the point for this discussion). Steffen Hasfjord was therefore asked to perform a few extra experiments while doing his own tests in the towing tank at The Norwegian Marine Technology Research Institute (MARINTEK), too which he kindly agreed. The result from these tests can be seen in section 5.1.2. The difference between 0 degree heel and 8.8 degrees is also here less than 1%, and in fact, the difference is smaller than the difference between the first test at 0 degrees and the second. Based on this, it was concluded that heel actually have very little effect on the resistance of a cargo ship, at least for heel angles that are roughly less than 10 degrees. This conclusion was further supported by the "Geitbåt" experiments, in section 5.1.3, which shows the same tendency for the traditional boat design.

However, this is not completely fitting with the statements in the yacht literature, discussed in chapter 2. The yacht literature claims that the resistance due to heel is somewhere between 2% and 7%. Yachts have not been tested in this project, but it is possible that yachts have significant added resistance due to heel, even if cargo ships and traditional sailing boats have practically no increase in the resistance. Modern yachts have significantly different hull forms compared to both cargo ships and the "Geitbåt", often having wide transom sterns. If this wide transom stern are heeled into the water, it might create added resistance. This theory has not been tested, but there are at least reasons to believe that there is a difference between sail yachts and cargo ships, based on the results seen so far.

The effect of heel is not modeled in the final result, because it is considered to be of very little importance, based on both CFD and experiments.

## 7.3 Effect of Yaw

---

The yaw was expected to create a lot of added resistance. In order to visualize this, the resistance at a given yaw angle was divided by the resistance without yaw. The result can be seen in figure 7.2. At the highest yaw angles, 8 and 10 degrees for the chemical tanker and the series 60 respectively, the added resistance is as high as 70% of the resistance without yaw, which is a lot. The added resistance for the chemical tanker must be considered with the knowledge that the CFD setup used in this project might predict too high resistance due to yaw, but since the Series 60 Experimental Fluid Dynamics Data (EFD Data) shows the same tendency, it is safe to conclude that yaw angles have an significant effect on the resistance. The data for the series 60 shows some bumps and kinks, which is believed to be explained by the fact that the EFD Data have some uncertainty. The increase in resistance due to yaw should be a smooth function, although the non-linear behavior is fitting with the general behavior of lift-induced drag for lifting surfaces.

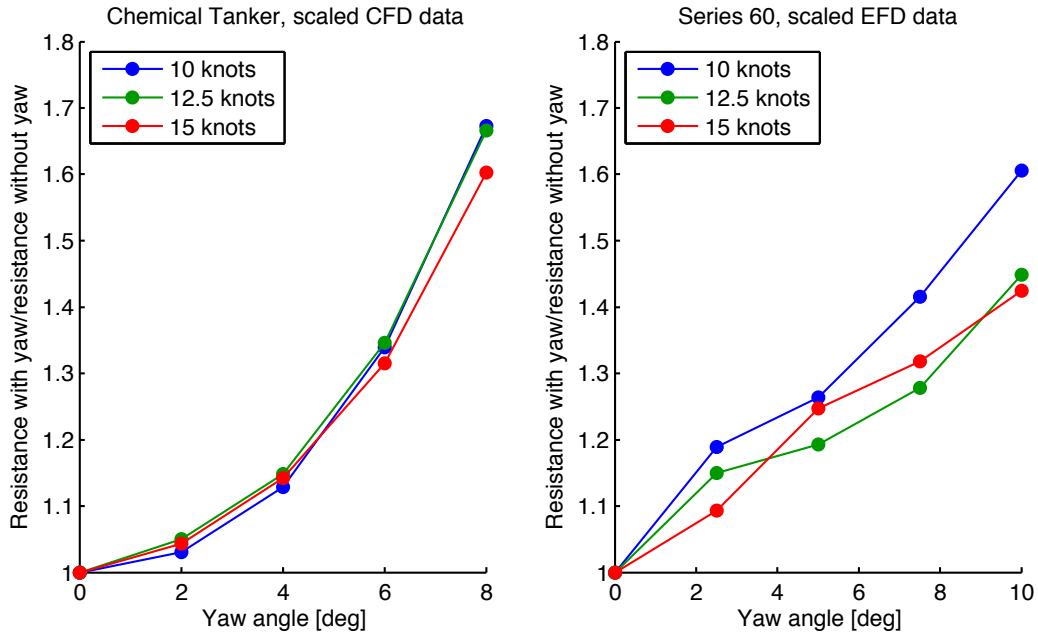


Figure 7.2: Resistance due to yaw, as a function of yaw angle

The side force on the two ships are very different in magnitude. The side force relative to the ship resistance without yaw is almost twice as high for the Series 60 as for the chemical tanker. Although the predicted lift from CFD is expected to be a bit too low, this is a very large difference, which might be due to the difference in geometry as well. The Series 60 hull seems to be a more efficient lifting surface. The side force for both ships are definitively non-linear, which is different from traditional wings. In general, wings have a linear behavior for small angles of attack, but both CFD and EFD Data shows non-linear behavior at relatively small yaw angles.

This can be explained by "cross flow drag". The flow around the bottom of the ship hull can separate, and create side force due to drag effects, rather than normal lift. A model for cross flow drag can be written as follows:

$$F_2 = \frac{1}{2}\rho \left( \int_L C_D(x)D(x)dx \right) |U|^2 \sin \beta | \sin \beta | \quad (7.1)$$

This model is taken from reference [11].  $U$  is the ship velocity,  $\beta$  is the yaw angle,  $D$  is the draft of the ship, while  $C_D(x)$  is a drag coefficient. The basic idea is to use the part of the incoming flow velocity that is normal to the longitudinal axis of the ship as a reference velocity. If this model is correct, there is clearly a non linear behavior in cross flow drag. This can be seen easily if one considers small yaw angles. Then,  $\sin \beta \approx \beta$ , so that the expression for cross flow drag can be considered to be  $\propto \beta^2$ . The side force at very small yaw angles (less than 5 degrees) seems to be almost linear for the EFD Data. The question of how important the effect of yaw is, does depend on how large the yaw angle due to a wing sail will be. If the yaw angles are small enough, the added resistance due to yaw might not be so important after all. See chapter 8 for more on this.

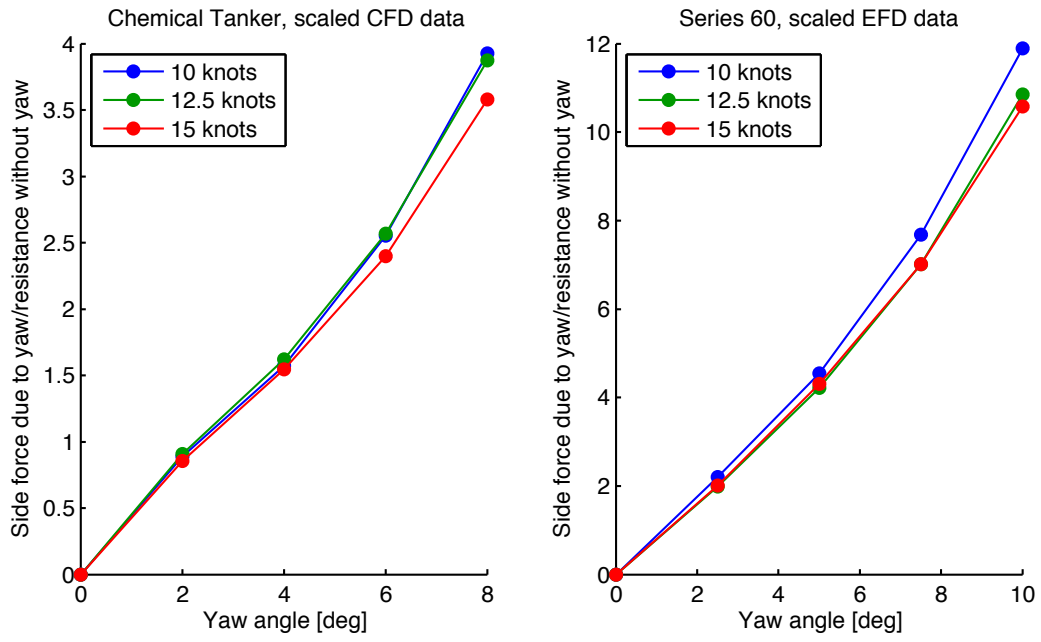


Figure 7.3: Side force due to yaw, as a function of yaw angle

## Chapter 8

# Results

This chapter contains the overall final results, based on the methods developed, and the data collected, in previous chapters. First, the forces from the wings are calculated, with a model that assumes rectangular wings, viscous data from XFOIL and includes interaction effects for 8 wing sails in a row, as a function of angle of attack. These force coefficients are then used to calculate the thrust and the side force on the ship for different speeds and wind directions. The yaw angle necessary for balancing the side force from the wings are found, and the added resistance due to yaw is calculated, and subtracted from the wing thrust in order to find the effective thrust. Maximum KG values are calculated based on the requirement that the hydrostatic restoring moment from the ship must balance the heeling moment from the wings. The velocity while using wings as the only thrust generator is predicted, and lastly there is a plot showing the lift to drag ratio for the hulls compared to a "normal" wing. The last graph are used for the discussion in chapter 9.

### 8.1 Forces From Wing Sails, including Interaction Effects

---

This section shows the thrust and side force coefficient for the two different wing sail setups, calculated with different methods. The force coefficients predicted by taking the rectangular shape of the wing, and the interaction effects, into account are much lower than the lift coefficient predicted by elliptic wing theory. This illustrates that elliptic wing theory might be too optimistic when modeling wing sails. Real wings must be modeled.

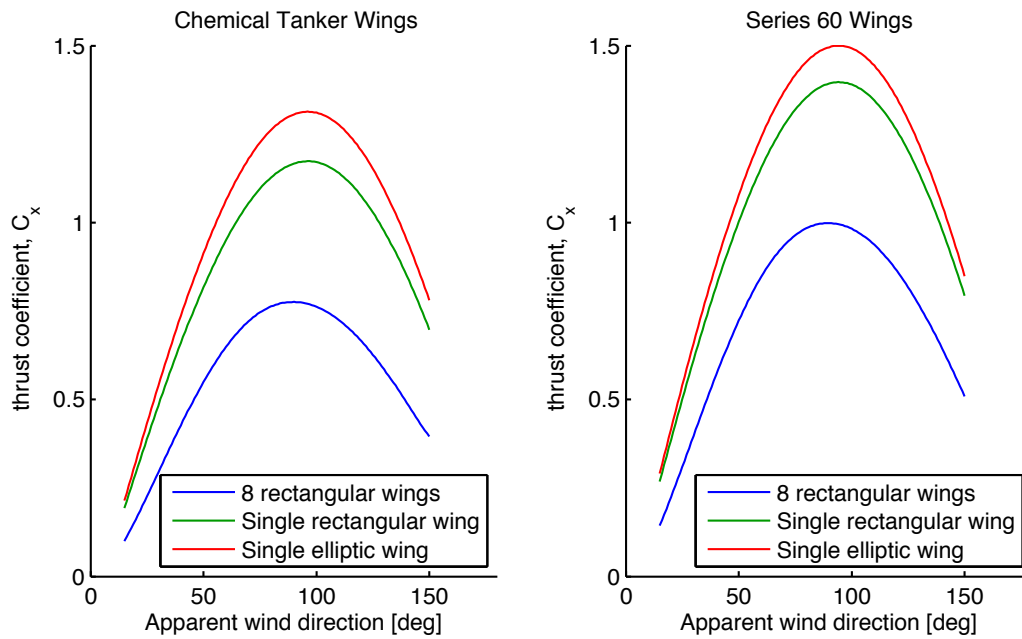


Figure 8.1: Maximum thrust coefficient for the wing sails for the different ships, calculated with different methods. The chemical tanker wings have physical  $Asp = 2.67$  while the Series 60 wings have physical  $Asp = 5$

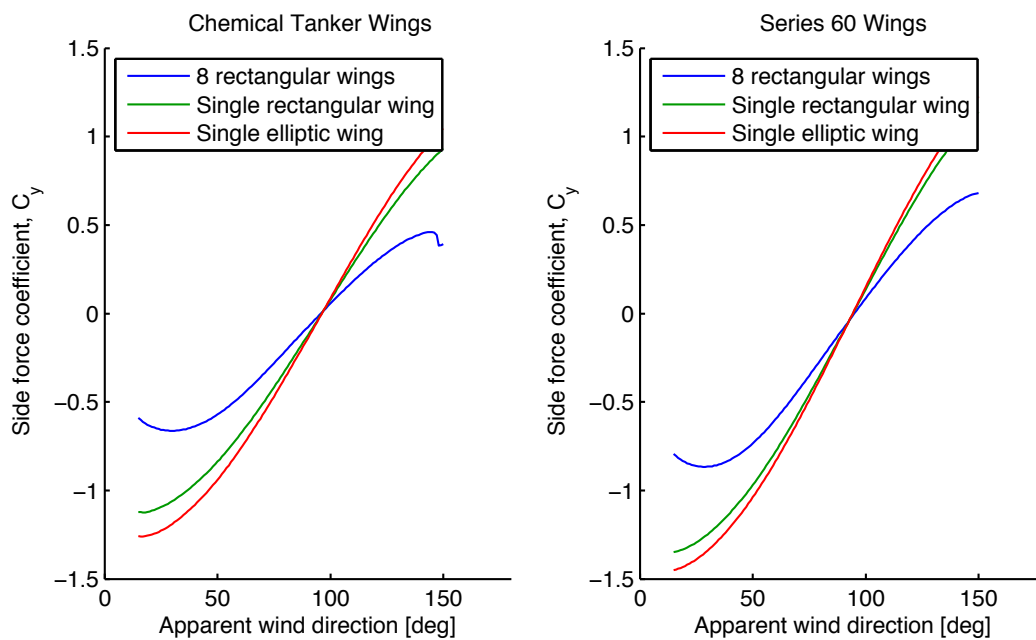


Figure 8.2: Side force coefficient for the wing sails for the different ships, calculated with different methods. The chemical tanker wings have physical  $Asp = 2.67$  while the Series 60 wings have physical  $Asp = 5$

## 8.2 Using Wing Sails as Auxilary Propulsion

---

This section shows the final predicted values for effective thrust, predicted yaw angles and maximum value of KG for the two ships.

### 8.2.1 Chemical Tanker

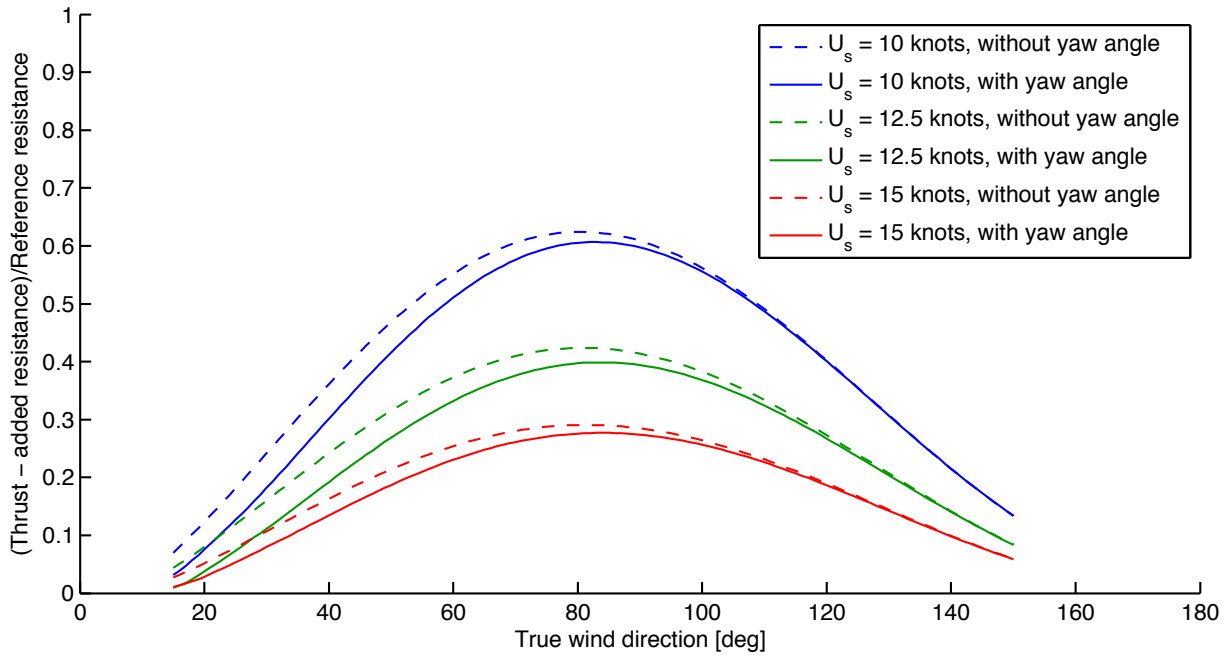


Figure 8.3: Effective thrust from wing sails on the chemical tanker, with and without hydrodynamic effects from yaw.  $U_s$  is the ship speed, and the wind speed is set to be 7 m/s

