pySailingVLM Documentation

Zuzanna Wieczorek, Grzegorz Gruszczyński

Jun 02, 2023

CONTENTS

The beginning of fluid dynamics has started in acient Greece, when Archimedes investigated the fluid statics and bouancy. From that time, many well-konwn mathematicians and engineers have contributed to defining how air and gases behave. The equations they have constructed are extremely complex and difficult to calculate by hand when the geometry of airfoil is complicated. The rapid improvement of computers' power in 1970s has led to developing complex flow desings. Over the past few decades Computational Fluid Dynamics (CFD) has been improved dramatically.

There are many methods, that are used in Computational Fluid Dynamics. Some of them are Finite Element Method (FEM), Volume of Fluid (VOF) and Lattice Boltzmann Method (LBM). They provide good and accurate results but require high amount of computational resources. To overcome this problem, low fidelity tools can be used, like Vortex Lattice Method (VLM). Using VLM is benefical in the early conceptual design phase [NJS+21], when many different desings should be tested. Thanks to fast computations of aerodynamics forces and pressures, engineers can quickly overview modelled design performance.

In this work, we present the first open source Python package which implements Vortex Lattice Method for initial aerodynamic analysis of upwind sails. Thanks to its light weight requirements, the software can be immediately installed and executed locally or accessed by cloud environment such as Google Collab. Additionally, package users can define own sail geometries and use pySailingVLM inside custom scripts which makes creating a set of dynamics very convenient.

Keywords: Vortex Lattice Method (VLM), initial sail analysis, yacht engineering, Python Package

Inside documentation:

Part I

Description

CHAPTER

ONE

THEORY

The Vortex Lattice Method (VLM) is a numerical method used in computational fluid dynamics. VLM models a surface on aircraft as infinite vortices to estimate the lift curve slope, induced drag, and force distribution. The VLM is the extension of Prandtl's lifting-line theory that is capable of computing swept and low aspect ratio wings. [SHR+19].

1.1 Introduction

Vortex Lattice Method is build on the Potential flow theory. The viscous effects, drag and flow separation in VLM is sufficient for aproximatting ideal flow seen in nature.

1.2 Potential flow theory

Potenttial Flow Theory treats external flows around bodies as invicid and irrotational [Tec05] .The viscous effects are limitted to a thin layer next to the body (boundary layer). Becouse of this and separation phenomena such fluid can be modelled only with small angle of attack. In irrotational flow fluid particles are not rotating.

In Potential Flow Theory, because velocity must satisfy the conservation of mass equation, the Laplace Equation is qoverning:

$$\nabla^2\phi = 0$$

where ϕ is a potentinal function defined as continous function which satisfies the conservation of mass and momentum (incompressible, inviscid and irrational flow).

CHAPTER

TWO

VORTEX LATTICE METHOD

The Vortex Lattice Method is a panel method where wing or other configuration is modelled by a large number of elementary, quadrilateral panels lying either on the actual aircraft surface, or on some mean surface, or combination thereof. To satisfy boundary conditions VLM uses following elements attached to each panel:

- source - a point from which fluid issues and flows radially outward such that the continuity equation is satisfied everywhere but at the singularity that exists at the source's center
- sink - a negative source
- doublet - singularity resulting when a source or a sink of equal strength are made to approach each other such that the product of their strangths and their distance apart remains constant at a preselected finite value in the limit as a distance between them approaches zero
- vortex - an element that generates a circulation, or tangential motion, around its origin

Such singularities are defined by specifying functional variation across the panel and its value is set by determinating strength parameters. Such parameters are known after solving appropriate boundary condition equations. When singularity strengts are determinated, the pressure, velocity can be computed[JJB09].

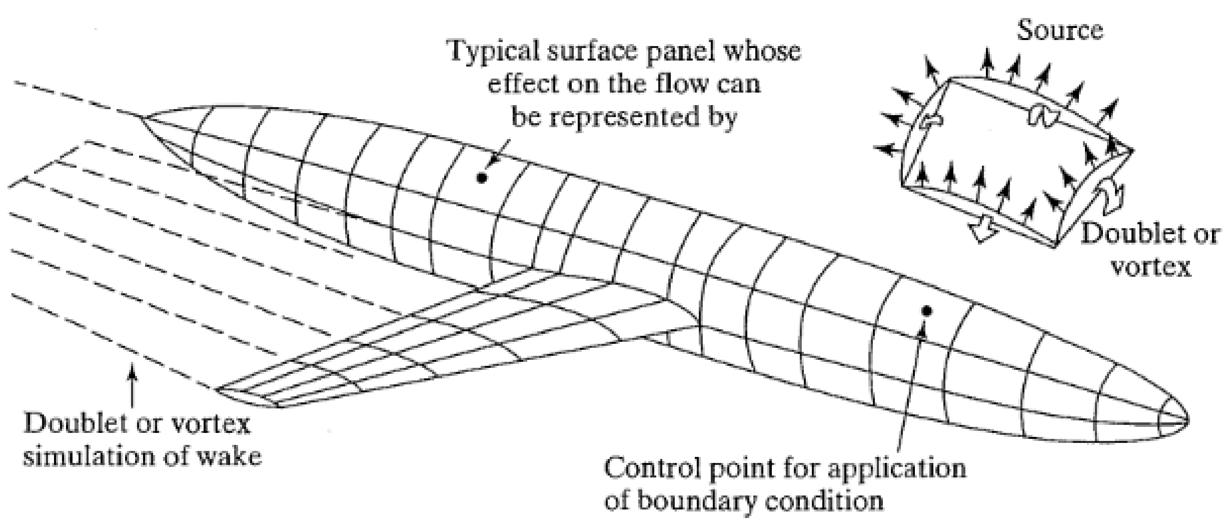


Fig. 2.1: Representation of an airplane flowfield by panel (or singularity) methods. Figure taken from JJB09.Fig. 7.22 page 350).

2.1 Vortices

The vortex theorems for inviscid incompressible flows has been developed by German scientist Hermann von Helmholtz (1821-1894). The theory is based on the following assumptions [JK01]:

- Flow is inviscid and incompressible
- The strength of vortex filament is constant along its length
- A vortex filament cannot start or end in a fluid (it must form or closed path or extend to infinity)
- The fluid that forms a vortex tube continues to form a vortex tube and the strength of the vortex tube remains constant as the tube moves about.

2.1.1 Vortex element

Vortex element in 2D is presented on figure 2.2. It is created by placing a rotating cylindrical core in two dimentional flowfield. If the cylinder has a radius R and a constant angular velocity of ω_y, then its motion results in a flow with circular streamlines [JK01].

The tangential velocity induced is described by equation:

$$q_\theta(r) = \frac{\Gamma}{2\pi \cdot r}$$

where r is described as distance from the vortex core.

If a vortex is located in free-strem with uniform velocity u_∞ then the total velocity at the distance r can be writtes as $\overrightarrow{u_\infty} + \overrightarrow{q_\theta(r)}$ [AH13].

2.1.2 Vortex segment

The velocity induced by a straight vortex segement (figure 2.3) in three dimentions with given vortex strength Γ at point $P(x, y, z)$ is qiven by equation:

$$\overrightarrow{q_{1,2}} = \frac{\Gamma}{4\pi} \frac{\overrightarrow{r_1} \times \overrightarrow{r_2}}{|\overrightarrow{r_1} \times \overrightarrow{r_2}|^2} \left(\frac{\overrightarrow{r_1}}{||\overrightarrow{r_1}||} - \frac{\overrightarrow{r_2}}{||\overrightarrow{r_2}||} \right) \cdot \overrightarrow{r_0} = \vec{\nu}\,\Gamma \tag{2.1}$$

where

$$\overrightarrow{r_0} = \overrightarrow{r_1} - \overrightarrow{r_2}$$

and

$$\begin{aligned} \overrightarrow{r_0} &= (x_2, y_2, z_2) - (x_1, y_1, z_1) \\ \overrightarrow{r_1} &= P - (x_1, y_1, z_1) \\ \overrightarrow{r_2} &= P - (x_2, y_2, z_2) \end{aligned}$$

In case of a semi-infinite vortex line, the point 2 is infinitely far away thus formula reads [PS00]:

$$\overrightarrow{q_{1,2}} = \frac{\Gamma}{4\pi} \frac{\overrightarrow{u_\infty} \times \overrightarrow{r_1}}{||\overrightarrow{r_1}||(||\overrightarrow{r_1}|| - \overrightarrow{u_\infty} \cdot \overrightarrow{r_1})} = \vec{\nu}\,\Gamma \tag{2.2}$$

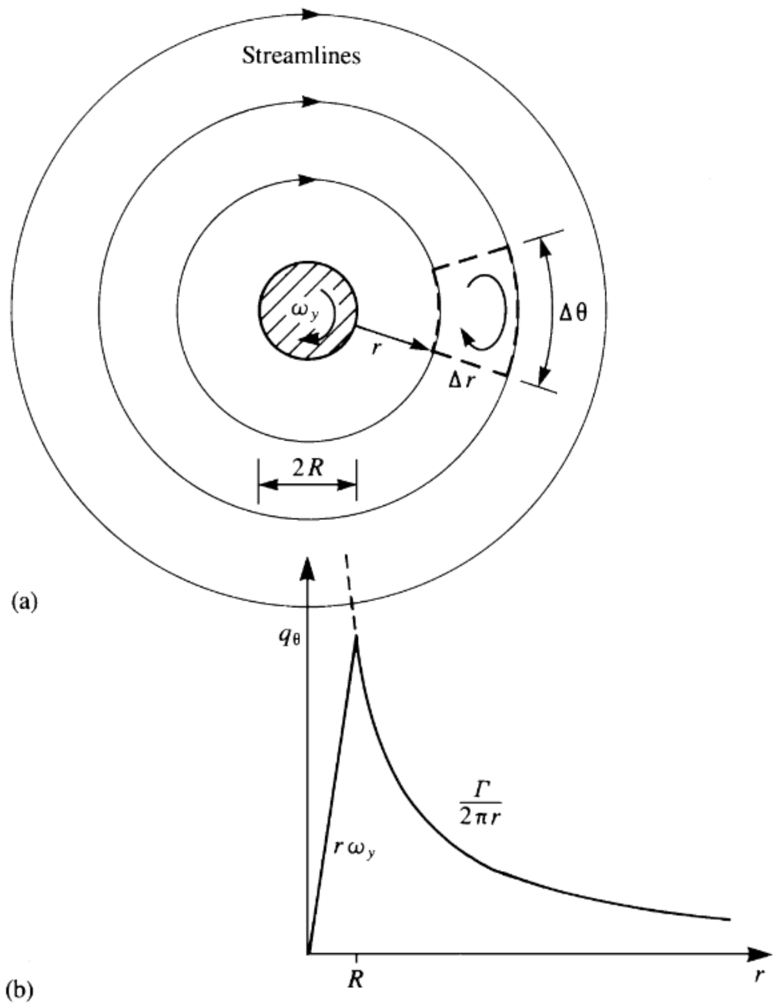


Fig. 2.2: Two dimentional flowfield around a cylindrical core rotating as a rigid body. Figure taken from [JK01] (Fig. 2.11 page 34).

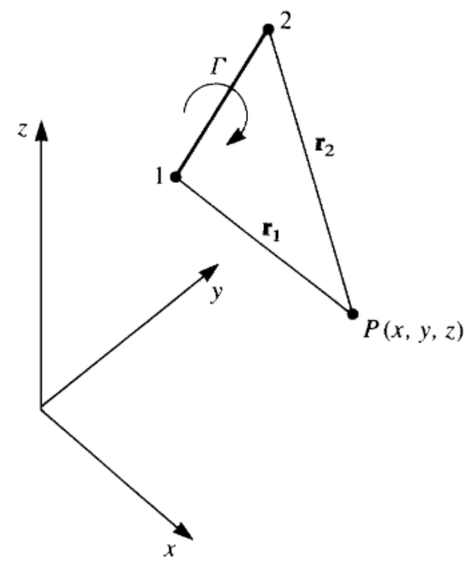


Fig. 2.3: Vortex segment in three dimentions. Figure taken from [JK01] (Fig. 2.16 page 40).