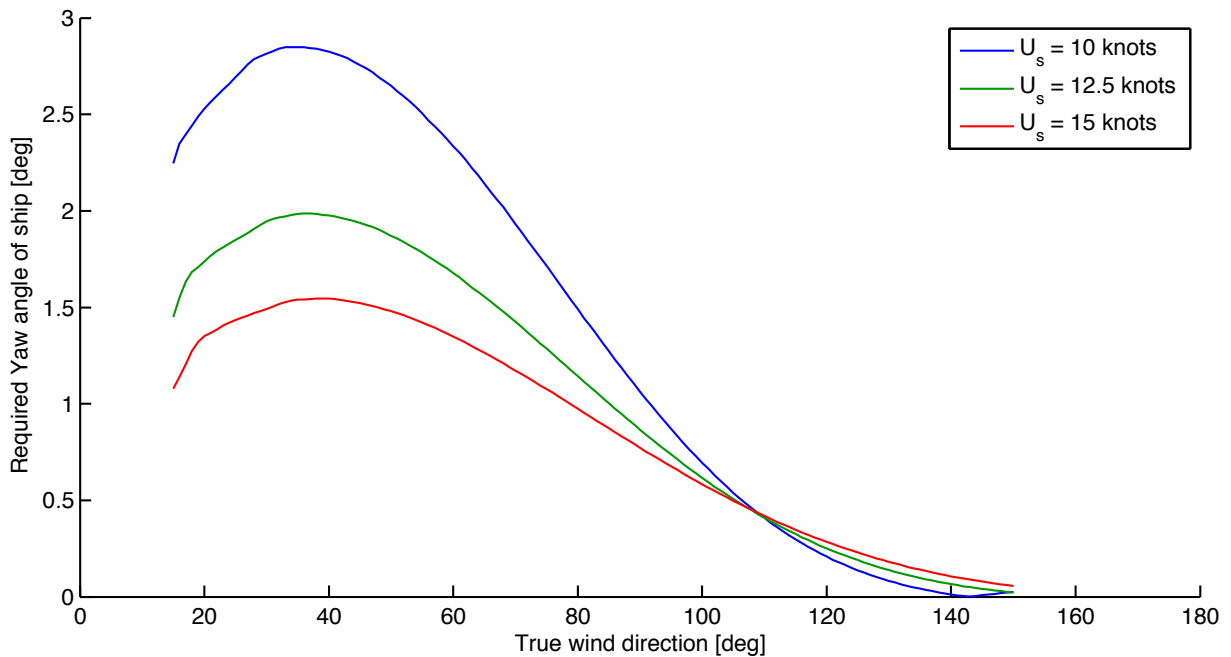


Figure 8.4: Required yaw angle of the chemical tanker, in order to balance the side forces from the wing sails.  $U_s$  is the ship speed, and the wind speed is set to be 7 m/s

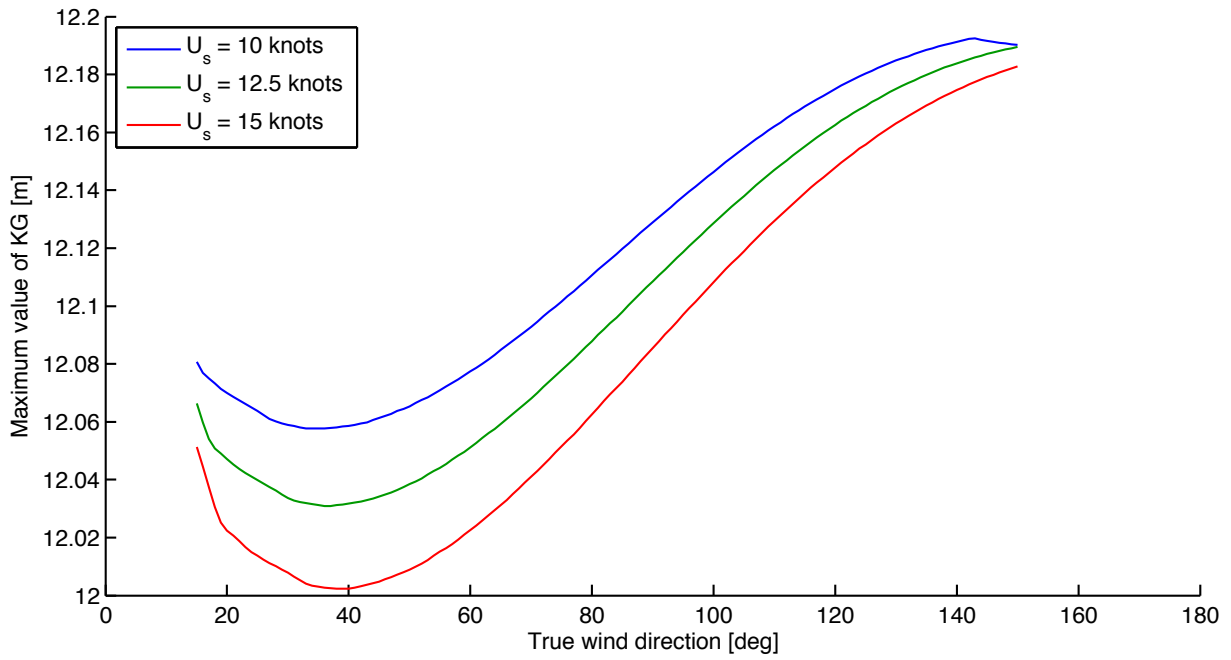


Figure 8.5: Maximum value of KG for the chemical tanker in order to withstand the heeling moment generated by the sails, at 10 deg heeling angle.  $U_s$  is the ship speed, and the wind speed is set to be 7 m/s

### 8.2.2 Series 60

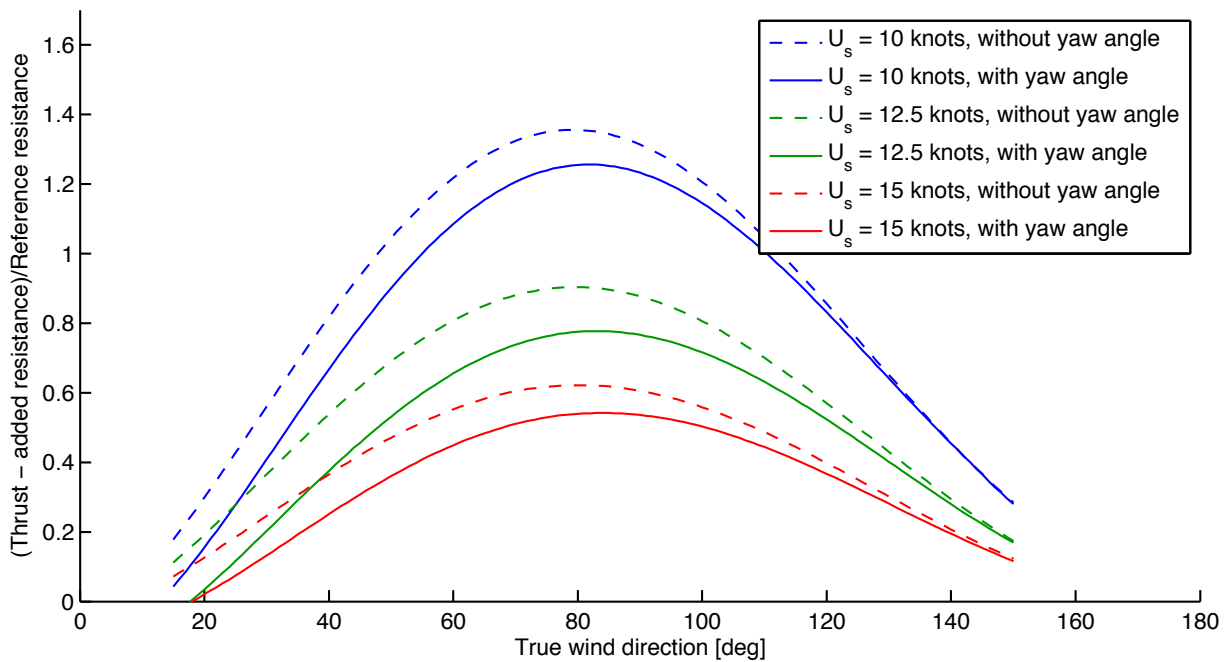


Figure 8.6: Effective thrust from wing sails on Series 60, with and without hydrodynamic effects from yaw.  $U_s$  is the ship speed, and the wind speed is set to be 7 m/s

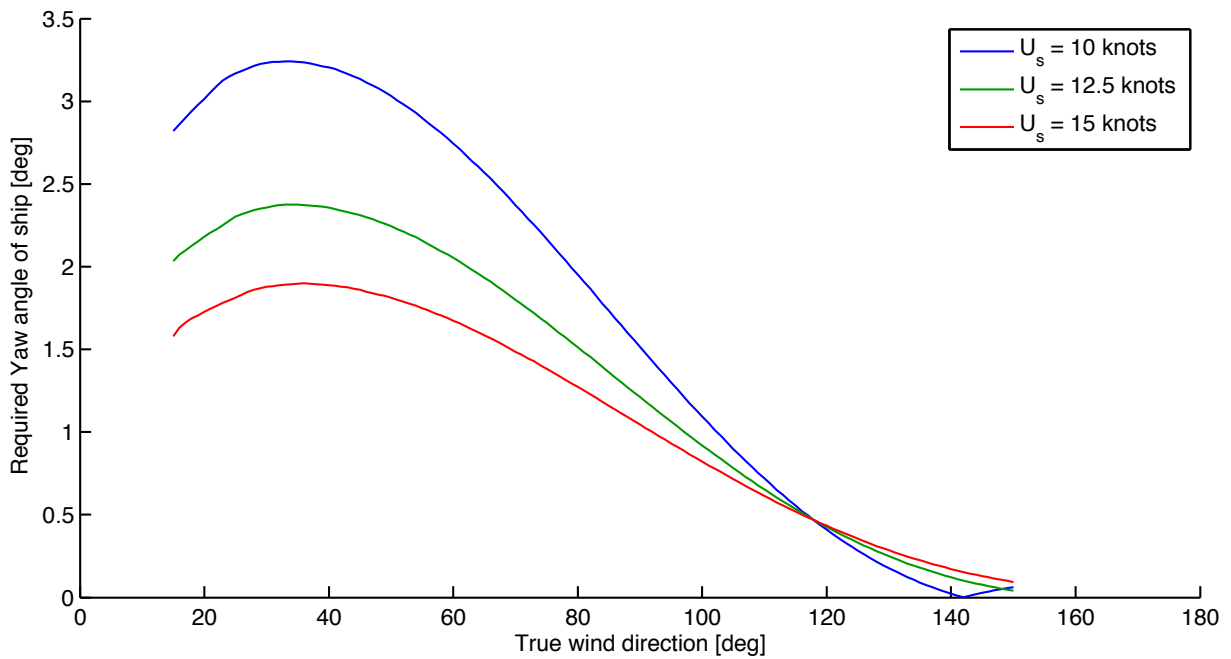


Figure 8.7: Required yaw angle of Series 60, in order to balance the side forces from the wing sails.  $U_s$  is the ship speed, and the wind speed is set to be 7 m/s

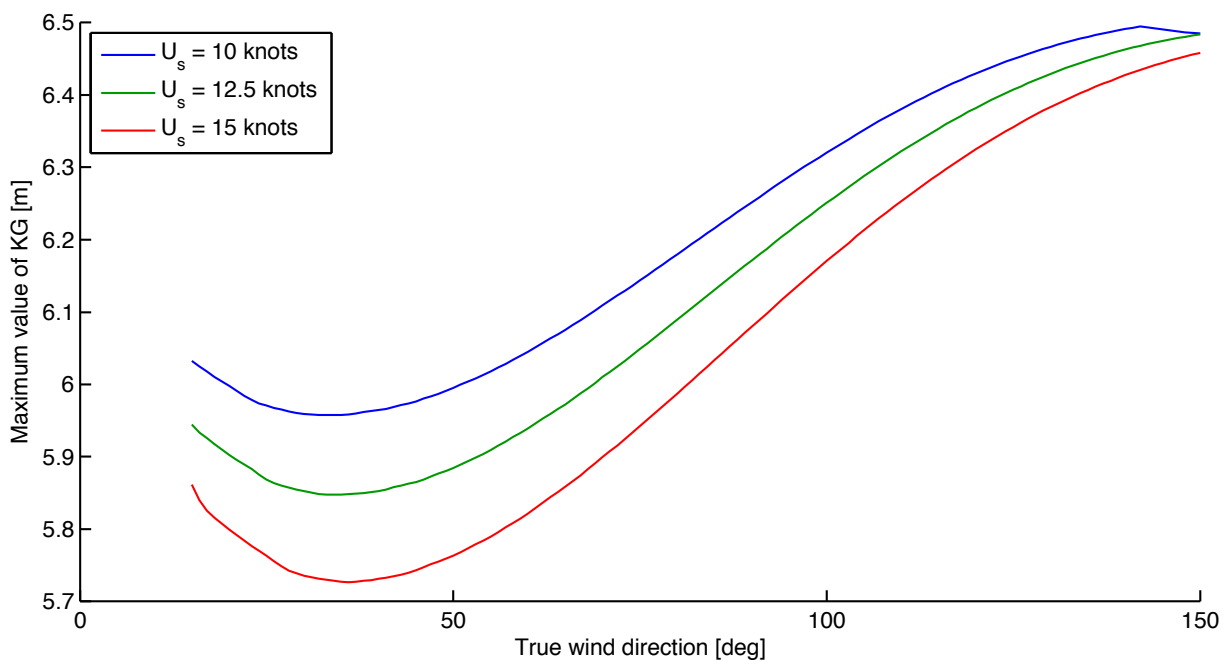


Figure 8.8: Maximum value of KG for Series 60 in order to withstand the heeling moment generated by the sails, at 10 deg heeling angle.  $U_s$  is the ship speed, and the wind speed is set to be 7 m/s

### 8.3 Using Wing Sails as the Only Propulsion

This section shows the predicted velocity for the Series 60 hull using wing sails as the only form of propulsion. Since resistance data for the series 60 hull is not available for speeds lower than roughly 6 knots, not all wind directions were possible to test. It is expected that the speed will drop rather quickly for near-head-wind conditions.



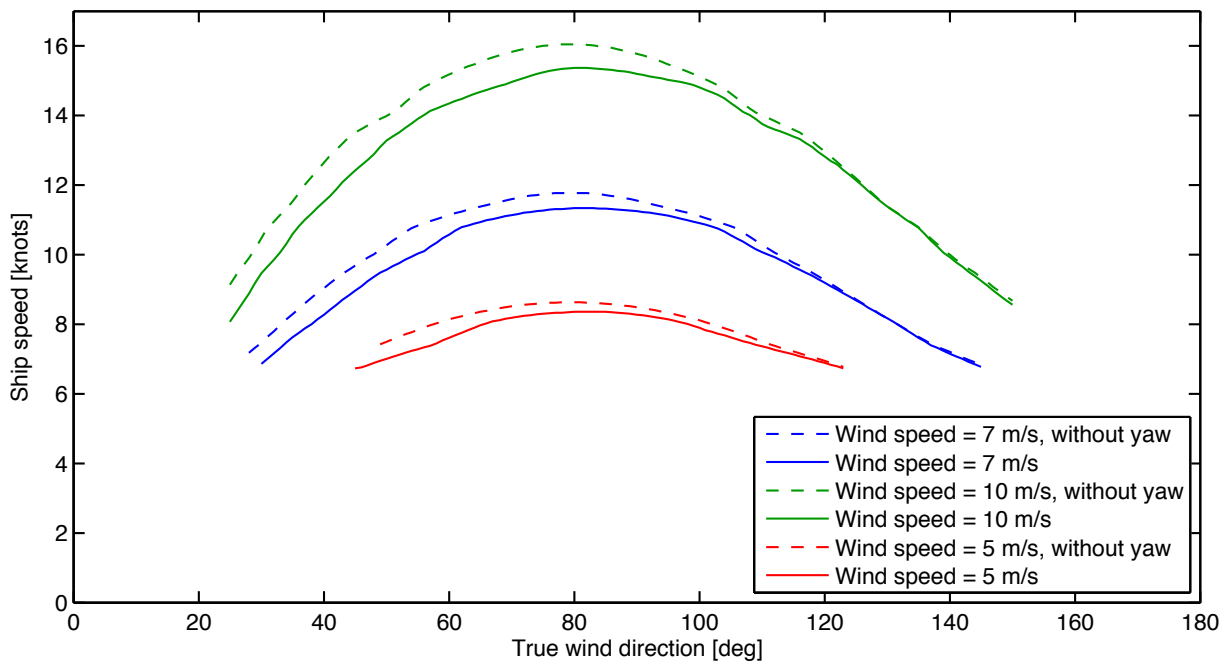


Figure 8.9: Maximum ship speed, while using wing sails as the only form of propulsion, for different wind speeds. Based on Series 60 data

## 8.4 Comparing Hulls to Wing

As a final result graph, the lift to drag ratio is plotted for the two hulls, and compared to a "normal" elliptic wing with aspect ratio equal to 5. This is to illustrate that the ship hulls are indeed "bad" lifting surfaces. The lift to drag ratio is much smaller for the two ship hull than for the wing

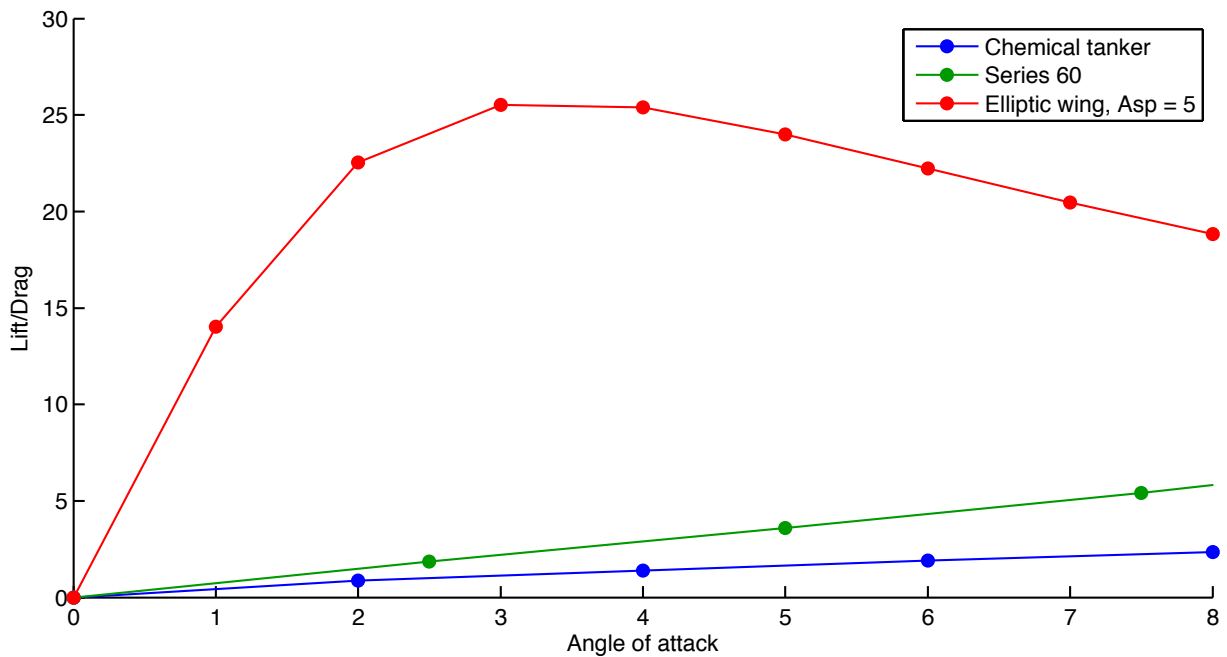


Figure 8.10: Lift force divided by drag force, for the two test hulls, and a theoretical elliptic wing, with viscous effects modeled by XFOIL

## Chapter 9

# Final Discussion and Conclusion

The main objective of this project was to get an overview of the important physical effects involved when using wing sails on a ship. Several physical effects have been explored. Everything has not been covered in its entirety, however, the results from this project gives an idea of the effects that were in question before starting this project. The project has also given the author experience in several modeling methods. Both Boundary Element Method (BEM) and Computational Fluid Dynamics (CFD) have been used, together with Experimental Fluid Dynamics Data (EFD Data) from external sources. Data from these different sources has been put together, in order to model a ship, using wing sails as propulsion.