2.2 The Kutta Condition

The Kutta Condition states that at small angle of attack the flow leaves the sharp trailing edge of an airfoil smoothly and the velocity is finite there [JK01]. Because of this the normal component of velocity, from both sides of the airfoil, must vanish. Circulation at trailing edge can be expressed by equation:

$$q_{T.E.} = 0$$

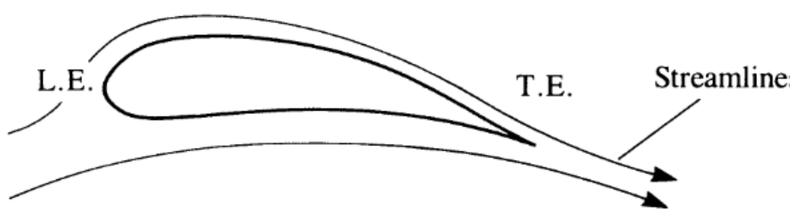


Fig. 2.4: Flow near cusped trailing edge. Figure taken from [JK01] (Fig. 4.12 page 89)

2.3 Lifting surface

The surface of the object is divided into panels (gray rectanges on figure 2.5). The total amount of them is a product of number of panels spanwise and chordwise. Vortex element is associated to each one of them (pink rectangles).

There are two types of vortex elements:

- vortex ring
- vortex horseshoe

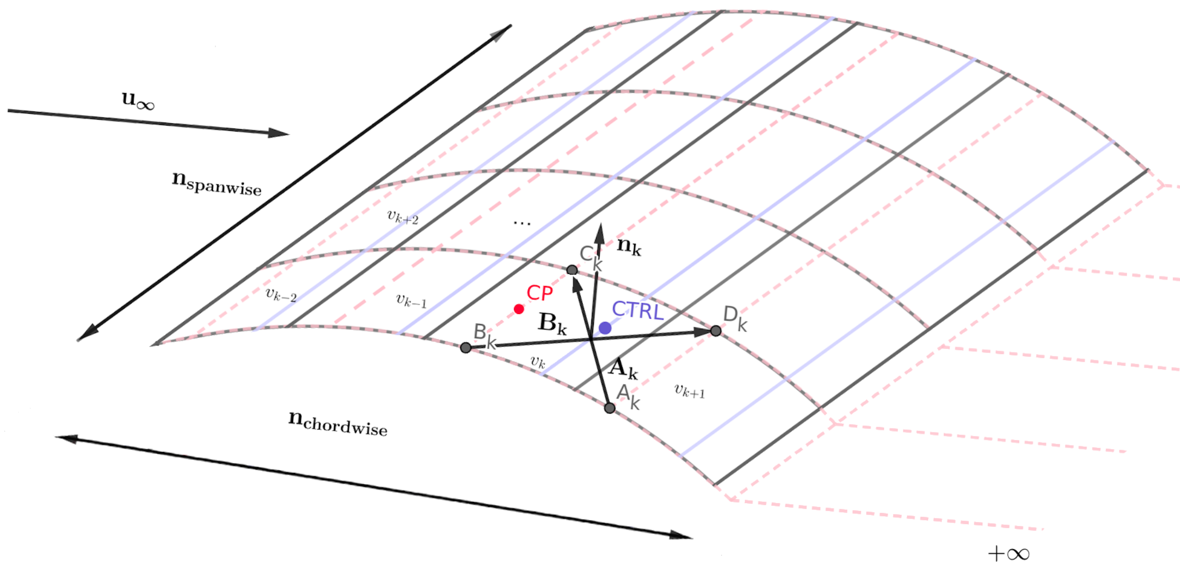


Fig. 2.5: Nomenclature of the lattice elements. Figure created by author.

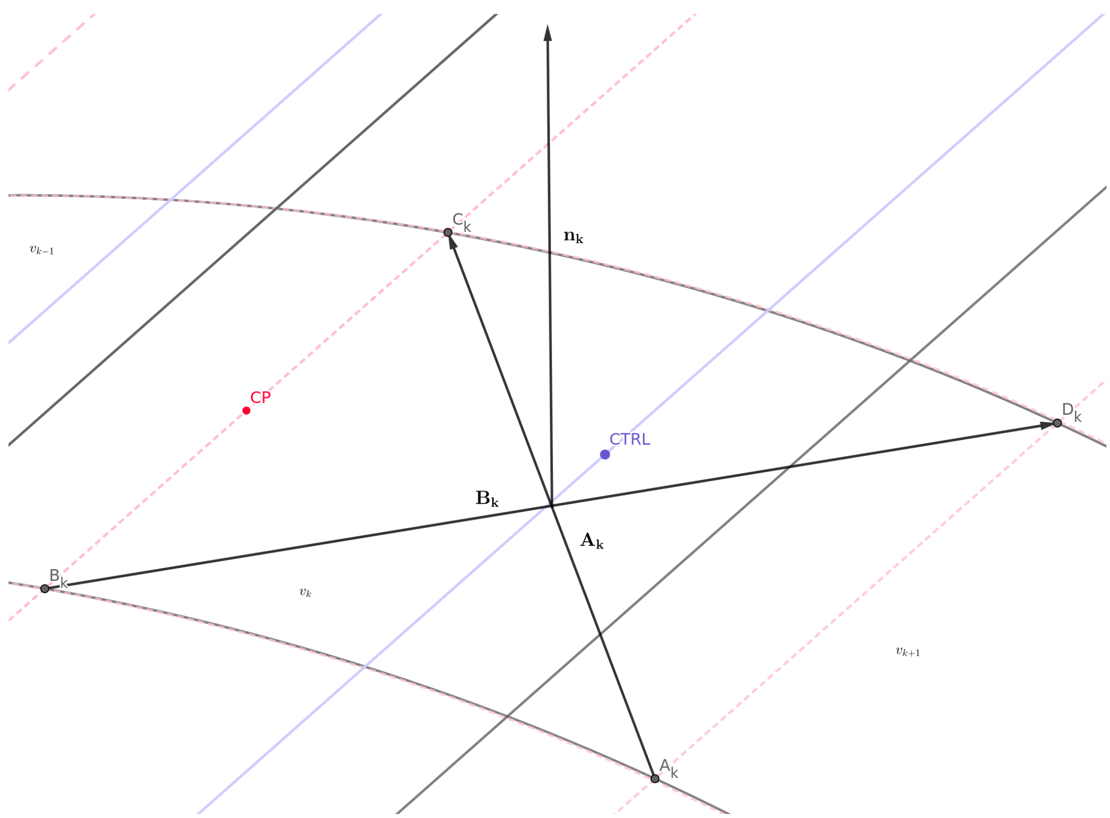


Fig. 2.6: Nomenclature of the vortex ring. Figure created by author.