Two test ships were chosen, namely a chemical tanker, and the Series 60 container ship. Many, and large wing sails have been used. Specifically, eight wing sails in a row, that nearly covers the entire ship in sails. This probably gives an optimistic value of the total sail area, but this was done in order to give large forces from the sails, so that potential hydrodynamic effects should be large as well. Realistic wind speeds were assumed, based on wind maps covering the coast of Norway (figure 1.5). For the Series 60 ship, the sail setup was enough to drive it at realistic cargo ship speeds. More than 10 knots could be reached, with a wind speed equal to 7 m/s (figure 8.9). Keeping the ship speed constant at 12.5 knots, the wing sails could give effective thrust equal to 78% of the total resistance of the ship (figure 8.6). The story for the chemical tanker is a bit different. At 12.5 knots, the wing sails give an effective thrust equal to 40 % of the resistance (figure 8.3).

One reason for the large difference between the two ships can probably be due to difference in resistance. The Series 60 ship is a slender small ship, while the chemical tanker is a wide, large ship. However, another important difference is the difference in thrust coefficient for the different sail setups. The Series 60 wings, having higher aspect ratio, gives significantly higher thrust coefficients (figure 8.1). The thrust coefficient is very dependent on the lift coefficient, which again is very dependent on aspect ratio. The drag is also dependent on the aspect ratio, but drag is less important for the thrust coefficient, as the drag force is much smaller in value. Potential modeling of lift, done with the custom BEM code developed for this project, predicts strong interaction effects between wing sails standing in a row. In fact, the interaction effect predicted by BEM is purely negative. It decreases the lift, with as much as 45%, and increases the drag, which as much as 80% (figure 6.15 and 6.16), which again decreases the thrust (figure 8.1). It seems that some of the negative interaction effects can be countered by the fact that higher angles of attack can be reached without stalling for the sails standing in a row, compared to sails standing alone, however this effect has not been investigated properly in this project.

A wing sail that creates thrust will also, depending on the wind direction, create side force and heeling moment. This side force must be balanced by the hull. Before starting this project, it was expected that this would be a challenge. Creating lift from any lifting surface has the negative effect of also creating lift-induced drag. As a normal ship hull has a very low aspect ratio and a high width, compared to wings, it was expected that the added resistance due to yaw would lower the effective thrust from the wing sails considerably. This was found to not be entirely correct. Side force from the wings will not create large yaw angles, at least not for most wind directions.

Both Series 60 and the chemical tanker experiences a maximum yaw angle at around 3 degrees (figure 8.4

and 8.7) in the test cases used in this project, but only when going high up against the wind (almost head wind). For most wind directions the yaw angle is much smaller. For such small yaw angles there will not be much added resistance on the ship (figure 7.2). This suggests that typical sailboat devices such as keels have little influence on the effective thrust from the wing sail, and these devices were therefore not modeled in this project.

The largest side forces from the wings are slightly larger than the the largest thrust (figure 7.3), and comparing lift to drag ratios for the two test hulls to a normal wing (figure 8.10) suggests that this would be a problem. But then again, water is almost a thousand times denser than air, which means that even an inefficient lifting surface can create much lift force without having a large yaw angle. In addition, the induced resistance due to side force/lift is roughly proportional to the yaw angle squared. A small yaw angle will therefore mean a small added resistance due to yaw. The added resistance due to yaw will not be a problem without large yaw angles. The difference in effective thrust at wind directions close to head wind is however relatively large. The problem is that the wing sails can not produce large amounts of thrust at these wind directions anyway, so the consequence of yaw seems small. If one wishes to improve the performance of a sail boat while going high up against the wind, yaw will definitely be a problem. If for instance the thrust from the wing sail is considerably increased, by using some form of high-lift device on the wing (flaps, leading edge slots, etc), the side force will quickly become a problem. The higher the thrust at these wind directions, the higher the side force. It is also seen that the effect of yaw seems to be larger for the Series 60 hull than the chemical tanker. This is explained by the fact that a larger amount of thrust is produced for the Series 60 hull, compared to the resistance of the ship. The Series 60 hull has more wing power, so the problem of yaw becomes larger.

Based on this, there seems to be two things one can do in order to improve the wing sail technology: focus on increasing the thrust from the wing sails, and reducing the resistance of the hull. Having a hull with low resistance and a lot of wing sail power would be great. However, this would lead to a situation where the effect of yaw could more severe, so that devices such as keels could be necessary. In other words, even if the yaw effects are not severe for the ship hulls in this project, the effects should be much more important if one tries to take advantage of the wind power at almost-head-wind conditions, as this would demand more powerful wings, which would create larger side forces.

The heeling moment from the wing must be countered with a restoring moment from the ship hull. In this project, only simple stability calculations are done, using initial stability theory, and neglecting many practical considerations, regarding traveling with a heeling angle. The necessary maximum distance from the keel to the center of gravity in order to counter the heeling moment from the wings, at 5 deg heel angle, is calculated (figure 8.5 and 8.8). These calculations suggest that stability is not a big problem for the chemical tanker, however slightly problematic for the Series 60 ship. The chemical tanker can have the center of gravity above deck for all wind directions, while Series 60 must have the center of gravity below deck for all wind directions in order to maximize the thrust from the wing sails. Stability can be increased by having a wider hull, however, this will also increase the resistance. As the power of the wind, extracted by wing sails, are a scarce resource, high stability could come with a large negative consequence as well. In addition to this it is important to consider that wing sails will raise the center of gravity as they are structures that necessarily must exist above the deck. This could lead to stability problems.

The hydrodynamic effect of going with a heel angle is found to not be negative. Both model tests, and CFD results suggest that there is practically no increase in resistance when going with a heel angle alone (section 7.2). This is slightly contrary to the yacht literature. It might be the case that normal sailing boats, having a wide transom stern, create more resistance while going at a heel angle than a cargo ship. However, normal sailboats probably have the wide transom stern for a reason. It should for instance help with the stability of the ship in an effective manner. It could be the case that increasing the stability will increase the consequence of heel.

A large portion of the time spent on this project was used on developing the custom BEM code. This BEM code can model lifting surfaces using constant value potential panels and a potential wake. Through comparison with experimental data for lift, and theoretical values for lift induced drag, the code was validated to predict accurate results for a single wing. The pressure distributions around wings and cylinders are compared to theoretical and other numerical results, and they show good agreement. The speed of the code was significantly increased through the use of Open Computing Language (OpenCL), which can execute

the same code on any parallel computational device in a modern computer. The open source software Blender was used to create and manage 3D models. It works as both 3D kernel while running a code, and a platform for post processing. This approach made it easier to deal with the 3D geometry in general, all the way from building it, working with it, to making the results visible in an understandable way after the simulation was complete.

## Chapter 10

# Further Work

As this project is supposed to be continued in a PhD project, this section will actually be of great importance for the author. Many things have not been explored, and the author sees possibilities for many interesting research opportunities on the topic of wing sail driven cargo ships. A better understanding of the wing sail interaction effects seems to be an important step. The interaction effects are seen to be strong in this project, but there is a viscous aspect to the interaction that has not been modeled. In addition, increasing the lift from the wing sails is believed to be a very good way to increase the thrust. High lift devices, such as flaps, leading edge slots, and non-planar wing concepts, such as winglets, all seem like interesting research topics. The general design of a wing sail is not very much explored.

It was a bit disappointing, from a research point of view, that there seemed to be such a small effect on the effective thrust when heel and yaw were included in the modeling. Researching elegant ways of reducing the yaw angles of cargo ship would be an interesting topic. However, the author does believe that effects of yaw will be more important if the wing sail technology is pushed to its limits. In particular, increasing the effect of the wing sails at small wind angles would push the ship harder to the side, and the added resistance due to yaw will quickly rise with increasing yaw angles, due to its non-linear nature.

Another way of making a wing sail more effective is to reduce the resistance of the ship in general. however, a slender, low resistance ship hull might have problems with stability. Finding a design with low resistance, enough stability and good lifting properties could be very interesting. The mix between high lift wing sails, and low resistance ship hulls is believed to not only be the key to making the wing sail technology viable, it also makes the physics of the problem much more exciting. It would require more interesting solutions for reducing the yaw and heel angle.

The modeling of a sailing ship is completely possible with today's fast computers, but it does take quite a lot of time. Finding a fast, but reliable, way of modeling the forces that acts on a ship that moves with arbitrary angles is an interesting challenge. The author has worked quite a lot with both CFD and BEM in this project, and it seems that one possible way of achieving this could be by coupling a lifting BEM code to a CFD code. There exists some software that already couples CFD with BEM, but the author has not seen anybody coupling a lifting BEM code to CFD for a marine purpose. That is, lifting BEM that models a full free surface, where the interaction between the potential wake and the free surface must be an important feature, and where viscous effects, such as cross flow drag, can be modeled properly by having a CFD code that only models viscosity where viscosity has an effect (i.e. the boundary layer).

Optimization of a wing sail driven cargo ship is also a big topic. Many design choices must be made in order to create the perfect wing sail driven cargo ship. Finding the perfect balance between stability and resistance, the optimal wing sail shape, and perhaps most importantly, finding the optimal routes for such a vessel are all interesting and important topics.

The author feels that there is a lot to do, and is very happy that this project is not yet over.

to be continued....

# Bibliography

- [1] Jørgen Amdahl, Anders Endal, Geir Fuglerud, Knut Minsaas, Magnus Rasmussen, Bjørn Sillerud, Bjørn Sortland, and Harald Valland. *TMR 4100 - Marin Teknikk Intro and TMR 4105 - Marin Teknikk 1*. Marin Teknisk Senter, NTNU, August 2005.
- [2] John D. Anderson. *Fundamentals of Aerodynamics*. McGraw-Hill, fourth edition, 2007.
- [3] Juan Baader. *The sailing yacht*. Adlard Coles Nautical, 1979.
- [4] Götz Bramesfeld. *A Higher Order Vortex-Lattice Method with a Force-Free Wake*. PhD thesis, The Pennsylvania State University, 2006.
- [5] GB Deng, A Leroyer, E Guilmineau, P Queutey, M Visonneau, and J Wackers. Verification and validation for unsteady computation. In *Proceedings of Gothenburg 2010: A Workshop on CFD in Ship Hydrodynamics, Gothenburg, Sweden, 2010*.
- [6] Andre Deperrois. XFLR5 analysis of foils and wings operating at low reynolds numbers, 2009. [https://engineering.purdue.edu/~aerodyn/AAE333/FALL10/HOMEWORKS/HW13/XFLR5\\_v6.01\\_Beta\\_Win32\(2\)/Release/Guidelines.pdf](https://engineering.purdue.edu/~aerodyn/AAE333/FALL10/HOMEWORKS/HW13/XFLR5_v6.01_Beta_Win32(2)/Release/Guidelines.pdf).
- [7] Mark Drela. Xfoil: An analysis and design system for low reynolds number airfoils. *Low Reynolds Number Aerodynamics*, 1989.
- [8] Mark Drela and Michael B Giles. Viscous-inviscid analysis of transonic and low reynolds number airfoils. *AIAA journal*, 25(10):1347--1355, 1987.
- [9] Gabriel Hugh Elkaim. Autonomous surface vehicle free-rotating wingsail section design and configuration analysis. *Journal Of Aircraft*, 45(6):1835--1852, 2008.
- [10] Sven Enger, Milovan Perić, and Robinson Perić. Simulation of flow around kcs-hull. In *A Workshop on Numerical Ship Hydrodynamics*, 2010.
- [11] Odd M. Faltinsen. *Sea Loads on Ships and Offshore Structures*. Cambridge Univeristy Press, 1990.
- [12] Odd M. Faltinsen. *Hydrodynamics of High-Speed Marine Vehicles*. Cambridge Univeristy Press, 2005.
- [13] Joel H Ferziger and Milovan Perić. *Computational methods for fluid dynamics*, volume 3. Springer Berlin, 2002.
- [14] Steffen Hasfjord. Optimization of propeller pitch and revolutions in behind condition. Master's thesis, Norwegian University of Science and Technology, 2014.
- [15] John Hess. Calculation of potential flow about arbitrary three-dimensional lifting bodies. Technical report, DTIC Document, 1972.
- [16] John Hess and AM Smith. Calculation of potential flow about arbitrary three-dimensional bodies. Tech-

- nical report, DTIC Document, 1962.
- [17] Jorge Izquierdo Yerón and Leo Miguel González Gutierrez. Urans computations of a dtmb 5415. 2010.
- [18] Joseph Katz and Allen Plotkin. *Low-Speed Aerodynamics*. Cambridge Univeristy Press, second edition, 2001.
- [19] Andreas Klöckner. Pyopencl. <http://mathematician.de/software/pyopencl/>.
- [20] Jarle Andre Kramer. Hydrofoil cargo ship - a feasibility study with main focus on hydrodynamics and resistance, December 2013.
- [21] I. Kroo. Nonplanar wing concepts for increased aircraft efficiency. [http://aero.stanford.edu/reports/VKI\\_nonplanar\\_Kroo.pdf](http://aero.stanford.edu/reports/VKI_nonplanar_Kroo.pdf), June 2005.
- [22] Lars Larsson, Rolf Eliasson, and Michal Orych. *Principles of yacht design*. A&C Black, 2014.
- [23] Lars Larsson and Lu Zou. Evaluation of resistance, sinkage and trim, self propulsion and wave pattern predictions. In *Numerical Ship Hydrodynamics*, pages 17--64. Springer, 2014.
- [24] J Longo and F Stern. Effects of drift angle on model ship flow. *Experiments in fluids*, 32(5):558--569, 2002.
- [25] Brian Maskew. Program VSAERO theory document. *NASA CR-4023*, 1987.
- [26] Jerome H Milgram. Fluid mechanics for sailing vessel design. *Annual review of fluid mechanics*, 30(1):613--653, 1998.
- [27] Takahiro Miyasaka, Takuji Nakashima, Yasunori Nihei, et al. Prediction and improvement of propulsive performance of wing sails considering their aerodynamic interaction. In *The Twenty-third International Offshore and Polar Engineering Conference*. International Society of Offshore and Polar Engineers, 2013.
- [28] NRK. Hurtigruta kom seg ikke under brua. <http://www.nrk.no/nordland/hurtigruta-kom-seg-ikke-under-brua-1.8028711>, September 2012.
- [29] Michal Orych, Lars Larsson, and Björn Regnström. Viscous free surface calculations for the kcs hull. In *Proceedings of Gothenburg 2010: A Workshop on CFD in Ship Hydrodynamics, Gothenburg, Sweden*, 2010.
- [30] Kazuyuki Ouchi, Kiyoshi Uzawa, Akihiro Kanai, and Masanobu Katori. "wind challenger" the next generation hybrid sailing vessel.
- [31] Ludwig Prandtl. *Applications of modern hydrodynamics to aeronautics*. National Advisory Committee for Aeronautics, 1923.
- [32] Ludwig Prandtl. *Applications of modern hydrodynamics to aeronautics*. National Advisory Committee for Aeronautics, 1923.
- [33] Rémi Retho and Sverre Steen. Comparative study of the hydrodynamic properties of two traditional norwegian sailing yachts. July 2014.
- [34] M. Nita Scholz. Estimating the oswald factor from basic aircraft geometrical parameters. *Deutscher Luft- und Raumfahrtkongress*, 2012.
- [35] David Scott. Sail-assist for cargo ships: computer controlled. *Popular Science*, 1985.
- [36] C Simonsen and F Stern. Cfd simulation of kcs sailing in regular head waves. In *Proceedings from*

*Gothenburg 2010---A Workshop on Numerical Ship Hydrodynamics*, volume 2, 2010.

- [37] Sverre Steen and Knut Minsaas. *TMR4220 Naval Hydrodynamics, Ship Resistance*. Department of Marine technology, NTNU, 2012.
- [38] Hendrik Tennekes and John Leask Lumley. *A first course in turbulence*. MIT press, 1972.
- [39] Blender foundation. Blender. <http://www.blender.org>.
- [40] IIHR-Hydroscience and Engineering at the University of Iowa. EFD data from the University of Iowa. <http://www.iihr.uiowa.edu/shiphydro/efd-data/>.
- [41] Norwegian Water Resources and Energy Directorate. Vindkraft. <http://www.nve.no/no/Energi1/Fornybar-energi/Vindkraft/>.
- [42] Airfoil Tools. Airfoil tools website. <http://airfoiltools.com>.
- [43] Robert Wall and Christopher Jasper. Clipper ships return: Green vessels for shipping are in the works. <http://www.businessweek.com/articles/2013-07-18/clipper-ships-return-green-vessels-for-shipping-are-in-the-works>, July 2013.
- [44] Wikipedia. Ac72. <http://en.wikipedia.org/wiki/AC72>.
- [45] Wikipedia. Liste over broer i norge etter lengde. [http://no.wikipedia.org/wiki/Liste\\_over\\_broer\\_i\\_Norge\\_etter\\_lengde](http://no.wikipedia.org/wiki/Liste_over_broer_i_Norge_etter_lengde).
- [46] Wikipedia. Opencl. <http://en.wikipedia.org/wiki/OpenCL>.
- [47] Wikipedia. Oswald efficiency number. [http://en.wikipedia.org/wiki/Oswald\\_efficiency\\_number](http://en.wikipedia.org/wiki/Oswald_efficiency_number).
- [48] Wikipedia. Wind profile power law. [http://en.wikipedia.org/wiki/Wind\\_profile\\_power\\_law](http://en.wikipedia.org/wiki/Wind_profile_power_law).
- [49] Q Wu, XM Feng, H Yu, JB Wang, RQ Cai, and XL Chen. Prediction of ship resistance and propulsion performance using multi-block structured grids. In *CFD Workshop Gothenburg, Gothenburg, Sweden*, 2010.