Vortex ring (figure 2.6) is created by four finite segments according to equation (2.1). Horseshoe vortex is attached to the trailing edge of lifting surface. It consists of three finite and two semi-infinite segments following equation (2.1) and (2.2) (see figure 2.7). By placing vortex at the quarter chord line of the the two-dimensional Kutta condition is satisfied along the chord. Also, along the wing trailing edges, the trailing vortex of the last panel row must be canceled to satisfy the three-dimensional trailing-edge condition.

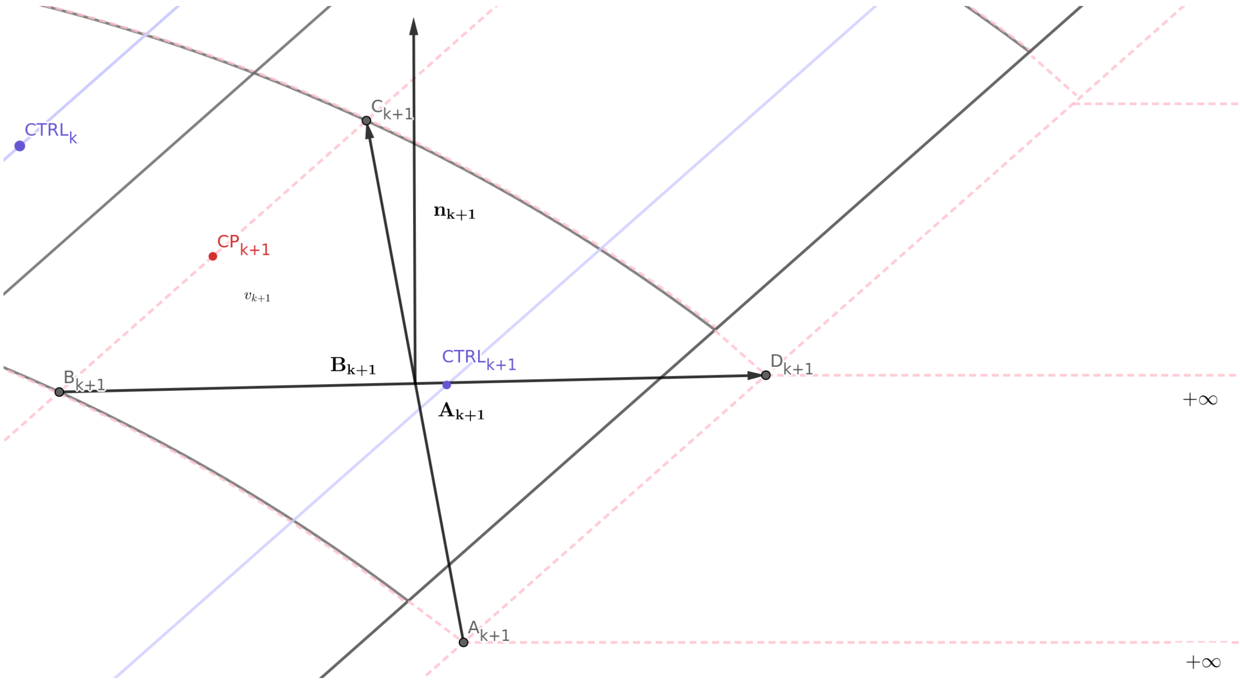


Fig. 2.7: Nomenclature of the vortex horseshoe. Figure created by author.

When thin surface is angled to a free-stream $\overrightarrow{u_\infty}$, the aerodynamic force is being generated at center of pressure. This point is located at the $1/4$ of a panel chord and at $1/2$ of the panel span (from panel leading edge). To fulfill no flow through the surface, the control point is defined at $3/4$ of the chord from the panel leading edge and in the middle of the span.

Vortex vertices B_k and C_k are placed at $1/4$ of panel chord (v_k), A_k and D_k at $1/4$ of the next panel (v_{k+1}). The panel opposite corner points define two vectors $\overrightarrow{A_k}$ and $\overrightarrow{B_k}$, and their vector product will point in the direction of $\overrightarrow{n_k}$ (normal vector).

2.3.1 Computations

To converse which claims that there should be of the k-th panel, the equation below is set:

If the velocity induced at k-th panel is $\overrightarrow{q_{ind}}$, no flow through the surface (boundary condition) is fulfield when:

$$(\overrightarrow{u_\infty} + \overrightarrow{q_{ind}}) \cdot \overrightarrow{n_k} = 0 \tag{2.3}$$

After equation transformation:

$$\sum_j \overrightarrow{\nu_{kj}} \Gamma_j \cdot \overrightarrow{n_k} = -\overrightarrow{u_\infty} \cdot \overrightarrow{n_k} \tag{2.4}$$

where $\overrightarrow{\nu_{kj}}$ is defined as coefficient of proportionality of induced velocity $\overrightarrow{q_{ind}}$ at k-th control point by j-th vortex.

By expanding equation (2.3) and (2.4) the RHS coefficient vector can be computed:

$$RHS_k = -\overrightarrow{u_\infty} \cdot \overrightarrow{n_k}$$

In order to obtain gamma magnitude at k-th panel, the following set of algebraic equations must be solved:

$$\begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{bmatrix} \begin{bmatrix} \Gamma_1 \\ \vdots \\ \Gamma_m \end{bmatrix} = \begin{bmatrix} RHS_1 \\ \vdots \\ RHS_m \end{bmatrix} \tag{2.5}$$

where $m = n_{spanwise} \cdot n_{chordwise}$ and $a_{kj} = \overrightarrow{\nu_{kj}} n_k$

The aerodynamics force can be expressed according to the Kutta-Joukowski theorem as:

$$\overrightarrow{F_{areo}} = \rho \overrightarrow{u_\infty} \times \vec{b}\Gamma$$

Where $\vec{b}$ is span vector.