# Chapter 11

## Appendix

### 11.1 BEM code

---

#### 11.1.1 Example control script

```
import bpy
import sys
import imp
import time
import numpy as np
from os import system

np.set_printoptions(threshold=np.nan)

print('starting simulation')

import Geometry
imp.reload(Geometry)
import Computation
imp.reload(Computation)

# Generate mesh data
wing = Geometry.MeshData('wing')
wing.findStrips(2*30-1)
wake = Geometry.MeshData('wake')
wake.findStrips(2*30-1)

wingList = [wing]
wakeList = [wake]

# Set up OpenCL
opencl = Computation.Computation(workPath)
wing.setOpenCL(opencl.cntxt)
wake.setOpenCL(opencl.cntxt)

# External velocity vector
U = np.array([1.0, 0, 0])

# System to be solved
for i in range(3):
    print('Iteration:', i)
    B = opencl.influenceMatrixSourceDirichlet(wingList)
    A = opencl.influenceMatrixDoubletDirichlet(wingList)
    opencl.influenceFromStripsDirichlet(A, wingList, wakeList)

    sigma = opencl.rightSideSigma(wingList, U)
    b = -np.dot(B, sigma)

    gamma = np.linalg.solve(A, b)

# Set strengths
wing.setGamma(gamma, opencl.cntxt)
wing.setSigma(sigma, opencl.cntxt)
wake.setGammaWake(wing, opencl.cntxt)

opencl.deformWake(wake, wingList, wakeList, U)

wing.postProcessingDirichlet(U)
```

```

print('Pressure_Force:', wing.Force[0], wing.Force[1], wing.Force[2])

F = openc1.forceCalculation(wake, wing, wakeList, wingList, U)

print('Wing_Kutta-joukowski_Force:', F[0], F[1], F[2])

wake.updateData(openc1.cntxt)
wing.Force = np.zeros(3)

wing.velocityColorMap()
wing.pressureColorMap()

print('ending_simulation')
print('\n')

```

## 11.1.2 Geometry.py

```

import bpy
import numpy as np
import mathutils
import ColorMaps
import pyopenc1 as cl
import os

class MeshData:

    def __init__(self, objectName):
        ''' Class that contains the blender data, and data formatet to be sent to computations'''

        ''' Blender data '''
        self.object = bpy.data.objects[objectName]

        # Create rotation matrix for transforming normals to global coordinate system
        euler_x = self.object.rotation_euler[0]
        euler_y = self.object.rotation_euler[1]
        euler_z = self.object.rotation_euler[2]
        Rx = mathutils.Matrix.Rotation(euler_x, 4, 'X')
        Ry = mathutils.Matrix.Rotation(euler_y, 4, 'Y')
        Rz = mathutils.Matrix.Rotation(euler_z, 4, 'Z')

        self.matrix_rotation = Rz * Ry * Rx
        self.matrix_world = self.object.matrix_world

        self.polygons = self.object.data.polygons

        ''' OpenCL formatet data '''
        self.nrPoly = len(self.polygons)
        self.nrVert = len(self.object.data.vertices)

        self.vertices = np.zeros((self.nrVert, 4), dtype = np.float32) # Global coordinates
        self.ctrlPoints = np.zeros((self.nrPoly, 4), dtype = np.float32) # Global coordinates
        self.topology = np.zeros((self.nrPoly, 4), dtype = np.int32 ) # Topology in openc1 formate

        # Local coordinate system
        self.l = np.zeros((self.nrPoly, 4), dtype = np.float32)
        self.m = np.zeros((self.nrPoly, 4), dtype = np.float32)
        self.n = np.zeros((self.nrPoly, 4), dtype = np.float32)

        # Go through all vertices
        for i in range(self.nrVert):
            p = self.matrix_world * self.object.data.vertices[i].co # Transform to global coordinates

            self.vertices[i][0] = np.float32(p[0])
            self.vertices[i][1] = np.float32(p[1])
            self.vertices[i][2] = np.float32(p[2])

        # Go through all polygons
        for i in range(self.nrPoly):
            n = self.matrix_rotation * self.polygons[i].normal # Transform to global coordinates
            p = self.matrix_world * self.polygons[i].center

            i1 = self.polygons[i].vertices[0]
            i2 = self.polygons[i].vertices[1]
            i3 = self.polygons[i].vertices[2]
            i4 = self.polygons[i].vertices[3]

            p1 = self.vertices[i1]
            p2 = self.vertices[i2]
            p3 = self.vertices[i3]
            p4 = self.vertices[i4]

            l = (p1 + p2 - p3 - p4)
            l = l/np.sqrt( l[0]**2 + l[1]**2 + l[2]**2 )

            m = np.zeros(3)

```

```

m[0] = n[1]*l[2] - l[1]*n[2]
m[1] = l[0]*n[2] - n[0]*l[2]
m[2] = n[0]*l[1] - l[0]*n[1]

m = m/np.sqrt( m[0]**2 + m[1]**2 + m[2]**2 )

for j in range(3):
    self.ctrlPoints[i][j] = np.float32(p[j])

    self.l[i][j] = np.float32(l[j])
    self.m[i][j] = np.float32(m[j])
    self.n[i][j] = np.float32(n[j])

for j in range(4):
    self.topology[i][j] = np.int32(self.polygons[i].vertices[j])

''' Initialize variables for later use '''
self.velocity = np.zeros( (self.nrPoly, 4) )
self.pressure = np.zeros( self.nrPoly )
self.Force     = np.zeros(3)

self.vertVelocity = np.zeros( (self.nrVert, 4) )
self.vertPressure = np.zeros( self.nrVert )

self.nrStrips = 0
self.nrPolyPrStrip = 0

def findStrips(self, nrStrips, nrPolyPrStrip = 0):
    ''' Divide mesh into strips, based on assumption that it os from NURBS object in blender'''

    self.nrStrips = nrStrips

    if nrPolyPrStrip == 0:
        self.nrPolyPrStrip = np.int(self.nrPoly/nrStrips)
    else:
        self.nrPolyPrStrip = nrPolyPrStrip

    self.polyInStrips = np.zeros( (self.nrStrips, self.nrPolyPrStrip), dtype = np.int32 ) # Indices of
    polygons in strips
    self.stripForPoly = np.zeros(self.nrPoly) # Which strip each polygon belongs to

    iPoly = 0
    for i in range(nrStrips):
        for j in range(self.nrPolyPrStrip):
            self.polyInStrips[i][j] = np.int32(iPoly)

            self.stripForPoly[iPoly] = i

            iPoly += 1

# Creating colormap to seperate out the strips
colorMaps = self.object.data.vertex_colors

createColorMap = True
colorMapName = 'strips'

for i in range(len(colorMaps)):
    if colorMaps[i].name == colorMapName:
        createColorMap = False

if createColorMap:
    colorMaps.new(colorMapName)

iColor = 0
for iPoly in range(self.nrPoly):
    strip = self.stripForPoly[iPoly]

    rgb = ColorMaps.jet(strip, nrStrips-1, 0)

    for iTop in range(4):
        colorMaps[colorMapName].data[iColor].color = rgb

        iColor += 1

def setOpenCL(self, cntxt):
    self.topologyBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
    = self.topology)
    self.verticesBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
    = self.vertices)
    self.lBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
    = self.l)
    self.mBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
    = self.m)
    self.nBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
    = self.n)
    self.ctrlPointsBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf

```

```

        = self.ctrlPoints)

def updateData(self, cntxt):
    # Go through all vertices
    for i in range(self.nrVert):
        p = self.matrix_world * self.object.data.vertices[i].co # Transform to global coordinates

        self.vertices[i][0] = np.float32(p[0])
        self.vertices[i][1] = np.float32(p[1])
        self.vertices[i][2] = np.float32(p[2])

    # Go through all polygons
    for i in range(self.nrPoly):
        n = self.matrix_rotation * self.polygons[i].normal # Transform to global coordinates
        p = self.matrix_world * self.polygons[i].center

        i1 = self.polygons[i].vertices[0]
        i2 = self.polygons[i].vertices[1]
        i3 = self.polygons[i].vertices[2]
        i4 = self.polygons[i].vertices[3]

        p1 = self.vertices[i1]
        p2 = self.vertices[i2]
        p3 = self.vertices[i3]
        p4 = self.vertices[i4]

        l = (p1 + p2 - p3 - p4)
        l = l/np.sqrt( l[0]**2 + l[1]**2 + l[2]**2 )

        m = np.zeros(3)
        m[0] = n[1]*l[2] - l[1]*n[2]
        m[1] = l[0]*n[2] - n[0]*l[2]
        m[2] = n[0]*l[1] - l[0]*n[1]

        m = m/np.sqrt( m[0]**2 + m[1]**2 + m[2]**2 )

    for j in range(3):
        self.ctrlPoints[i][j] = np.float32(p[j])

        self.l[i][j] = np.float32(l[j])
        self.m[i][j] = np.float32(m[j])
        self.n[i][j] = np.float32(n[j])

    # Update buffers
    self.verticesBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
        = self.vertices)
    self.lBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
        = self.l)
    self.mBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
        = self.m)
    self.nBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
        = self.n)
    self.ctrlPointsBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf
        = self.ctrlPoints)

def setGamma(self, gamma, cntxt):
    self.gamma = np.float32(gamma)

    self.gammaBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf =
        self.gamma)

def setSigma(self, sigma, cntxt):
    self.sigma = np.float32(sigma)

    self.sigmaBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf =
        self.sigma)

def setGammaWake(self, wingData, cntxt):
    self.gamma = np.zeros(self.nrPoly, dtype = np.float32)

    for i in range(self.nrStrips):
        trailingIndex1 = wingData.polyInStrips[i][0]
        trailingIndex2 = wingData.polyInStrips[i][-1]

        g = wingData.gamma[trailingIndex1] - wingData.gamma[trailingIndex2]

        for j in range(self.nrPolyPrStrip):
            index = self.polyInStrips[i][j]
            self.gamma[index] = g

    self.gammaBuff = cl.Buffer(cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.COPY_HOST_PTR, hostbuf =
        self.gamma)

def postProcessingDirichlet(self, U):
    Uinf = U[0]**2 + U[1]**2 + U[2]**2
    # Go through all polygons once

```

```

for i in range(self.nrStrips):
    for j in range(self.nrPolyPrStrip):

        # Current polygon
        i0 = self.polyInStrips[i][j]
        p0 = self.ctrlPoints[i0]

        if (i==0):
            iN = self.polyInStrips[i+1][j]

            pN = self.ctrlPoints[iN]

            gamma_N2 = self.gamma[iN]
            gamma_N1 = self.gamma[i0]

            L_m = np.sqrt( (pN[0] - p0[0])**2 + (pN[1] - p0[1])**2 + (pN[2] - p0[2])**2 )
        elif (i == self.nrStrips - 1):
            iS = self.polyInStrips[i-1][j]

            pS = self.ctrlPoints[iS]

            gamma_N2 = self.gamma[i0]
            gamma_N1 = self.gamma[iS]

            L_m = np.sqrt( (p0[0] - pS[0])**2 + (p0[1] - pS[1])**2 + (p0[2] - pS[2])**2 )
        else:
            iN = self.polyInStrips[i+1][j]
            iS = self.polyInStrips[i-1][j]

            pN = self.ctrlPoints[iN]
            pS = self.ctrlPoints[iS]

            gamma_N2 = self.gamma[iN]
            gamma_N1 = self.gamma[iS]

            L_m = np.sqrt( (pN[0] - p0[0])**2 + (pN[1] - p0[1])**2 + (pN[2] - p0[2])**2 ) + np.sqrt(
                (p0[0] - pS[0])**2 + (p0[1] - pS[1])**2 + (p0[2] - pS[2])**2 )

        if (j==0):
            iE = self.polyInStrips[i][j+1]

            pE = self.ctrlPoints[iE]

            gamma_E2 = self.gamma[iE]
            gamma_0 = self.gamma[i0]

            SA = np.sqrt( (pE[0] - p0[0])**2 + (pE[1] - p0[1])**2 + (pE[2] - p0[2])**2 )
            SB = 0

            DA = (gamma_E2 - gamma_0)/SA
            DB = 0

            L_l = np.sqrt( (pE[0] - p0[0])**2 + (pE[1] - p0[1])**2 + (pE[2] - p0[2])**2 )
        elif (j==self.nrPolyPrStrip-1):
            iW = self.polyInStrips[i][j-1]

            pW = self.ctrlPoints[iW]

            gamma_0 = self.gamma[i0]
            gamma_E1 = self.gamma[iW]

            SA = 0
            SB = np.sqrt( (pW[0] - p0[0])**2 + (pW[1] - p0[1])**2 + (pW[2] - p0[2])**2 )
            DA = 0
            DB = (gamma_E1 - gamma_0)/SB
        else:
            iE = self.polyInStrips[i][j+1]
            iW = self.polyInStrips[i][j-1]

            pE = self.ctrlPoints[iE]
            pW = self.ctrlPoints[iW]

            gamma_E2 = self.gamma[iE]
            gamma_E1 = self.gamma[iW]
            gamma_0 = self.gamma[i0]

            SA = np.sqrt( (pE[0] - p0[0])**2 + (pE[1] - p0[1])**2 + (pE[2] - p0[2])**2 )
            SB = np.sqrt( (pW[0] - p0[0])**2 + (pW[1] - p0[1])**2 + (pW[2] - p0[2])**2 )

            DA = (gamma_E2 - gamma_0)/SA
            DB = (gamma_E1 - gamma_0)/SB

        u_l = -4*np.pi*( DA*SA - DB*SB )/(SA + SB)
        u_m = -4*np.pi*( gamma_N2 - gamma_N1 )/L_m
        u_n = 4*np.pi*self.sigma[i0]

```

```

# Transformation to global coordinate system
u = u_l*self.l[i0] + u_m*self.m[i0] + u_n*self.n[i0]

self.velocity[i0][0] = U[0] + u[0]
self.velocity[i0][1] = U[1] + u[1]
self.velocity[i0][2] = U[2] + u[2]

self.velocity[i0][3] = np.sqrt( self.velocity[i0][0]**2 + self.velocity[i0][1]**2 + self.
    velocity[i0][2]**2 )

self.pressure[i0] = 1 - (self.velocity[i0][3]**2)/Uinf

force = -self.pressure[i0]*self.n[i0]*self.object.data.polygons[i0].area

self.Force[0] += force[0]
self.Force[1] += force[1]
self.Force[2] += force[2]

def velocityMagnitude(self):
    for i in range(self.nrPoly):
        self.velocity[i][3] = np.sqrt( self.velocity[i][0]**2 + self.velocity[i][1]**2 + self.velocity[i]
            [2]**2 )

def vertVelocityMagnitude(self):
    for i in range(self.nrVert):
        self.vertVelocity[i][3] = np.sqrt( self.vertVelocity[i][0]**2 + self.vertVelocity[i][1]**2 +
            self.vertVelocity[i][2]**2 )

def velocityColorMap(self):
    colorMaps = self.object.data.vertex_colors

    createColorMap = True
    colorMapName = 'velocity'

    for i in range(len(colorMaps)):
        if colorMaps[i].name == colorMapName:
            createColorMap = False

    if createColorMap:
        colorMaps.new(colorMapName)

    uMax = self.velocity[0][3]
    uMin = self.velocity[0][3]

    for i in range(self.nrPoly):
        if uMax < self.velocity[i][3]:
            uMax = self.velocity[i][3]
        if uMin > self.velocity[i][3]:
            uMin = self.velocity[i][3]

    print('Min_velocity_magnitude:', uMin)
    print('Max_velocity_magnitude:', uMax)

    iColor = 0
    for iPoly in range(self.nrPoly):
        u = self.velocity[iPoly][3]

        rgb = ColorMaps.jet(u, uMax, uMin)

        for iTop in range(4):
            colorMaps[colorMapName].data[iColor].color = rgb

            iColor += 1

def vertVelocityColorMap(self):
    colorMaps = self.object.data.vertex_colors

    createColorMap = True
    colorMapName = 'vertVelocity'

    for i in range(len(colorMaps)):
        if colorMaps[i].name == colorMapName:
            createColorMap = False

    if createColorMap:
        colorMaps.new(colorMapName)

    uMax = self.vertVelocity[0][3]
    uMin = self.vertVelocity[0][3]

    for i in range(self.nrVert):
        if uMax < self.vertVelocity[i][3]:
            uMax = self.vertVelocity[i][3]
        if uMin > self.vertVelocity[i][3]:
            uMin = self.vertVelocity[i][3]

```

```

print('Min_vertVelocity_magnitude:', uMin)
print('Max_vertVelocity_magnitude:', uMax)

iColor = 0
for iPoly in range(self.nrPoly):

    for iTop in range(4):
        iVert = self.polygons[iPoly].vertices[iTop]

        u = self.vertVelocity[iVert][3]

        rgb = ColorMaps.jet(u, uMax, uMin)

        colorMaps[colorMapName].data[iColor].color = rgb

        iColor += 1

def pressureColorMap(self):
    colorMaps = self.object.data.vertex_colors

    createColorMap = True
    colorMapName = 'pressure'

    for i in range(len(colorMaps)):
        if colorMaps[i].name == colorMapName:
            createColorMap = False

    if createColorMap:
        colorMaps.new(colorMapName)

    pMax = self.pressure[0]
    pMin = self.pressure[0]

    for i in range(self.nrPoly):
        if pMax < self.pressure[i]:
            pMax = self.pressure[i]
        if pMin > self.pressure[i]:
            pMin = self.pressure[i]

    print('Min_pressure_coefficient:', pMin)
    print('Max_pressure_coefficient:', pMax)

    iColor = 0
    for iPoly in range(self.nrPoly):
        p = self.pressure[iPoly]

        rgb = ColorMaps.jet(p, pMax, pMin)

        for iTop in range(4):
            colorMaps[colorMapName].data[iColor].color = rgb

        iColor += 1

```

### 11.1.3 Computation.py

```

''' Class containing OpenCL variables, such as context and que, along with methods that perform operations
that are dependent on
OpenCL. Input to the methods are geometry data'''

import numpy as np
import pyopencl as cl
import os

class Computation:
    def __init__(self, workPath):
        platforms = cl.get_platforms()
        my_CPU_devices = platforms[0].get_devices(device_type = cl.device_type.CPU)
        my_GPU_devices = platforms[0].get_devices(device_type = cl.device_type.GPU)

        os.environ['PYOPENCL_COMPILER_OUTPUT'] = '1'

        self.cntxt = cl.Context(devices=my_CPU_devices)
        self.queue = cl.CommandQueue(self.cntxt)

        # Load kernel file
        kernelFile = open(workPath+'kernels.cl', 'r')
        kernelStr = "".join(kernelFile.readlines())
        kernelFile.close()

        # build kernel programs
        self.kernels = cl.Program(self.cntxt, kernelStr).build()

    def influenceMatrixSourceNewmann(self, meshData):

```

```

A = np.zeros( (meshData.nrPoly, meshData.nrPoly) )

a      = np.zeros(meshData.nrPoly, dtype = np.float32)
aBuff = cl.Buffer(self.cntxt, cl.mem_flags.WRITE_ONLY, a.nbytes)

for i in range(meshData.nrPoly):
    p = meshData.ctrlPoints[i]
    n = meshData.n[i]

    launch = self.kernels.sourceInfluenceNewmann(self.queue,
                                                (meshData.nrPoly,),
                                                (1,),
                                                p,
                                                n,
                                                meshData.verticesBuff,
                                                meshData.ctrlPointsBuff,
                                                meshData.lBuff,
                                                meshData.mBuff,
                                                meshData.nBuff,
                                                meshData.topologyBuff,
                                                aBuff)

    launch.wait()

    cl.enqueue_read_buffer(self.queue, aBuff, a).wait()

    A[i] = a

return A

def influenceMatrixSourceDirichlet(self, meshDataList):
    nrObjects = len(meshDataList)

    # Get overview of system size
    systemSize = 0
    for i in range(nrObjects):
        systemSize += meshDataList[i].nrPoly

    # Initialize influence matrix
    A = np.zeros( (systemSize, systemSize) )

    iRow = 0 # Row counter

    for objectNrRow in range(nrObjects):
        meshDataRow = meshDataList[objectNrRow] # The object where the control point is evaluated

        for i in range(meshDataRow.nrPoly):
            p = meshDataRow.ctrlPoints[i]
            aRow = np.array([])

            for objectNrCol in range(nrObjects):
                meshDataCol = meshDataList[objectNrCol]

                a      = np.zeros(meshDataCol.nrPoly, dtype = np.float32)
                aBuff = cl.Buffer(self.cntxt, cl.mem_flags.WRITE_ONLY, a.nbytes)

                launch = self.kernels.sourceInfluenceDirichlet(self.queue,
                                                            (meshDataCol.nrPoly,),
                                                            (1,),
                                                            p,
                                                            meshDataCol.verticesBuff,
                                                            meshDataCol.ctrlPointsBuff,
                                                            meshDataCol.lBuff,
                                                            meshDataCol.mBuff,
                                                            meshDataCol.nBuff,
                                                            meshDataCol.topologyBuff,
                                                            aBuff)

                launch.wait()

                cl.enqueue_read_buffer(self.queue, aBuff, a).wait()

                aRow = np.append(aRow, a) # Add result to end of list

            A[iRow] = aRow

            iRow += 1

    return A

def influenceMatrixDoubletNewmann(self, meshData):
    A = np.zeros( (meshData.nrPoly, meshData.nrPoly) )

    a      = np.zeros(meshData.nrPoly, dtype = np.float32)
    aBuff = cl.Buffer(self.cntxt, cl.mem_flags.WRITE_ONLY, a.nbytes)

    for i in range(meshData.nrPoly):
        p = meshData.ctrlPoints[i]
        n = meshData.n[i]

```



```

        launch = self.kernels.doubletInfluenceNewmann(self.queue,
                                                    (meshData.nrPoly,),
                                                    (1,),
                                                    p,
                                                    n,
                                                    meshData.verticesBuff,
                                                    meshData.ctrlPointsBuff,
                                                    meshData.lBuff,
                                                    meshData.mBuff,
                                                    meshData.nBuff,
                                                    meshData.topologyBuff,
                                                    aBuff)

        launch.wait()

        cl.enqueue_read_buffer(self.queue, aBuff, a).wait()

        A[i] = a

    return A

def influenceMatrixDoubletDirichlet(self, meshDataList):
    nrObjects = len(meshDataList)

    # Get overview of system size
    systemSize = 0
    for i in range(nrObjects):
        systemSize += meshDataList[i].nrPoly

    # Initialize influence matrix
    A = np.zeros( (systemSize, systemSize) )

    iRow = 0 # Row counter

    for objectNrRow in range(nrObjects):
        meshDataRow = meshDataList[objectNrRow] # The object where the control point is evaluated

        for i in range(meshDataRow.nrPoly):
            p = meshDataRow.ctrlPoints[i]
            aRow = np.array([])

            for objectNrCol in range(nrObjects):
                meshDataCol = meshDataList[objectNrCol]

                a = np.zeros(meshDataCol.nrPoly, dtype = np.float32)
                aBuff = cl.Buffer(self.ctx, cl.mem_flags.WRITE_ONLY, a.nbytes)

                launch = self.kernels.doubletInfluenceDirichlet(self.queue,
                                                            (meshDataCol.nrPoly,),
                                                            (1,),
                                                            p,
                                                            meshDataCol.verticesBuff,
                                                            meshDataCol.ctrlPointsBuff,
                                                            meshDataCol.lBuff,
                                                            meshDataCol.mBuff,
                                                            meshDataCol.nBuff,
                                                            meshDataCol.topologyBuff,
                                                            aBuff)

                launch.wait()

                cl.enqueue_read_buffer(self.queue, aBuff, a).wait()

                aRow = np.append(aRow, a) # Add result to end of list

            A[iRow] = aRow

            iRow += 1

    return A

def influenceFromStripsDirichlet(self, A, wingDataList, wakeDataList):
    nrObjects = len(wingDataList)

    iRow = 0

    for objectNrRow in range(nrObjects):
        wingDataRow = wingDataList[objectNrRow]
        wakeDataRow = wakeDataList[objectNrRow]

        for i in range(wingDataRow.nrPoly):
            p = wingDataRow.ctrlPoints[i]

            indexAdjust = 0

            for objectNrStrip in range(nrObjects):
                wingDataStrip = wingDataList[objectNrStrip]

```

```

wakeDataStrip = wakeDataList[objectNrStrip]

a = np.zeros( wakeDataStrip.nrPolyPrStrip, dtype = np.float32 )
aBuff = cl.Buffer(self.cntxt, cl.mem_flags.WRITE_ONLY, a.nbytes)

for j in range(wakeDataStrip.nrStrips):

    polyInStripBuff = cl.Buffer(self.cntxt, cl.mem_flags.READ_ONLY | cl.mem_flags.
        COPY_HOST_PTR, hostbuf = wakeDataStrip.polyInStrips[j])

    launch = self.kernels.stripInfluenceDirichlet(self.queue,
        (wakeDataStrip.nrPolyPrStrip,),
        (1,),
        p,
        wakeDataStrip.verticesBuff,
        wakeDataStrip.ctrlPointsBuff,
        wakeDataStrip.lBuff,
        wakeDataStrip.mBuff,
        wakeDataStrip.nBuff,
        wakeDataStrip.topologyBuff,
        polyInStripBuff,
        aBuff)

    launch.wait()

    cl.enqueue_read_buffer(self.queue, aBuff, a).wait()

    trailingIndex1 = indexAdjust + wingDataStrip.polyInStrips[j][0]
    trailingIndex2 = indexAdjust + wingDataStrip.polyInStrips[j][-1]

    asum = a.sum()

    A[iRow][trailingIndex1] += asum
    A[iRow][trailingIndex2] -= asum

    indexAdjust += wingDataStrip.nrPoly

    iRow += 1

def velocityAtPoint(self, p, wingDataList, wakeDataList, U):
    nrObjects = len(wingDataList)

    u = np.zeros(3) # Return value

    for i in range(nrObjects):
        wingData = wingDataList[i]
        wakeData = wakeDataList[i]

        aWing = np.zeros( (wingData.nrPoly, 4), dtype = np.float32 ) # Store values from wing
        aWingBuff = cl.Buffer(self.cntxt, cl.mem_flags.WRITE_ONLY, aWing.nbytes)

        aWake = np.zeros( (wakeData.nrPoly, 4), dtype = np.float32 ) # Store values from wake
        aWakeBuff = cl.Buffer(self.cntxt, cl.mem_flags.WRITE_ONLY, aWake.nbytes)

        # Find contribution from wing sources
        launch = self.kernels.velocitySource(self.queue,
            [wingData.nrPoly,],
            None,
            p,
            wingData.verticesBuff,
            wingData.ctrlPointsBuff,
            wingData.lBuff,
            wingData.mBuff,
            wingData.nBuff,
            wingData.topologyBuff,
            wingData.sigmaBuff,
            aWingBuff)

        launch.wait()

        cl.enqueue_read_buffer(self.queue, aWingBuff, aWing).wait()

        asum = aWing.sum(axis=0)

        u[0] += asum[0]
        u[1] += asum[1]
        u[2] += asum[2]

        # Find contribution from wing doublets
        launch = self.kernels.velocityDoublet(self.queue,
            [wingData.nrPoly,],
            None,
            p,
            wingData.verticesBuff,
            wingData.ctrlPointsBuff,
            wingData.lBuff,
            wingData.mBuff,
            wingData.nBuff,
            wingData.topologyBuff,

```

```

                                wingData.gammaBuff,
                                aWingBuff)

    launch.wait()

    cl.enqueue_read_buffer(self.queue, aWingBuff, aWing).wait()

    asum = aWing.sum(axis=0)

    u[0] += asum[0]
    u[1] += asum[1]
    u[2] += asum[2]

    # Find contribution from wake
    launch = self.kernels.velocityDoublet(self.queue,
                                           [wakeData.nrPoly,],
                                           None,
                                           p,
                                           wakeData.verticesBuff,
                                           wakeData.ctrlPointsBuff,
                                           wakeData.lBuff,
                                           wakeData.mBuff,
                                           wakeData.nBuff,
                                           wakeData.topologyBuff,
                                           wakeData.gammaBuff,
                                           aWakeBuff)

    launch.wait()

    cl.enqueue_read_buffer(self.queue, aWakeBuff, aWake).wait()

    asum = aWake.sum(axis=0)

    u[0] += asum[0]
    u[1] += asum[1]
    u[2] += asum[2]

    # Free stream velocity
    u[0] += U[0]
    u[1] += U[1]
    u[2] += U[2]

    return u

def inducedVelocityAtPoint(self, p, wakeDataList, wingDataList, wing):
    nrObjects = len(wakeDataList)

    u = np.zeros(3) # Return value

    for i in range(nrObjects):
        wakeData = wakeDataList[i]

        aWake = np.zeros( (wakeData.nrPoly, 4), dtype = np.float32 ) # Store values from wake
        aWakeBuff = cl.Buffer(self.cntxt, cl.mem_flags.WRITE_ONLY, aWake.nbytes)

        wingData = wingDataList[i]

        if wingData != wing:
            aWing = np.zeros( (wingData.nrPoly, 4), dtype = np.float32 ) # Store values from wing
            aWingBuff = cl.Buffer(self.cntxt, cl.mem_flags.WRITE_ONLY, aWing.nbytes)

            # Find contribution from wing sources
            launch = self.kernels.velocitySource(self.queue,
                                                [wingData.nrPoly,],
                                                None,
                                                p,
                                                wingData.verticesBuff,
                                                wingData.ctrlPointsBuff,
                                                wingData.lBuff,
                                                wingData.mBuff,
                                                wingData.nBuff,
                                                wingData.topologyBuff,
                                                wingData.sigmaBuff,
                                                aWingBuff)

            launch.wait()

            cl.enqueue_read_buffer(self.queue, aWingBuff, aWing).wait()

            asum = aWing.sum(axis=0)

            u[0] += asum[0]
            u[1] += asum[1]
            u[2] += asum[2]

            # Find contribution from wing doublets
            launch = self.kernels.velocityDoublet(self.queue,
                                                  [wingData.nrPoly,],

```

```

None,
p,
wingData.verticesBuff,
wingData.ctrlPointsBuff,
wingData.lBuff,
wingData.mBuff,
wingData.nBuff,
wingData.topologyBuff,
wingData.gammaBuff,
aWingBuff)

launch.wait()

cl.enqueue_read_buffer(self.queue, aWingBuff, aWing).wait()

asum = aWing.sum(axis=0)

u[0] += asum[0]
u[1] += asum[1]
u[2] += asum[2]

# Find contribution from wake
launch = self.kernels.velocityDoublet(self.queue,
[wakeData.nrPoly,],
None,
p,
wakeData.verticesBuff,
wakeData.ctrlPointsBuff,
wakeData.lBuff,
wakeData.mBuff,
wakeData.nBuff,
wakeData.topologyBuff,
wakeData.gammaBuff,
aWakeBuff)

launch.wait()

cl.enqueue_read_buffer(self.queue, aWakeBuff, aWake).wait()

asum = aWake.sum(axis=0)

u[0] += asum[0]
u[1] += asum[1]
u[2] += asum[2]

return u

def rightSideSigma(self, meshDataList, U):
nrObjects = len(meshDataList)

systemSize = 0

for objectNr in range(nrObjects):
systemSize += meshDataList[objectNr].nrPoly

sigma = np.zeros(systemSize)

iRow = 0

for objectNr in range(nrObjects):
meshData = meshDataList[objectNr]

for i in range(meshData.nrPoly):
n = meshData.n[i]

sigma[iRow] = -(n[0]*U[0] + n[1]*U[1] + n[2]*U[2])/(4*np.pi)

iRow += 1

return sigma

def deformWake(self, wake, wingDataList, wakeDataList, U):

Uinf = np.sqrt( U[0]**2 + U[1]**2 + U[2]**2 )

for i in range(wake.nrPolyPrStrip):
for j in range(wake.nrStrips+1):
vertIndex1 = (wake.nrPolyPrStrip+1)*(1 + j) - i - 1
vertIndex2 = (wake.nrPolyPrStrip+1)*(1 + j) - i - 2

p1 = wake.vertices[vertIndex1]
p2 = wake.vertices[vertIndex2]

p11 = 0.75*p1 + 0.25*p2
p22 = 0.25*p1 + 0.75*p2

dist = p2 - p1
dx = np.sqrt( dist[0]**2 + dist[1]**2 + dist[2]**2 )

```

```

    dt = dx/Uinf

    u1 = self.velocityAtPoint(p11, wingDataList, wakeDataList, U)
    u2 = self.velocityAtPoint(p22, wingDataList, wakeDataList, U)

    u = 0.5*(u1 + u2)

    dp = np.zeros(3)
    dp[0] = u[0]*dt
    dp[1] = u[1]*dt
    dp[2] = u[2]*dt

    wake.object.data.vertices[vertIndex2].co[0] = wake.object.data.vertices[vertIndex1].co[0] +
    dp[0]
    wake.object.data.vertices[vertIndex2].co[1] = wake.object.data.vertices[vertIndex1].co[1] +
    dp[1]
    wake.object.data.vertices[vertIndex2].co[2] = wake.object.data.vertices[vertIndex1].co[2] +
    dp[2]

    wake.object.data.update()
    wake.updateData(self.cntxt)

def forceCalculation(self, wake, wing, wakeDataList, wingDataList, U):
    Uinf = np.sqrt( U[0]**2 + U[1]**2 + U[2]**2 )

    Force = np.zeros(3)

    for i in range(wake.nrStrips):
        polyIndex = wake.polyInStrips[i][-1]

        vertIndex1 = (wake.nrPolyPrStrip+1)*(1 + i) - 1
        vertIndex2 = (wake.nrPolyPrStrip+1)*(2 + i) - 1

        p1 = wake.vertices[vertIndex1]
        p2 = wake.vertices[vertIndex2]

        dist = np.sqrt( (p2[0] - p1[0])**2 + (p2[1] - p1[1])**2 + (p2[2] - p1[2])**2)

        gammaVec = (p2 - p1)/dist

        p = 0.5*(p1 + p2)

        u = self.inducedVelocityAtPoint(p, wakeDataList, wingDataList, wing)

        gamma = wake.gamma[polyIndex]*gammaVec

        forceFactor = 4*np.pi*dist*2/Uinf**2

        Force[0] += forceFactor*((U[1] + u[1])*gamma[2] - (U[2] + u[2])*gamma[1])
        Force[1] += forceFactor*((U[2] + u[2])*gamma[0] - (U[0] + u[0])*gamma[2])
        Force[2] += forceFactor*((U[1] + u[1])*gamma[0] - (U[0] + u[0])*gamma[1])

    return Force