Discretizing, the aerodunamic force corresponding to the j-th section is:

$$\begin{aligned} \overrightarrow{F_j} = \rho \overrightarrow{V}_{app\_wind\_fs\_j} \times \overrightarrow{b_j} \Gamma_j = \\ \rho(\overrightarrow{V}_{app\_wind\_infs\_j} + \vec{q}_{ind\_j}) \times \overrightarrow{b_j} \Gamma_j = \\ \rho(\overrightarrow{V}_{app\_wind\_infs\_j} + \sum \overrightarrow{\nu_{jk}} \Gamma_k) \times \overrightarrow{b_j} \Gamma_j \end{aligned}$$

where b_j is a vector representing a finite vortex filement going through the center of pressure of the j-th section, $\overrightarrow{V}_{app\_wind\_infs\_j}$ is apparent wind velocity for an 'infinite sail' (without induced wind velocity) and $\overrightarrow{V}_{app\_wind\_fs\_j}$ is apparent wind velocity for a finite sail' (with induced wind velocity).

Lift is defined as the component of the aerodynamic force that is perpendicular to the flow direction ($\overrightarrow{u_\infty}$) and can is defined as a dot product of force and normal vector:

$$L_k = \overrightarrow{F_k} \cdot \overrightarrow{n_k}$$

Pressure at k-th panel:

$$p_k = \frac{L_k}{S_k}$$

Pressure coefficient at k-th panel can be defined as:

$$c_p = \frac{p_k}{\frac{1}{2}\rho \left\| \overrightarrow{u_{\infty_k}} \right\|^2}$$

CHAPTER
THREE

TECHNOLOGY

The main purpose of the thesis was to optimize the existing code for initial sail analysis. The surface of the geometry used is defined with coordinates in three dimensions (figure 3.1) with the x-axis in the yachts longitudinal (chordwise) direction positivevbackwards, the y-axis in the transverse direction positive to starboard and the z-axis in spanwise direction positive upwards.

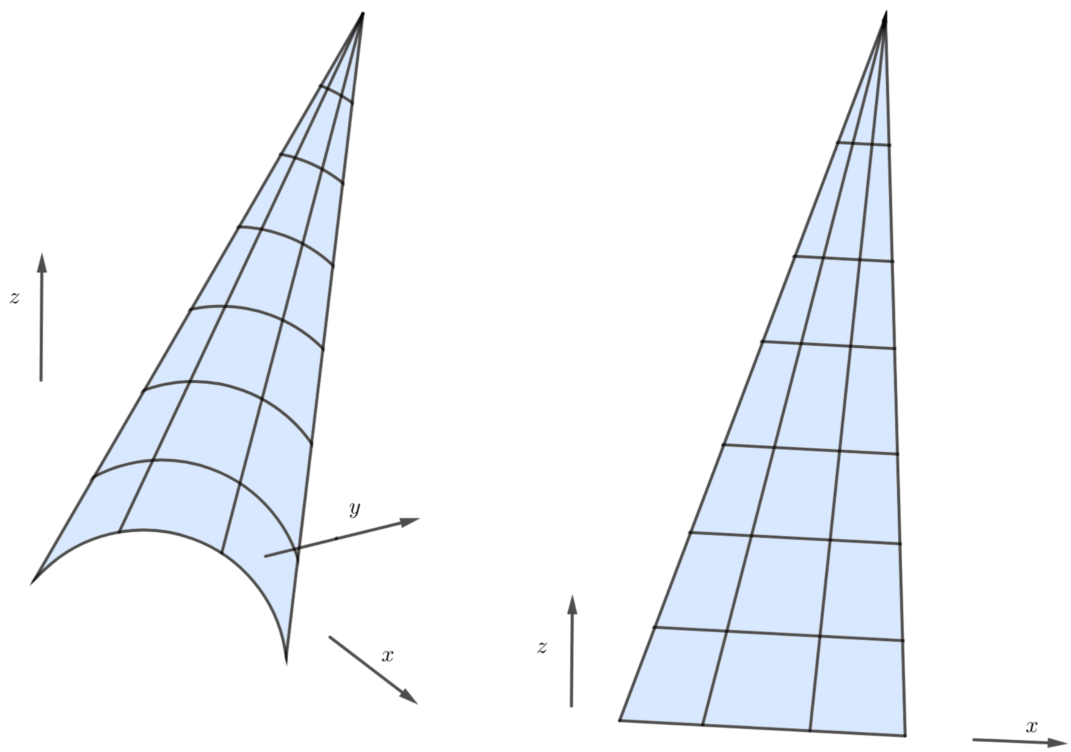


Fig. 3.1: Discretization of jib sail into 3 chordwise and 7 spanwise panels each. Figure created by author.

Orginally the code was written in pure OPP (Object Oriented Programming) Python which led to time consuming computations. To solve this problem, the code was rewritten in a non-objective way. Figure 3.2 shows data layout implemented. Insted of creating many Python objects, data such as coordinates of panels, pressure coefficients were arranged into sets of arrays. Thanks to this, the code has become less complicated and the Numba (a JIT compiler that translates Python code and NumPy into machine code) enabled parallel coalcuations.