```

#### 11.1.4 Kernels.cl

```

// Prototypes
#define error 0.000000001

float4 multiplyMatrix(float4 p, float4 X, float4 Y, float4 Z);

float4 sourceLineVelocity(float4 p, float4 p1, float4 p2, float4 l, float4 m, float4 n);
float4 doubletLineVelocity(float4 p, float4 p1, float4 p2);

float sourceLinePotential(float4 p, float4 p1, float4 p2, float4 l, float4 m, float4 n);
float doubletLinePotential(float4 p, float4 p1, float4 p2, float4 l, float4 m, float4 n);

float calcC(float PN, float RNUM, float DNOM);

float4 unitSourceVelocity(int i,
    float4 p,
    __global float4 *vertices,
    __global float4 *ctrlPoints,
    __global float4 *l,
    __global float4 *m,
    __global float4 *n,
    __global int4 *topology);

float4 unitDoubletVelocity(int i,
    float4 p,
    __global float4 *vertices,
    __global float4 *ctrlPoints,
    __global float4 *l,
    __global float4 *m,

```

```

        __global float4 *n,
        __global int4 *topology);

float unitSourcePotential(int i,
        float4 p,
        __global float4 *vertices,
        __global float4 *ctrlPoints,
        __global float4 *l,
        __global float4 *m,
        __global float4 *n,
        __global int4 *topology);

float unitDoubletPotential(int i,
        float4 p,
        __global float4 *vertices,
        __global float4 *ctrlPoints,
        __global float4 *l,
        __global float4 *m,
        __global float4 *n,
        __global int4 *topology);

// Implementation

float4 multiplyMatrix(float4 p, float4 X, float4 Y, float4 Z) {
    float4 pOut;

    pOut.x = p.x*X.x + p.y*X.y + p.z*X.z;
    pOut.y = p.x*Y.x + p.y*Y.y + p.z*Y.z;
    pOut.z = p.x*Z.x + p.y*Z.y + p.z*Z.z;

    return pOut;
}

float calcC(float PN, float RNUM, float DNOM) {
    float C;

    if (PN == 0 && DNOM < 0) {
        C = -M_PI;
    }
    else if (PN == 0 && DNOM == 0) {
        C = -0.5*M_PI;
    }
    else if (PN == 0 && DNOM > 0) {
        C = 0.0;
    }
    else {
        C = atan2(RNUM, DNOM);
    }

    return C;
}

float4 sourceLineVelocity(float4 p, float4 p1, float4 p2, float4 l, float4 m, float4 n) {
    float4 u; // Return variable

    float4 s = p2 - p1;
    float4 a = p - p1;
    float4 b = p - p2;

    float S = sqrt(s.x*s.x + s.y*s.y + s.z*s.z);

    if (S > error) {
        float A = sqrt(a.x*a.x + a.y*a.y + a.z*a.z);
        float B = sqrt(b.x*b.x + b.y*b.y + b.z*b.z);

        float PN = p.x*n.x + p.y*n.y + p.z*n.z;

        float SL = s.x*l.x + s.y*l.y + s.z*l.z;
        float SM = s.x*m.x + s.y*m.y + s.z*m.z;

        float AL = a.x*l.x + a.y*l.y + a.z*l.z;
        float AM = a.x*m.x + a.y*m.y + a.z*m.z;

        float A1 = AM*SL - AL*SM;

        float PA = PN*PN*SL + A1*AM;
        float PB = PA - A1*SM;

        float RNUM = SM*PN*(B*PA - A*PB);
        float DNOM = PA*PB + PN*PN*A*B*SM*SM;

        float GL = (1/S)*log( fabs((A+B+S)/(A+B-S)) );

        float C = calcC(PN, RNUM, DNOM);

        u = GL*SM*l - GL*SL*m + C*n;
    }
}

```

```

else {
    u.x = 0;
    u.y = 0;
    u.z = 0;
}
return u;
}

float4 doubletLineVelocity(float4 p, float4 p1, float4 p2) {
    float4 u; // Return variable

    float4 a = p - p1;
    float4 b = p - p2;
    float4 s = p2 - p1;

    float AVBx = a.y*b.z - a.z*b.y;
    float AVBy = a.z*b.x - a.x*b.z;
    float AVBz = a.x*b.y - a.y*b.x;

    float AVB = AVBx*AVBx + AVBy*AVBy + AVBz*AVBz;

    float A = sqrt( a.x*a.x + a.y*a.y + a.z*a.z );
    float B = sqrt( b.x*b.x + b.y*b.y + b.z*b.z );
    float S = sqrt( s.x*s.x + s.y*s.y + s.z*s.z );

    if (A < error || B < error || AVB < error) {
        u.x = 0;
        u.y = 0;
        u.z = 0;
    }
    else if (S < error) {
        u.x = 0;
        u.y = 0;
        u.z = 0;
    }
    else {
        float ADB = a.x*b.x + a.y*b.y + a.z*b.z;

        float K = (A+B)/( A*B*(A*B + ADB) );

        u.x = K*AVBx;
        u.y = K*AVBy;
        u.z = K*AVBz;
    }

    return u;
}

float sourceLinePotential(float4 p, float4 p1, float4 p2, float4 l, float4 m, float4 n) {
    float4 s = p2 - p1;
    float4 a = p - p1;
    float4 b = p - p2;

    float S = sqrt(s.x*s.x + s.y*s.y + s.z*s.z);

    float phi;

    if (S > error) {
        float A = sqrt(a.x*a.x + a.y*a.y + a.z*a.z);
        float B = sqrt(b.x*b.x + b.y*b.y + b.z*b.z);

        float PN = p.x*n.x + p.y*n.y + p.z*n.z;

        float SL = s.x*l.x + s.y*l.y + s.z*l.z;
        float SM = s.x*m.x + s.y*m.y + s.z*m.z;

        float AL = a.x*l.x + a.y*l.y + a.z*l.z;
        float AM = a.x*m.x + a.y*m.y + a.z*m.z;

        float A1 = AM*SL - AL*SM;

        float PA = PN*PN*SL + A1*AM;
        float PB = PA - A1*SM;

        float RNUM = SM*PN*(B*PA - A*PB);
        float DNOM = PA*PB + PN*PN*A*B*SM*SM;

        float C = calcC(PN, RNUM, DNOM);

        float GL = (1/S)*log( fabs( (A+B+S)/(A+B-S) ) );

        phi = A1*GL - PN*C;
    }
    else {
        phi = 0;
    }
}

```

```

    return phi;
}

float doubletLinePotential(float4 p, float4 p1, float4 p2, float4 l, float4 m, float4 n) {
    float4 s = p2 - p1;
    float4 a = p - p1;
    float4 b = p - p2;

    float S = sqrt(s.x*s.x + s.y*s.y + s.z*s.z);

    float phi;

    if (S > error) {

        float A = sqrt(a.x*a.x + a.y*a.y + a.z*a.z);
        float B = sqrt(b.x*b.x + b.y*b.y + b.z*b.z);

        float PN = p.x*n.x + p.y*n.y + p.z*n.z;

        float SL = s.x*l.x + s.y*l.y + s.z*l.z;
        float SM = s.x*m.x + s.y*m.y + s.z*m.z;

        float AL = a.x*l.x + a.y*l.y + a.z*l.z;
        float AM = a.x*m.x + a.y*m.y + a.z*m.z;

        float A1 = AM*SL - AL*SM;

        float PA = PN*PN*SL + A1*AM;
        float PB = PA - A1*SM;

        float RNUM = SM*PN*(B*PA - A*PB);
        float DNOM = PA*PB + PN*PN*A*B*SM*SM;

        phi = calcC(PN, RNUM, DNOM);

    }
    else {
        phi = 0;
    }

    return phi;
}

float4 unitSourceVelocity(int i,
    float4 p,
    __global float4 *vertices,
    __global float4 *ctrlPoints,
    __global float4 *l,
    __global float4 *m,
    __global float4 *n,
    __global int4 *topology) {

    float4 u;

    float4 p0 = ctrlPoints[i]; //Center of panel

    p = p - p0; // Translate point into new panel centered coordinate system

    // Corner points
    int i1 = topology[i][0];
    int i2 = topology[i][1];
    int i3 = topology[i][2];
    int i4 = topology[i][3];

    float4 p1 = vertices[i1] - p0;
    float4 p2 = vertices[i2] - p0;
    float4 p3 = vertices[i3] - p0;
    float4 p4 = vertices[i4] - p0;

    // Find transformation matrices between global and local panel coordinate system
    float4 X = l[i];
    float4 Y = m[i];
    float4 Z = n[i];

    float PL = p.x*X.x + p.y*X.y + p.z*X.z;
    float PM = p.x*Y.x + p.y*Y.y + p.z*Y.z;
    float PN = p.x*Z.x + p.y*Z.y + p.z*Z.z;

    float4 u1 = sourceLineVelocity(p, p1, p2, l[i], m[i], n[i]);
    float4 u2 = sourceLineVelocity(p, p2, p3, l[i], m[i], n[i]);
    float4 u3 = sourceLineVelocity(p, p3, p4, l[i], m[i], n[i]);
    float4 u4 = sourceLineVelocity(p, p4, p1, l[i], m[i], n[i]);

    u = u1 + u2 + u3 + u4;

    if (fabs(PN) < error) {
        if ( fabs(PL) < error && fabs(PM) < error ) {

```



```

        u.z = 2*M_PI;
        u.x = 0;
        u.y = 0;
    }
    else {
        u.z = 0;
    }
}

return u;
}

float4 unitDoubletVelocity(int i,
                          float4 p,
                          __global float4 *vertices,
                          __global float4 *ctrlPoints,
                          __global float4 *l,
                          __global float4 *m,
                          __global float4 *n,
                          __global int4 *topology) {

    float4 p0 = ctrlPoints[i]; //Center of panel

    p = p - p0; // Translate point into new panel centered coordinate system

    // Corner points
    int i1 = topology[i][0];
    int i2 = topology[i][1];
    int i3 = topology[i][2];
    int i4 = topology[i][3];

    float4 p1 = vertices[i1] - p0;
    float4 p2 = vertices[i2] - p0;
    float4 p3 = vertices[i3] - p0;
    float4 p4 = vertices[i4] - p0;

    // Calculate contribution from every line
    float4 u1 = doubletLineVelocity(p, p1, p2);
    float4 u2 = doubletLineVelocity(p, p2, p3);
    float4 u3 = doubletLineVelocity(p, p3, p4);
    float4 u4 = doubletLineVelocity(p, p4, p1);

    float4 u = u1 + u2 + u3 + u4;

    return u;
}

float unitSourcePotential(int i,
                          float4 p,
                          __global float4 *vertices,
                          __global float4 *ctrlPoints,
                          __global float4 *l,
                          __global float4 *m,
                          __global float4 *n,
                          __global int4 *topology) {

    float4 p0 = ctrlPoints[i]; //Center of panel

    p = p - p0; // Translate point into new panel centered coordinate system

    // Corner points
    int i1 = topology[i][0];
    int i2 = topology[i][1];
    int i3 = topology[i][2];
    int i4 = topology[i][3];

    float4 p1 = vertices[i1] - p0;
    float4 p2 = vertices[i2] - p0;
    float4 p3 = vertices[i3] - p0;
    float4 p4 = vertices[i4] - p0;

    float phi1 = sourceLinePotential(p, p1, p2, l[i], m[i], n[i]);
    float phi2 = sourceLinePotential(p, p2, p3, l[i], m[i], n[i]);
    float phi3 = sourceLinePotential(p, p3, p4, l[i], m[i], n[i]);
    float phi4 = sourceLinePotential(p, p4, p1, l[i], m[i], n[i]);

    return phi1 + phi2 + phi3 + phi4;
}

float unitDoubletPotential(int i,
                           float4 p,
                           __global float4 *vertices,
                           __global float4 *ctrlPoints,
                           __global float4 *l,
                           __global float4 *m,
                           __global float4 *n,
                           __global int4 *topology) {

```

```

float4 p0 = ctrlPoints[i]; //Center of panel

float phi;

p = p - p0; // Translate point into new panel centered coordinate system

// Corner points
int i1 = topology[i][0];
int i2 = topology[i][1];
int i3 = topology[i][2];
int i4 = topology[i][3];

float4 p1 = vertices[i1] - p0;
float4 p2 = vertices[i2] - p0;
float4 p3 = vertices[i3] - p0;
float4 p4 = vertices[i4] - p0;

// Find transformation matrices between global and local panel coordinate system
float4 X = l[i];
float4 Y = m[i];
float4 Z = n[i];

float phi1 = doubletLinePotential(p, p1, p2, X, Y, Z);
float phi2 = doubletLinePotential(p, p2, p3, X, Y, Z);
float phi3 = doubletLinePotential(p, p3, p4, X, Y, Z);
float phi4 = doubletLinePotential(p, p4, p1, X, Y, Z);

phi = phi1 + phi2 + phi3 + phi4;

return phi;
}

__kernel void sourceInfluenceNewmann(float4 p,
                                     float4 n_p,
                                     __global float4 *vertices,
                                     __global float4 *ctrlPoints,
                                     __global float4 *l,
                                     __global float4 *m,
                                     __global float4 *n,
                                     __global int4 *topology,
                                     __global float *a) {

int i = get_global_id(0);

float4 u = unitSourceVelocity(i, p, vertices, ctrlPoints, l, m, n, topology);

float4 pm = p;
pm.z = -p.z;

float4 um = unitSourceVelocity(i, pm, vertices, ctrlPoints, l, m, n, topology);

a[i] = n_p.x*(u.x + um.x) + n_p.y*(u.y + um.y) + n_p.z*(u.z - um.z);
}

__kernel void doubletInfluenceNewmann(float4 p,
                                       float4 n_p,
                                       __global float4 *vertices,
                                       __global float4 *ctrlPoints,
                                       __global float4 *l,
                                       __global float4 *m,
                                       __global float4 *n,
                                       __global int4 *topology,
                                       __global float *a) {

int i = get_global_id(0);

float4 u = unitDoubletVelocity(i, p, vertices, ctrlPoints, l, m, n, topology);

float4 pm = p;
pm.z = -p.z;

float4 um = unitDoubletVelocity(i, pm, vertices, ctrlPoints, l, m, n, topology);

a[i] = n_p.x*(u.x + um.x) + n_p.y*(u.y + um.y) + n_p.z*(u.z - um.z);
}

__kernel void sourceInfluenceDirichlet(float4 p,
                                       __global float4 *vertices,
                                       __global float4 *ctrlPoints,
                                       __global float4 *l,
                                       __global float4 *m,
                                       __global float4 *n,
                                       __global int4 *topology,
                                       __global float *a) {

int i = get_global_id(0);

```

```

float phi = unitSourcePotential(i, p, vertices, ctrlPoints, l, m, n, topology);

float4 pm = p;
pm.z = -p.z;

float phim = unitSourcePotential(i, pm, vertices, ctrlPoints, l, m, n, topology);

a[i] = phi + phim;
}

__kernel void doubletInfluenceDirichlet(float4 p,
    __global float4 *vertices,
    __global float4 *ctrlPoints,
    __global float4 *l,
    __global float4 *m,
    __global float4 *n,
    __global int4 *topology,
    __global float *a){

int i = get_global_id(0);

float phi = unitDoubletPotential(i, p, vertices, ctrlPoints, l, m, n, topology);

float4 pm = p;
pm.z = -p.z;

float phim = unitDoubletPotential(i, pm, vertices, ctrlPoints, l, m, n, topology);

a[i] = phi + phim;
}

__kernel void stripInfluenceDirichlet(float4 p,
    __global float4 *vertices,
    __global float4 *ctrlPoints,
    __global float4 *l,
    __global float4 *m,
    __global float4 *n,
    __global int4 *topology,
    __global int *polyInStrip,
    __global float *a) {

int i = get_global_id(0);

int iPoly = polyInStrip[i]; // index of current wake polygon

float phi = unitDoubletPotential(iPoly, p, vertices, ctrlPoints, l, m, n, topology);

float4 pm = p;
pm.z = -p.z;

float phim = unitDoubletPotential(iPoly, pm, vertices, ctrlPoints, l, m, n, topology);

a[i] = phi + phim;
}

__kernel void stripInfluenceNewmann(float4 p,
    float4 n_p,
    __global float4 *vertices,
    __global float4 *ctrlPoints,
    __global float4 *l,
    __global float4 *m,
    __global float4 *n,
    __global int4 *topology,
    __global int *polyInStrip,
    __global float *a) {

int i = get_global_id(0);

int iPoly = polyInStrip[i]; // index of current wake polygon

float4 u = unitDoubletVelocity(iPoly, p, vertices, ctrlPoints, l, m, n, topology);

float4 pm = p;
pm.z = -p.z;

float4 um = unitDoubletVelocity(iPoly, pm, vertices, ctrlPoints, l, m, n, topology);

a[i] = n_p.x*(u.x + um.x) + n_p.y*(u.y + um.y) + n_p.z*(u.z - um.z);
}

__kernel void velocitySource(float4 p,
    __global float4 *vertices,
    __global float4 *ctrlPoints,
    __global float4 *l,
    __global float4 *m,
    __global float4 *n,
    __global int4 *topology,
    __global float *sigma,

```

```

    __global float4 *u) {
int i = get_global_id(0);

float4 uOut = unitSourceVelocity(i, p, vertices, ctrlPoints, l, m, n, topology);

float4 pm = p;
pm.z = -p.z;

float4 uOut_m = unitSourceVelocity(i, pm, vertices, ctrlPoints, l, m, n, topology);

u[i].x = sigma[i]*(uOut.x + uOut_m.x);
u[i].y = sigma[i]*(uOut.y + uOut_m.y);
u[i].z = sigma[i]*(uOut.z - uOut_m.z);
}

__kernel void velocityDoublet(float4 p,
    __global float4 *vertices,
    __global float4 *ctrlPoints,
    __global float4 *l,
    __global float4 *m,
    __global float4 *n,
    __global int4 *topology,
    __global float *gamma,
    __global float4 *u) {
int i = get_global_id(0);

float4 uOut = unitDoubletVelocity(i, p, vertices, ctrlPoints, l, m, n, topology);

float4 pm = p;
pm.z = -p.z;

float4 uOut_m = unitDoubletVelocity(i, pm, vertices, ctrlPoints, l, m, n, topology);

u[i].x = gamma[i]*(uOut.x + uOut_m.x);
u[i].y = gamma[i]*(uOut.y + uOut_m.y);
u[i].z = gamma[i]*(uOut.z - uOut_m.z);
}