To benchmark pySalingVLM, the time comparison tests were conducted. Three approaches was summaries in the table 3.1: objective code, objective code with Numba and non-objective compiled with Numba. The tests were carried out on

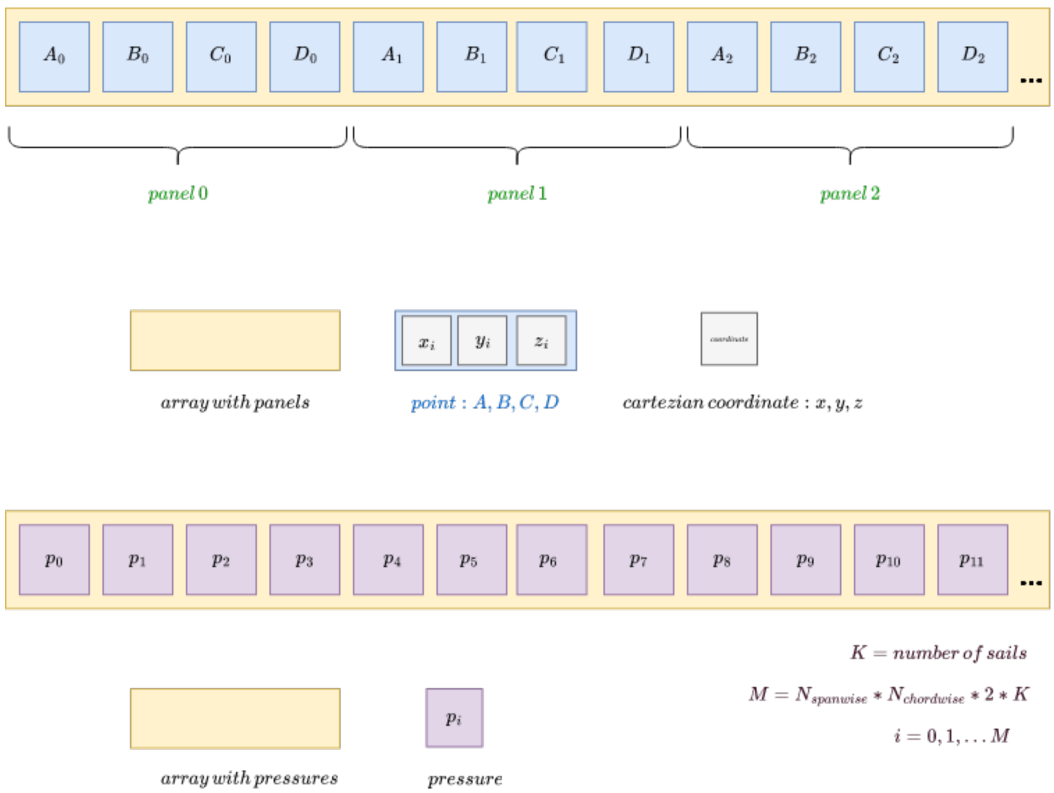


Fig. 3.2: Data layout implemented in pySailingVLM. Figure created by author.

a laptop with the following parameters: AMD Ryzen 7 4800H with Radeon, 2.9 GHz, 32 GB RAM, Nvidia GeForce graphics card GTX 1650. For 20x20 shape more then thirty times acceleration was obtained.

Table 3.1: Time execution comparison between different approaches of implementing pySailingVLM depending on sail shape.

$n_{spanwise} \times n_{chordwise}$	objective [s]	objective + numba [s]	non-objective + numba [s]
5x5	16.51	1.71	1.04
10x10	269.13	21.81	10.41
15x10	593.85	48.44	22.51
20x20	4325.78	320.54	143.69
30x20	no data	724.70	320.12
30x30	no data	1653.52	706.62

Moreover, the Vortex Lattice Method implementation was improved: the horseshoe vortex was introduced and is used at trailing edge (insted of vortex ring). New approach has added possibility to calculate cambered sailis and visualize pressure coefficients on colormap. Code has been packaged and now is available at PyPi. It can be run locally from command line or in a cloud using Jupyter Notebook. pySailingVLM can be executed in a user defined script, which make it easy to calculate and compare many sailing cases.

Part II

Getting Started

CHAPTER

FOUR

INSTALLATION

4.1 For Users

pySailingVLM package is available at PyPI, to install it do:

```
pip install pySailingVLM
```

4.2 For Developers

If you would like to dive into pySailingVLM code and test it on your own please do the folowing steps:

- Clone git project

```
git clone #TODO wsadzic tutaj link
```

- Go into project

```
cd #TODO project
```

- There is no requirements.txt file, for installing dependencies install a project in editable mode (i.e. setuptools "develop mode") from a local project path:

```
pip install -q -e .
```

4.2.1 Create package

Building package requires setup.cfg and pyproject.toml files located in main tree of pySailingVLM project. If you do not have the latest pip build package do:

```
pip install --upgrade build
```

Then:

```
pip install build
```

Build and install

```
python3 -m build
```

```
pip install dist/pySailingVLM-VERSION.tar.gz
```

CHAPTER

FIVE

USAGE

Note: First usage of pySailingVLm will produce numba warining. This behaviour is correct. Warining messages will disappear during second run of program.

Example warning:

/home/user/miniconda3/envs/sv_build_test_2/lib/python3.10/site-packages/numba/core/lowering.py:107: NumbaDebugInfoWarning: Could not find source for function: <function __numba_array_expr_0x7fbf333f1780 at 0x7fbf33361b40>. Debug line information may be inaccurate.

5.1 Jupyter Notebook

For Jupyter Notebook examples see Usage subsection.

5.2 Command line

5.2.1 Input file

In order to run pySailingVLM from command line you must provide a variable.py file. Example file is shown below. Modify it for your needs.

```
import os
import numpy as np
import time

mgirths =  np.array([0.00, 1./8, 1./4, 1./2, 3./4, 7./8, 1.00])
jgirths = np.array([0.00, 1./4, 1./2, 3./4, 1.00])

output_args = {
    'case_name': os.path.basename(__file__),  # get name of the current file
    'case_dir': os.path.abspath(''), # get dir of the current file
    'name': os.path.join("results_example_jib_and_mainsail_vlm", time.strftime("%Y-%m-
↪%d_%Hh%Mm%Ss")),
    'file_name': 'my_fancy_results', # name of xlsx excel file
}

solver_args = {
    'n_spanwise':  15,  # No of control points (above the water) per sail,␣
↪recommended: 50
```

(continues on next page)

(continued from previous page)

```
    'n_chordwise': 10, # No of control points (above the water) per sail,␣
↪recommended: 50
    'interpolation_type': "spline",  # either "spline" or "linear"
    'LLT_twist': "real_twist",  # defines how the Lifting Line discretize the sail␣
↪twist.
}

conditions_args = {
    'leeway_deg': 5.,     # [deg]
    'heel_deg': 10.,      # [deg]
    'SOG_yacht': 4.63,    # [m/s] yacht speed - speed over ground (leeway is a␣
↪separate variable)
    'tws_ref': 4.63,      # [m/s] true wind speed
    'alpha_true_wind_deg': 50.,   # [deg] true wind angle (with reference to course␣
↪over ground) => Course Wind Angle to the boat track = true wind angle to centerline␣
↪+ Leeway
    'reference_water_level_for_wind_profile': -0.,  # [m] this is an attempt to␣
↪mimick the deck effect
    # by lowering the sheer_above_waterline
    # while keeping the wind profile as in original geometry
    # this shall be negative (H = sail_ctrl_point - water_level)
    'wind_exp_coeff': 0.1428,  # [-] coefficient to determine the exponential wind␣
↪profile
    'wind_reference_measurment_height': 10.,  # [m] reference height for exponential␣
↪wind profile
    'rho': 1.225,  # air density [kg/m3]
    'wind_profile': 'exponential', # allowed: 'exponential' or 'flat' or 'logarithmic'
    'roughness': 0.05, # for logarithmic profile only
}

rig_args = {
    'main_sail_luff': 12.4,  # [m]
    'jib_luff': 10.0,  # [m]
    'foretriangle_height': 11.50,  # [m]
    'foretriangle_base': 3.90,  # [m]
    'sheer_above_waterline': 1.2,#[m]
    'boom_above_sheer': 1.3, # [m],
    'rake_deg': 92. ,  # rake angle [deg]
    'mast_LOA': 0.15,  # [m]
    'sails_def': 'jib_and_main', # definition of sail set, possible: 'jib' or 'main'␣
↪or 'jib_and_main'
}
# INFO for camber:
# First digit describing maximum camber as percentage of the chord.
# Second digit describing the distance of maximum camber from the airfoil leading␣
↪edge in tenths of the chord.
main_sail_args = {
    'girths' : mgirths,
    'chords': np.array([4.00, 3.82, 3.64, 3.20, 2.64, 2.32, 2.00]),
    'centerline_twist_deg': 12 * mgirths + 5,
    'camber': 5*np.array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]),
}

jib_sail_args = {
    'centerline_twist_deg': 15. * jgirths + 7,
```

(continues on next page)

(continued from previous page)

```
    'girths': jgirths,
    'chords': np.array([3.80, 2.98, 2.15, 1.33, 0.5]),
    'camber': 5*np.array([0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5]), # starting from␣
↪leading edge
}

# REFERENCE CSYS
# The origin of the default CSYS is located @ waterline level and aft face of the mast
# The positive x-coord: towards stern
# The positive y-coord: towards leeward side
# The positive z-coord: above the water
# To shift the default CSYS, adjust the 'reference_level_for_moments' variable.
# Shifted CSYS = original + reference_level_for_moments
# As a results the moments will be calculated around the new origin.

# yaw_reference [m] - distance from the aft of the mast towards stern, at which the␣
↪yawing moment is calculated.
# sway_reference [m] - distance from the aft of the mast towards leeward side. 0 for␣
↪symmetric yachts ;)
# heeling_reference [m] - distance from the water level,  at which the heeling moment␣
↪is calculated.
csys_args = {
    'reference_level_for_moments': np.array([0, 0, 0]),  # [yaw_reference, sway_
↪reference, heeling_reference]
}

# GEOMETRY OF THE KEEL
# to estimate heeling moment from keel, does not influence the optimizer.
# reminder: the z coord shall be negative (under the water)
keel_args={
    'center_of_lateral_resistance_upright': np.array([0, 0, -1.0]),  # [m] the␣
↪coordinates for a yacht standing in upright position
}
```

5.2.2 Run script

To run script with variables.py located in working directory do:

```
pySailingVLM
```

If variables.py is located in different directory, you must sepecify its location by providing additional option for pySailingVLM script:

```
pySailingVLM --dvars path_to_foler_with_variables.py
```

More information is available inside script help:

```
pySailingVLM --help
```

5.2.3 Output

pySailingVLM produces output files: data in xlsx file, matplotlib figures and pressure coefficient colormap in html extention (interactive plot). It is saved in the location specified in varaibles.py.

5.3 Cambered jib and main example

5.4 Example

In cell below insert your initial parameters. If some of them are not required for your model, simply pass 0 value (for numbers). Some parameters are necessary only for specific cases like roughness (used by package when logarithmic profile is set) and they are omitted during computation.

More information can be found in code comments below.

```
# varaibles.py for jupyter
import os
import numpy as np
import time

mgirths =  np.array([0.00, 1./8, 1./4, 1./2, 3./4, 7./8, 1.00])
jgirths = np.array([0.00, 1./4, 1./2, 3./4, 1.00])

output_args = {
    'case_name': 'my_case_name',  # get name of the current file
    'case_dir': os.path.abspath(''), # get dir of the current file
    'name': os.path.join("results_example_jib_and_mainsail_vlm", time.strftime("%Y-%m-
↪%d_%Hh%Mm%Ss")),
    'file_name': 'my_fancy_results', # name of xlsx excel file
}

solver_args = {
    'n_spanwise':  15,  # No of control points (above the water) per sail,␣
↪recommended: 50
    'n_chordwise': 10, # No of control points (above the water) per sail,␣
↪recommended: 50
    'interpolation_type': "spline",  # either "spline" or "linear"
    'LLT_twist': "real_twist",  # defines how the Lifting Line discretize the sail␣
↪twist.
}

conditions_args = {
    'leeway_deg': 5.,    # [deg]
    'heel_deg': 10.,     # [deg]
    'SOG_yacht': 4.63,   # [m/s] yacht speed - speed over ground (leeway is a␣
↪separate variable)
    'tws_ref': 4.63,     # [m/s] true wind speed
    'alpha_true_wind_deg': 50.,   # [deg] true wind angle (with reference to course␣
↪over ground) => Course Wind Angle to the boat track = true wind angle to centerline␣
↪+ Leeway
    'reference_water_level_for_wind_profile': -0.,  # [m] this is an attempt to␣
↪mimick the deck effect
    # by lowering the sheer_above_waterline
    # while keeping the wind profile as in original geometry
```

(continues on next page)