```

### 11.1.5 Example Control Script

#### 11.1.6 Convergence Analysis

Number of Iterations in Order to Find a Wake Shape

Vsaero: 2

Physical aspect ratio: 2

Number of strips: 39

Number of panels pr strip: 89

Angle of attack 5

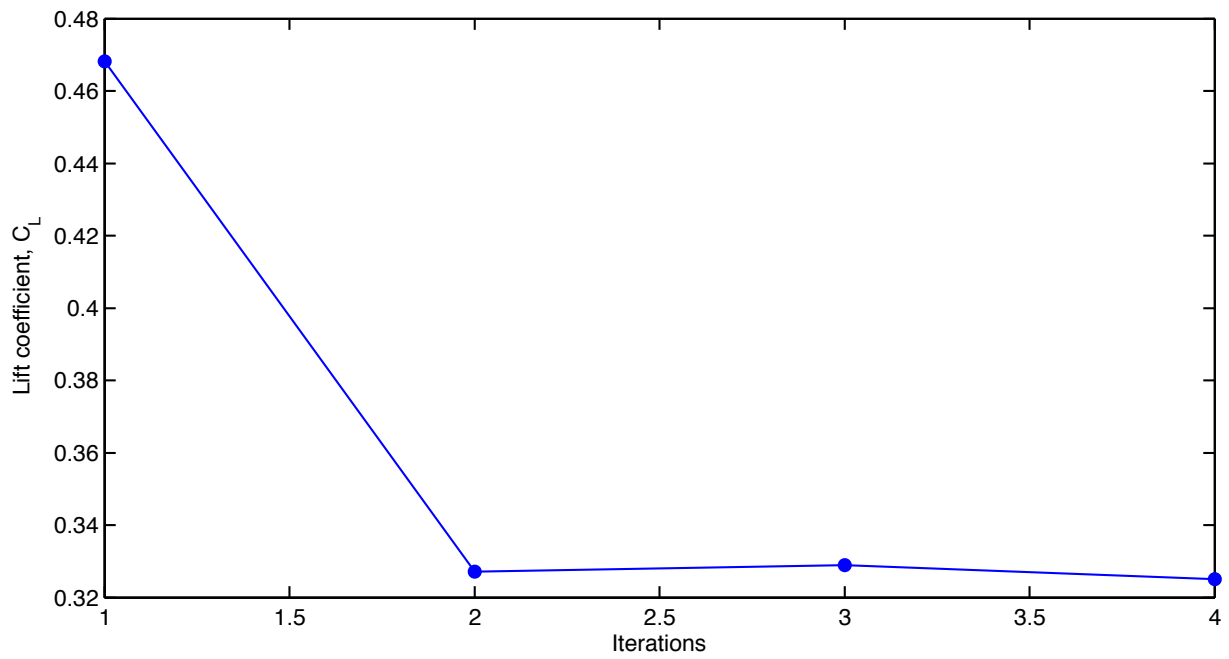


Figure 11.1: Iteration test to see the influence of wake shape on lift coefficient

Number of Panels pr Strip

Number of strips: 39

Physical aspect ratio: 5

Angle of attack: 5

Number of wake shape iterations: 2

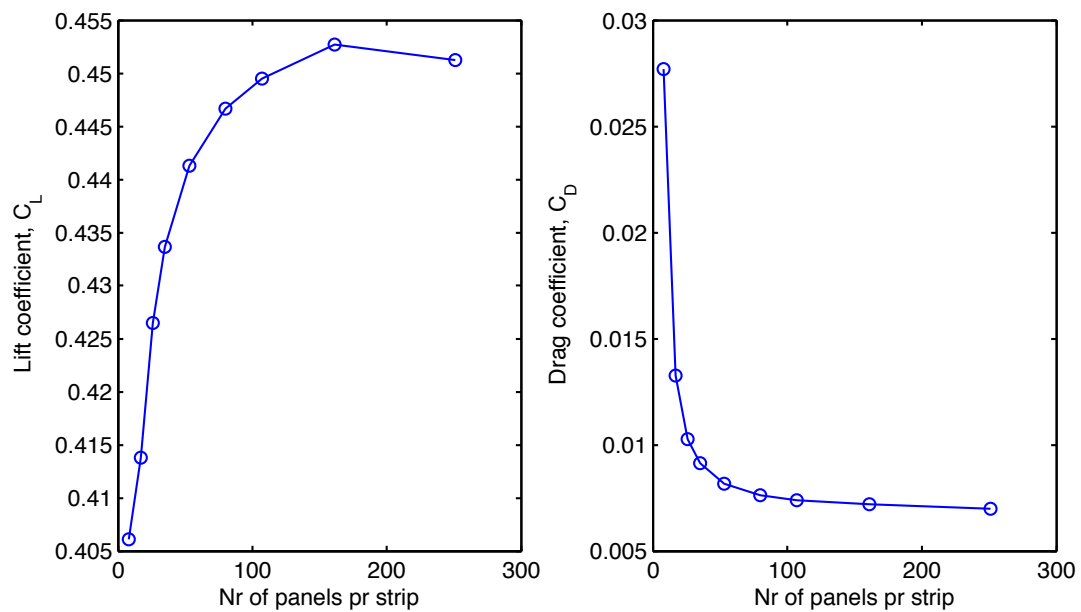


Figure 11.2: Lift and drag coefficient as a function of number of panels pr strip

Number of Strips

Number of panels pr strip: 161

Physical aspect ratio: 5

Angle of attack: 5

Number of wake shape iterations: 2

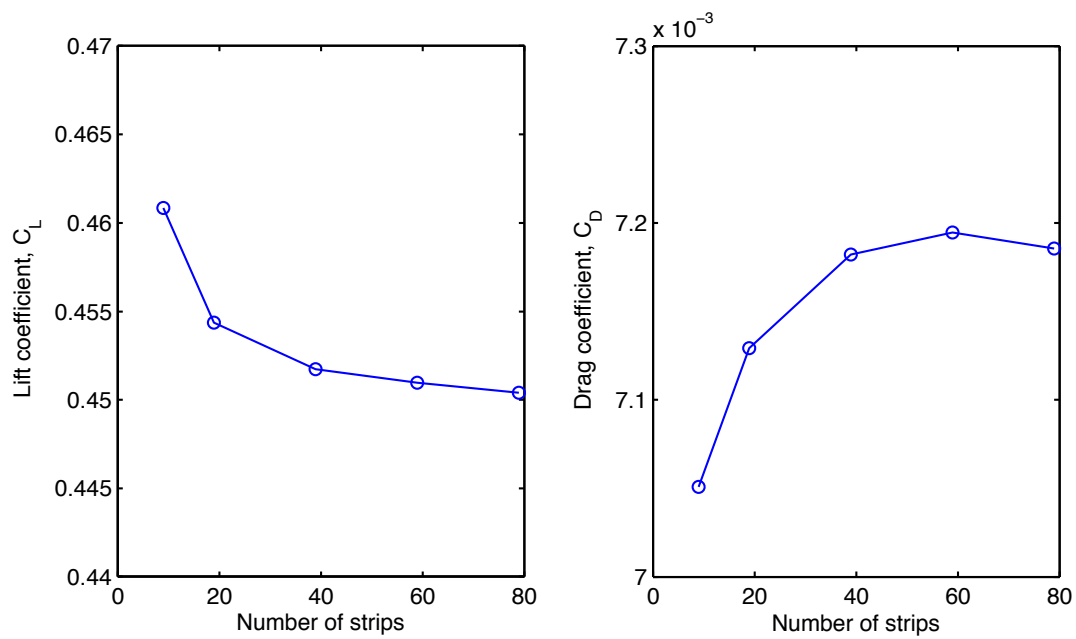


Figure 11.3: Lift and drag coefficient as a function of number of strips

Wake Length

number of strips: 39

number of panels pr strip 161

physical aspect ratio: 5

angle of attack: 5

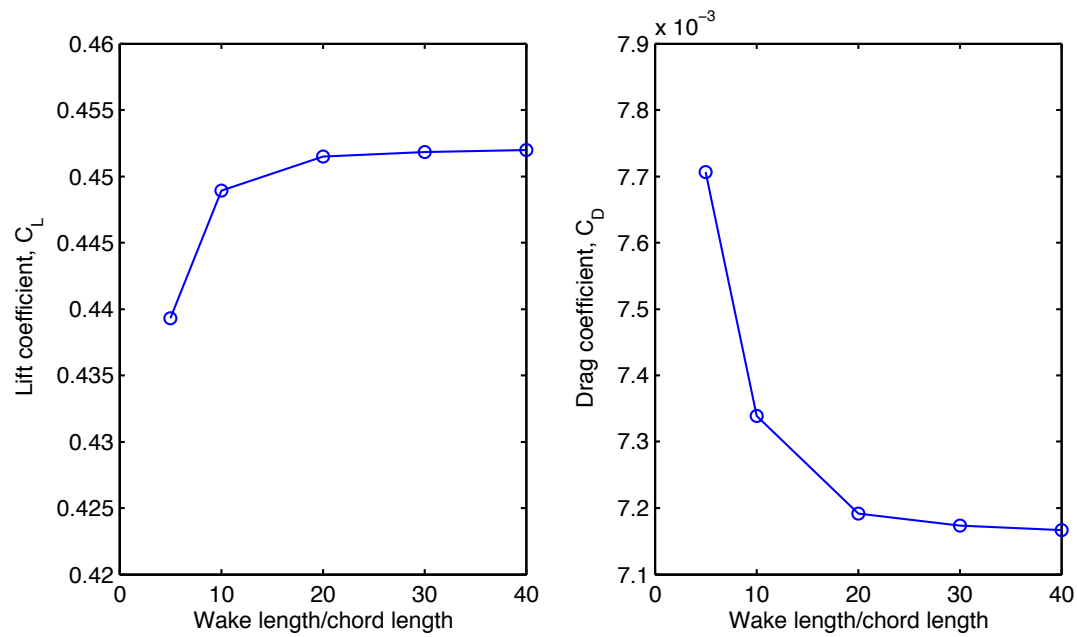


Figure 11.4: Lift and drag coefficient as a function of wake length

## 11.2 Final Results - Matlab scripts

### 11.2.1 wingCoefficientsChemicalTanker.m

```

close all
clear all

load('NACA0014_Data.mat')

% Script that calculates thrust and side force coefficients for different
% wind directions and angles of attack

% Wind directions are Apparent wind directions

%% Initialize variables
windAngles = 1:1:150;
nrWind = length(windAngles);

wingAngles = 0:0.1:24;
nrWing = length(wingAngles);

Cx = zeros(nrWind, nrWing);
Cy = zeros(nrWind, nrWing);

Cx_elliptic = zeros(nrWind, nrWing);
Cy_elliptic = zeros(nrWind, nrWing);

Cx_rectangular = zeros(nrWind, nrWing);
Cy_rectangular = zeros(nrWind, nrWing);

%% Wing factors
liftRec = 0.65;
interAng = [30, 60, 90, 120, 150];
liftInter = [0.6243, 0.6850, 0.6647, 0.6210, 0.5499];
dragInter = [1.5704, 1.2985, 1.0230, 0.9123, 0.8958];
Asp = 2.67*2;
delta = 0.054;

parfor i = 1:nrWind
    liftInterCurrent = interp1(interAng, liftInter, windAngles(i), 'spline');
    dragInterCurrent = interp1(interAng, dragInter, windAngles(i), 'spline');

    s = sind(windAngles(i));
    c = cosd(windAngles(i));

```

```

for j = 1:nrWing
    CL2D_current = interp1(angleFoil, CL2D, wingAngles(j), 'spline');
    CDv_Current = interp1(angleFoil, CDv, wingAngles(j), 'spline');

    CL_elliptic = CL2D_current/(1+2/Asp);
    CL_rectangular = CL2D_current*liftRec;
    CL = CL2D_current*liftRec*liftInterCurrent;

    CDi_elliptic = CL_elliptic^2/(pi*Asp);
    CDi_rectangular = (1+delta)*CL_rectangular^2/(pi*Asp);
    CDi = dragInterCurrent*(1+delta)*CL^2/(pi*Asp);

    CD_elliptic = CDi_elliptic + CDv_Current;
    CD_rectangular = CDi_rectangular + CDv_Current;
    CD = CDi + CDv_Current;

    Cx(i, j) = CL*s - CD*c;
    Cy(i, j) = -CL*c - CD*s;

    Cx_elliptic(i, j) = CL_elliptic*s - CD_elliptic*c;
    Cy_elliptic(i, j) = -CL_elliptic*c - CD_elliptic*s;

    Cx_rectangular(i, j) = CL_rectangular*s - CD_rectangular*c;
    Cy_rectangular(i, j) = -CL_rectangular*c - CD_rectangular*s;
end
end

[CxMax, index] = max(Cx, [], 2);
[CxMax_rectangular, index_rectangular] = max(Cx_rectangular, [], 2);
[CxMax_elliptic, index_elliptic] = max(Cx_elliptic, [], 2);

CyMax = zeros(1, nrWind);
CyMax_rectangular = zeros(1, nrWind);
CyMax_elliptic = zeros(1, nrWind);

testAngle = zeros(1, nrWind);

parfor i = 1:nrWind
    CyMax(i) = Cy(i, index(i));
    CyMax_rectangular(i) = Cy_rectangular(i, index_rectangular(i));
    CyMax_elliptic(i) = Cy_elliptic(i, index_elliptic(i));

    testAngle(i) = wingAngles(index(i))
end
figure(1)
plot(windAngles, CxMax, 'color', 'blue')
hold on
plot(windAngles, CxMax_rectangular, 'color', [0, 0.6, 0])
plot(windAngles, CxMax_elliptic, 'color', 'red')

xlabel('Apparent wind direction [deg]')
ylabel('thrust coefficient, C_x')
legend('Rectangular wings with interaction', 'Rectangular wing without interaction', 'Elliptic wing without interaction', 4)
axis([0, 180, 0, 1.5])

figure(2)
plot(windAngles, CyMax, 'color', 'blue')
hold on
plot(windAngles, CyMax_rectangular, 'color', [0, 0.6, 0])
plot(windAngles, CyMax_elliptic, 'color', 'red')

axis([0, 180, -1.4, 1.4])

xlabel('Apparent wind direction [deg]')
ylabel('Side force coefficient, C_y')
legend('Rectangular wings with interaction', 'Rectangular wing without interaction', 'Elliptic wing without interaction', 4)

```

## 11.2.2 wingCoefficientsSeries60.m

```

close all
clear all

load('NACA0014_Data.mat')
% Script that calculates thrust and side force coefficients for different
% wind directions and angles of attack

% Wind directions are Apparent wind directions

figure(1)
hold on

figure(2)

```



```

hold on

%% Series 60
windAngles = 5:1:150;
nrWind = length(windAngles);

wingAngles = 0:0.1:24;
nrWing = length(wingAngles);

Cx = zeros(nrWind, nrWing);
Cy = zeros(nrWind, nrWing);

Cx_elliptic = zeros(nrWind, nrWing);
Cy_elliptic = zeros(nrWind, nrWing);

Cx_rectangular = zeros(nrWind, nrWing);
Cy_rectangular = zeros(nrWind, nrWing);

% Wing factors
liftRec = 0.776;
interAng = [30, 60, 90, 120, 150];
liftInter = [0.6862, 0.7429, 0.7168, 0.6818, 0.6232];
dragInter = [2.0098, 1.7330, 1.4302, 1.3254, 1.3814];
Asp = 5*2;
delta = 0.076;

parfor i = 1:nrWind
    liftInterCurrent = interp1(interAng, liftInter, windAngles(i), 'spline');
    dragInterCurrent = interp1(interAng, dragInter, windAngles(i), 'spline');

    s = sind(windAngles(i));
    c = cosd(windAngles(i));

    for j = 1:nrWing
        CL2D_current = interp1(angleFoil, CL2D, wingAngles(j), 'spline');
        CDv_Current = interp1(angleFoil, CDv, wingAngles(j), 'spline');

        CL_elliptic = CL2D_current/(1+2/Asp);
        CL_rectangular = CL2D_current*liftRec;
        CL = CL2D_current*liftRec*liftInterCurrent;

        CDi_elliptic = CL_elliptic^2/(pi*Asp);
        CDi_rectangular = (1+delta)*CL_rectangular^2/(pi*Asp);
        CDi = dragInterCurrent*(1+delta)*CL^2/(pi*Asp);

        CD_elliptic = CDi_elliptic + CDv_Current;
        CD_rectangular = CDi_rectangular + CDv_Current;
        CD = CDi + CDv_Current;

        Cx(i, j) = CL*s - CD*c;
        Cy(i, j) = -CL*c - CD*s;

        Cx_elliptic(i, j) = CL_elliptic*s - CD_elliptic*c;
        Cy_elliptic(i, j) = -CL_elliptic*c - CD_elliptic*s;

        Cx_rectangular(i, j) = CL_rectangular*s - CD_rectangular*c;
        Cy_rectangular(i, j) = -CL_rectangular*c - CD_rectangular*s;
    end
end

[CxMax, index] = max(Cx, [], 2);
[CxMax_rectangular, index_rectangular] = max(Cx_rectangular, [], 2);
[CxMax_elliptic, index_elliptic] = max(Cx_elliptic, [], 2);

CyMax = zeros(1, nrWind);
CyMax_rectangular = zeros(1, nrWind);
CyMax_elliptic = zeros(1, nrWind);

testAngle = zeros(1, nrWind);

parfor i = 1:nrWind
    CyMax(i) = Cy(i, index(i));
    CyMax_rectangular(i) = Cy_rectangular(i, index_rectangular(i));
    CyMax_elliptic(i) = Cy_elliptic(i, index_elliptic(i));

    testAngle(i) = wingAngles(index(i))
end

figure(1)
plot(windAngles, CxMax, 'color', 'blue')
plot(windAngles, CxMax_rectangular, 'color', [0, 0.6, 0])
plot(windAngles, CxMax_elliptic, 'color', 'red')

xlabel('Apparent wind direction [deg]')
ylabel('thrust coefficient, C_x')
legend('8 rectangular wings', 'Single rectangular wing', 'Single elliptic wing', 4)
axis([0, 180, 0, 1.5])

```

```

title('Series_60_Wings')

figure(2)
plot(windAngles, CyMax, 'color', 'blue')
plot(windAngles, CyMax_rectangular, 'color', [0, 0.6, 0])
plot(windAngles, CyMax_elliptic, 'color', 'red')

axis([0, 180, -1.5, 1.5])

xlabel('Apparent_wind_direction_[deg]')
ylabel('Side_force_coefficient,_C_y')

legend('8_rectangular_wings', 'Single_rectangular_wing', 'Single_elliptic_wing', 2)
title('Series_60_Wings')

```

### 11.2.3 finalResultsChemicalTanker.m

```

clear all
close all

load('wingDataChemicalTanker.mat')
load('ShipForcesChemicalTanker.mat')

%% Plot Settings
color = zeros(3);
color(1, :) = [0, 0, 1.0];
color(2, :) = [0, 0.6, 0];
color(3, :) = [1.0, 0, 0];

figure(1);
hold on
figure(2);
hold on
figure(3);
hold on

%% Environment data
rho_m = 997.561; %kg/m^3
rho_s = 1025.9; %kg/m^3
nu_m = 1.19E-06; %m^2/s
nu_s = 8.91E-07; %m^2/s
rho_air = 1.225; %kg/m^3

Uwind = 7; %m/s

%% Sail data
nrSails = 8;
heightSails = 40; %m
chord = 15; %m
areaSail = heightSails*chord; %m^2
totalAreaSail = nrSails*areaSail; %m^2

sailFactor = 0.5*rho_air*totalAreaSail;

%% Ship data
shipAngles = 0:2:8;

%% Stability data
D = 10;
B = 29.5;
nabla = 38147;

KB = 5.215;
Ixx = 266172;
BM = Ixx/nabla;

stabSin = sind(5);

%% Different wind directions
windAngles2 = 15:1:150;
nrWind2 = length(windAngles2);
for k = 1:3
    % Initialize variables dependent on wind direction
    KG = zeros(1, nrWind2);

    Thrust = zeros(1, nrWind2);
    SideForce_wing = zeros(1, nrWind2);
    yawAngle = zeros(1, nrWind2);
    ResistanceWind = zeros(1, nrWind2);
    AddedResistance = zeros(1, nrWind2);
    RealThrust = zeros(1, nrWind2);

    % Go through each wind direction
    for i = 1:nrWind2

```

```

% Calculate apparent wind
u_apparent = Uwind*cosd(windAngles2(i)) + U_s(k);
v_apparent = Uwind*sind(windAngles2(i));

theta = atan2(v_apparent,u_apparent)*180/pi;

Uapparent = sqrt(u_apparent^2 + v_apparent^2);

% Thrust and side force coefficients
Cx_current = interp1(windAngles, CxMax, theta, 'spline');
Cy_current = interp1(windAngles, CyMax, theta, 'spline');

Thrust(i) = Cx_current*sailFactor*Uapparent^2;
SideForce_wing(i) = Cy_current*sailFactor*Uapparent^2;

yawAngle(i) = interp1(SideForce(k, :), shipAngles, abs(SideForce_wing(i)), 'spline');

ResistanceWind(i) = interp1(shipAngles, Resistance(k, :), yawAngle(i), 'spline');

AddedResistance(i) = ResistanceWind(i) - Resistance(k, 1);

RealThrust(i) = Thrust(i) - AddedResistance(i);

KG(i) = (9.81*rho_s*nabla*(KB + BM)*stabSin - abs(SideForce_wing(i))*(heightSails/2 + D))/(9.81*
rho_s*nabla*stabSin - abs(SideForce_wing(i)));
end

figure(1);
plot(windAngles2, Thrust/Resistance(k, 1), '--','color', color(k, :))
plot(windAngles2, RealThrust/Resistance(k, 1), 'color', color(k, :))

figure(2);
plot(windAngles2, yawAngle, 'color', color(k, :))

figure(3);
plot(windAngles2, KG, 'color', color(k, :))

end

% Thrust plotting
figure(1);
legend('U_s=10knots,without_yaw_angle', 'U_s=10knots,with_yaw_angle', 'U_s=12.5knots,without_yaw_angle', 'U_s=12.5knots,with_yaw_angle', 'U_s=15knots,without_yaw_angle', 'U_s=15knots,with_yaw_angle')
xlabel('True_wind_direction[deg]')
ylabel('(Thrust-added_resistance)/Reference_resistance')
axis([0, 180, 0, 1])

% Yaw angle plotting
figure(2);
legend('U_s=10knots', 'U_s=12.5knots', 'U_s=15knots', 1)
xlabel('True_wind_direction[deg]')
ylabel('Required_Yaw_angle_of_ship[deg]')
axis([0, 180, 0, 3])

% KG plotting
figure(3)
ylabel('Maximum_value_of_KG[m]')
legend('U_s=10knots', 'U_s=12.5knots', 'U_s=15knots', 2)
xlabel('True_wind_direction[deg]')
axis([0, 180, 12, 12.2])

```

## 11.2.4 finalResultsSeries60.m

```

%clear all
close all
clear all

load('wingDataSeries60.mat')
load('ShipForcesSeries60.mat')

%% Plot Settings
color = zeros(3);
color(1, :) = [0, 0, 1.0];
color(2, :) = [0, 0.6, 0];
color(3, :) = [1.0, 0, 0];

figure(1);
hold on
figure(2);
hold on
figure(3);
hold on

```

```

%% Environment data
rho_m = 997.561; %kg/m^3
rho_s = 1025.9; %kg/m^3
nu_m = 1.19E-06; %m^2/s
nu_s = 8.91E-07; %m^2/s
rho_air = 1.225; %kg/m^3

Uwind = 7; %m/s

%% Sail data
nrSails = 8;
heightSails = 40; %m
chord = 8; %m
areaSail = heightSails*chord; %m^2
totalAreaSail = nrSails*areaSail; %m^2

sailFactor = 0.5*rho_air*totalAreaSail;

AspPhys = 5;
Asp = 2*AspPhys;

%% Ship data
shipAngles = 0:2.5:10;

%% Stability data
D = 6.5;
B = 16.3;
nabla = 7744;

KB = 3.5;
Ixx = 23191;
BM = Ixx/nabla;

stabSin = sind(5);

%% Different wind directions

windAngles2 = 15:1:150;
nrWind2 = length(windAngles2);

for k = 1:3
    % Initialize variables dependent on wind direction
    KG = zeros(1, nrWind2);

    Thrust = zeros(1, nrWind2);
    SideForce_wing = zeros(1, nrWind2);
    yawAngle = zeros(1, nrWind2);
    ResistanceWind = zeros(1, nrWind2);
    AddedResistance = zeros(1, nrWind2);
    RealThrust = zeros(1, nrWind2);

    % Go through each wind direction
    for i = 1:nrWind2
        % Calculate apparent wind
        u_apparent = Uwind*cosd(windAngles2(i)) + U_s(k);
        v_apparent = Uwind*sind(windAngles2(i));

        theta = atan2(v_apparent, u_apparent);

        Uapparent = sqrt(u_apparent^2 + v_apparent^2);

        % Thrust and side force coefficients
        Cx_current = interp1(windAngles, CxMax, theta*180/pi, 'spline');
        Cy_current = interp1(windAngles, CyMax, theta*180/pi, 'spline');

        Thrust(i) = Cx_current*sailFactor*Uapparent^2;
        SideForce_wing(i) = Cy_current*sailFactor*Uapparent^2;

        yawAngle(i) = interp1(SideForce(k, :), shipAngles, abs(SideForce_wing(i)), 'spline');

        ResistanceWind(i) = interp1(shipAngles, Resistance(k, :), yawAngle(i), 'spline');

        AddedResistance(i) = ResistanceWind(i) - Resistance(k, 1);

        RealThrust(i) = Thrust(i) - AddedResistance(i);

        KG(i) = (9.81*rho_s*nabla*(KB + BM)*stabSin - abs(SideForce_wing(i))*(heightSails/2 + D))/(9.81*
            rho_s*nabla*stabSin - abs(SideForce_wing(i)));
    end

    figure(1);
    plot(windAngles2, Thrust/Resistance(k, 1), '--','color', color(k, :))
    plot(windAngles2, RealThrust/Resistance(k, 1), 'color', color(k, :))

    figure(2);
    plot(windAngles2, yawAngle, 'color', color(k, :))

```

```

    figure(3);
    plot(windAngles2, KG, 'color', color(k, :))

end

% Thrust plotting
figure(1);
legend('U_s=10knots,without_yaw_angle', 'U_s=10knots,with_yaw_angle', 'U_s=12.5knots,without_yaw_angle', 'U_s=12.5knots,with_yaw_angle', 'U_s=15knots,without_yaw_angle', 'U_s=15knots,with_yaw_angle')
xlabel('True_wind_direction[deg]')
ylabel('(Thrust-added_resistance)/Reference_resistance')
axis([0, 180, 0, 1.7])

% Yaw angle plotting
figure(2);
legend('U_s=10knots', 'U_s=12.5knots', 'U_s=15knots', 1)
xlabel('True_wind_direction[deg]')
ylabel('Required_Yaw_angle_of_ship[deg]')
axis([0, 180, 0, 3.5])

% KG plotting
figure(3)
ylabel('Maximum_value_of_KG[m]')
legend('U_s=10knots', 'U_s=12.5knots', 'U_s=15knots', 2)
xlabel('True_wind_direction[deg]')

```

### 11.2.5 SailsOnlySeries60.m

```

close all
clear all

% Script that calculates the speed of series 60 for different wind
% idrections and wind speeds, using wing sails as the only propulsion

load('ShipForcesSailsOnlySeries60.mat')
load('FittedValues.mat')
load('wingDataSeries60.mat')

nrIterations = 60;
error = 0.001;

% Environment data
rho_m = 997.561; %kg/m^3
rho_s = 1025.9; %kg/m^3
nu_m = 1.19E-06; %m^2/s
nu_s = 8.91E-07; %m^2/s
rho_air = 1.225; %kg/m^3

% Sail data
nrSails = 8;
heightSails = 40; %m
chord = 8; %m
areaSail = heightSails*chord; %m^2
totalAreaSail = nrSails*areaSail; %m^2

sailFactor = 0.5*rho_air*totalAreaSail;

%% Wind Speed 7
U_wind = 7;

windAngles2 = 25:1:150;
nrWind2 = length(windAngles2);

U_noYaw = zeros(1, nrWind2);
U_Yaw = zeros(1, nrWind2);
theta = zeros(1, nrWind2);

[shipAngles2D, U_s2D] = meshgrid(0:2.5:10, U_s) ;
shipAngles = 0:2.5:10;

for i = 1:nrWind2
    % No yaw effects on resistance
    U_ship = 0;

    u_wind = U_wind*cosd(windAngles2(i));

    v_apparent = U_wind*sind(windAngles2(i));

    lookingForU = 1;

    n = 1;
    while lookingForU
        U_test = U_ship;

```

```

    u_apparent = u_wind + U_ship;
    U_apparent = u_apparent^2 + v_apparent^2;
    theta = atan2(v_apparent, u_apparent)*180/pi; %deg
    Cx_current = interp1(windAngles, CxMax, theta, 'spline');
    Thrust = Cx_current*sailFactor*U_apparent;

    U_ship = Ufit(0, Thrust);

    if abs(U_test - U_ship)/U_ship < error
        lookingForU = 0;
    elseif n > nrIterations
        lookingForU = 0;
        disp('convergence not reached')
    end

    n = n+1;
end

U_noYaw(i) = U_ship;

% Include yaw effects on resistance
U_ship = min(U_s);
yaw_ship = 0;

lookingForU = 1;

n = 1;
while lookingForU
    U_test = U_ship;

    u_apparent = u_wind + U_ship;
    U_apparent = u_apparent^2 + v_apparent^2;
    theta = atan2(v_apparent, u_apparent)*180/pi; %deg
    Cx_current = interp1(windAngles, CxMax, theta, 'spline');
    Cy_current = interp1(windAngles, CyMax, theta, 'spline');

    Side = Cy_current*sailFactor*U_apparent;
    Thrust = Cx_current*sailFactor*U_apparent;

    yaw_ship = angFit(U_ship, abs(Side));

    U_ship = Ufit(yaw_ship, Thrust);

    if abs(U_test - U_ship)/U_ship < error
        lookingForU = 0;
    elseif n > nrIterations
        lookingForU = 0;
        disp('convergence not reached')
    end

    n = n+1;
end

U_Yaw(i) = U_ship;
theta(i) = yaw_ship;
end

plot(windAngles2, U_noYaw/0.51444444, '--', 'color', 'blue')
hold on
plot(windAngles2, U_Yaw/0.51444444, 'color', 'blue')

%% Wind speed 10
U_wind = 10;

windAngles2 = 25:1:150;
nrWind2 = length(windAngles2);

U_noYaw = zeros(1, nrWind2);
U_Yaw = zeros(1, nrWind2);
theta = zeros(1, nrWind2);

[shipAngles2D, U_s2D] = meshgrid(0:2.5:10, U_s);
shipAngles = 0:2.5:10;

for i = 1:nrWind2
    % No yaw effects on resistance
    U_ship = 0;

    u_wind = U_wind*cosd(windAngles2(i));

```

```

v_apparent = U_wind*sind(windAngles2(i));
lookingForU = 1;
n = 1;
while lookingForU
    U_test = U_ship;

    u_apparent = u_wind + U_ship;

    U_apparent = u_apparent^2 + v_apparent^2;

    theta = atan2(v_apparent, u_apparent)*180/pi; %deg

    Cx_current = interp1(windAngles, CxMax, theta, 'spline');
    Thrust = Cx_current*sailFactor*U_apparent;

    U_ship = Ufit(0, Thrust);

    if abs(U_test - U_ship)/U_ship < error
        lookingForU = 0;
    elseif n > nrIterations
        lookingForU = 0;
        disp('convergence_not_reached')
    end

    n = n+1;
end

U_noYaw(i) = U_ship;

% Include yaw effects on resistance
U_ship = min(U_s);
yaw_ship = 0;

lookingForU = 1;
n = 1;
while lookingForU
    U_test = U_ship;

    u_apparent = u_wind + U_ship;

    U_apparent = u_apparent^2 + v_apparent^2;

    theta = atan2(v_apparent, u_apparent)*180/pi; %deg

    Cx_current = interp1(windAngles, CxMax, theta, 'spline');
    Cy_current = interp1(windAngles, CyMax, theta, 'spline');

    Side = Cy_current*sailFactor*U_apparent;
    Thrust = Cx_current*sailFactor*U_apparent;

    yaw_ship = angFit(U_ship, abs(Side));

    U_ship = Ufit(yaw_ship, Thrust);

    if abs(U_test - U_ship)/U_ship < error
        lookingForU = 0;
    elseif n > nrIterations
        lookingForU = 0;
        disp('convergence_not_reached')
    end

    n = n+1;
end

U_Yaw(i) = U_ship;
theta(i) = yaw_ship;
end

plot(windAngles2, U_noYaw/0.51444444, '--', 'color', [0, 0.6, 0])
plot(windAngles2, U_Yaw/0.51444444, 'color', [0, 0.6, 0])

%% Wind speed 5
U_wind = 5;

windAngles2 = 25:1:150;
nrWind2 = length(windAngles2);

U_noYaw = zeros(1, nrWind2);
U_Yaw = zeros(1, nrWind2);
theta = zeros(1, nrWind2);

[shipAngles2D, U_s2D] = meshgrid(0:2.5:10, U_s);
shipAngles = 0:2.5:10;

```

```

for i = 1:nrWind2
    % No yaw effects on resistance
    U_ship = 0;

    u_wind = U_wind*cosd(windAngles2(i));
    v_apparent = U_wind*sind(windAngles2(i));

    lookingForU = 1;

    n = 1;
    while lookingForU
        U_test = U_ship;

        u_apparent = u_wind + U_ship;

        U_apparent = u_apparent^2 + v_apparent^2;

        theta = atan2(v_apparent, u_apparent)*180/pi; %deg

        Cx_current = interp1(windAngles, CxMax, theta, 'spline');
        Thrust = Cx_current*sailFactor*U_apparent;

        U_ship = Ufit(0, Thrust);

        if abs(U_test - U_ship)/U_ship < error
            lookingForU = 0;
        elseif n > nrIterations
            lookingForU = 0;
            disp('convergence_not_reached')
        end

        n = n+1;
    end

    U_noYaw(i) = U_ship;

    % Include yaw effects on resistance
    U_ship = min(U_s);
    yaw_ship = 0;

    lookingForU = 1;

    n = 1;
    while lookingForU
        U_test = U_ship;

        u_apparent = u_wind + U_ship;

        U_apparent = u_apparent^2 + v_apparent^2;

        theta = atan2(v_apparent, u_apparent)*180/pi; %deg

        Cx_current = interp1(windAngles, CxMax, theta, 'spline');
        Cy_current = interp1(windAngles, CyMax, theta, 'spline');

        Side = Cy_current*sailFactor*U_apparent;
        Thrust = Cx_current*sailFactor*U_apparent;

        yaw_ship = angFit(U_ship, abs(Side));

        U_ship = Ufit(yaw_ship, Thrust);

        if abs(U_test - U_ship)/U_ship < error
            lookingForU = 0;
        elseif n > nrIterations
            lookingForU = 0;
            disp('convergence_not_reached')
        end

        n = n+1;
    end

    U_Yaw(i) = U_ship;
    theta(i) = yaw_ship;
end

plot(windAngles2, U_noYaw/0.51444444, '--', 'color', 'red')
plot(windAngles2, U_Yaw/0.51444444, 'color', 'red')

legend('Wind speed=7m/s, without yaw', 'Wind speed=7m/s', 'Wind speed=10m/s, without yaw', 'Wind speed=10m/s', 'Wind speed=5m/s, without yaw', 'Wind speed=5m/s', 4)

axis([0, 180, 0, 17])
xlabel('True wind direction [deg]')
ylabel('Ship speed [knots]')

```