(continued from previous page)

```
    # this shall be negative (H = sail_ctrl_point - water_level)
    'wind_exp_coeff': 0.1428,  # [-] coefficient to determine the exponential wind␣
↪profile
    'wind_reference_measurment_height': 10.,  # [m] reference height for exponential␣
↪wind profile
    'rho': 1.225,  # air density [kg/m3]
    'wind_profile': 'exponential', # allowed: 'exponential' or 'flat' or 'logarithmic'
    'roughness': 0.05, # for logarithmic profile only
}

rig_args = {
    'main_sail_luff': 12.4,  # [m]
    'jib_luff': 10.0,  # [m]
    'foretriangle_height': 11.50,  # [m]
    'foretriangle_base': 3.90,  # [m]
    'sheer_above_waterline': 1.2,#[m]
    'boom_above_sheer': 1.3, # [m],
    'rake_deg': 92. ,  # rake angle [deg]
    'mast_LOA': 0.15,  # [m]
    'sails_def': 'jib_and_main', # definition of sail set, possible: 'jib' or 'main'␣
↪or 'jib_and_main'
}
# INFO for camber:
# First digit describing maximum camber as percentage of the chord.
# Second digit describing the distance of maximum camber from the airfoil leading␣
↪edge in tenths of the chord.
main_sail_args = {
    'girths' : mgirths,
    'chords': np.array([4.00, 3.82, 3.64, 3.20, 2.64, 2.32, 2.00]),
    'centerline_twist_deg': 12 * mgirths + 5,
    'camber': 5*np.array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]),
}

jib_sail_args = {
    'centerline_twist_deg': 15. * jgirths + 7,
    'girths': jgirths,
    'chords': np.array([3.80, 2.98, 2.15, 1.33, 0.5]),
    'camber': 5*np.array([0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5]), # starting from␣
↪leading edge
}

# REFERENCE CSYS
# The origin of the default CSYS is located @ waterline level and aft face of the mast
# The positive x-coord: towards stern
# The positive y-coord: towards leeward side
# The positive z-coord: above the water
# To shift the default CSYS, adjust the 'reference_level_for_moments' variable.
# Shifted CSYS = original + reference_level_for_moments
# As a results the moments will be calculated around the new origin.

# yaw_reference [m] - distance from the aft of the mast towards stern, at which the␣
↪yawing moment is calculated.
# sway_reference [m] - distance from the aft of the mast towards leeward side. 0 for␣
↪symmetric yachts ;)
```

(continues on next page)

(continued from previous page)

```
# heeling_reference [m] - distance from the water level,  at which the heeling moment␣
↪is calculated.
csys_args = {
    'reference_level_for_moments': np.array([0, 0, 0]),  # [yaw_reference, sway_
↪reference, heeling_reference]
}

# GEOMETRY OF THE KEEL
# to estimate heeling moment from keel, does not influence the optimizer.
# reminder: the z coord shall be negative (under the water)
keel_args={
    'center_of_lateral_resistance_upright': np.array([0, 0, -1.0]),  # [m] the␣
↪coordinates for a yacht standing in upright position
}
```

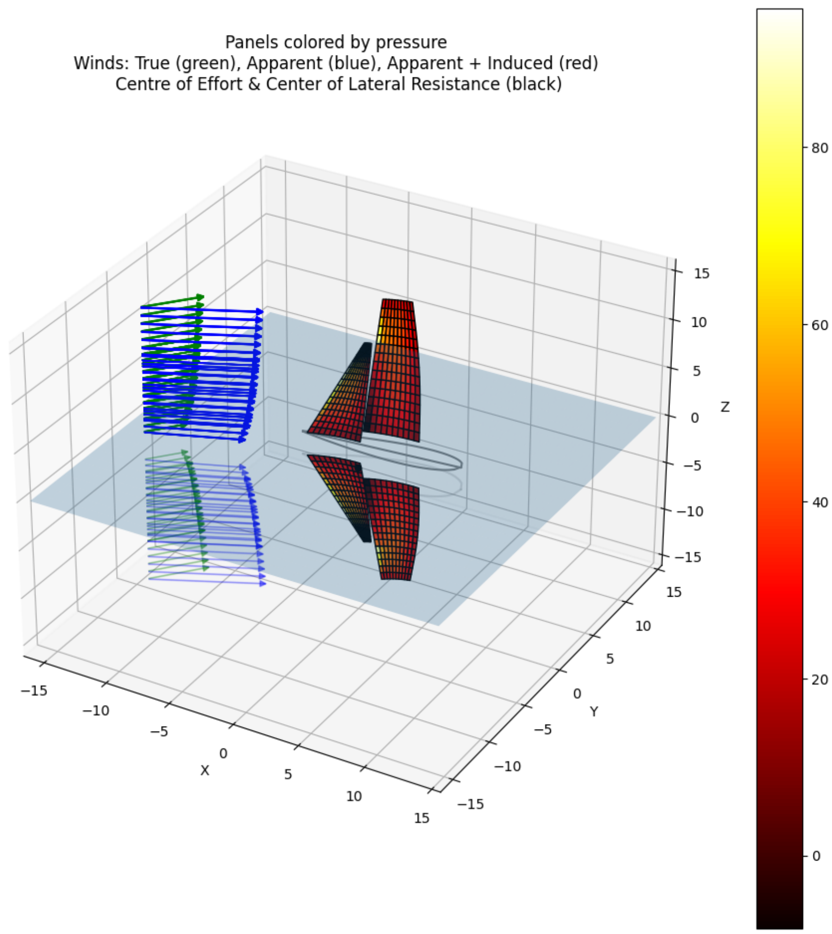
```
import shutil
from pySailingVLM.rotations.csys_transformations import CSYS_transformations
from pySailingVLM.yacht_geometry.hull_geometry import HullGeometry
from pySailingVLM.results.save_utils import save_results_to_file
from pySailingVLM.solver.panels_plotter import display_panels_xyz_and_winds
from pySailingVLM.results.inviscid_flow import InviscidFlowResults
from pySailingVLM.solver.vlm import Vlm
from pySailingVLM.runner.sail import Wind, Sail
from pySailingVLM.runner.container import Output, Rig, Conditions, Solver, MainSail,␣
↪JibSail, Csys, Keel
from pySailingVLM.solver.panels_plotter import plot_cp
```

```
out = Output(**output_args)
conditions = Conditions(**conditions_args)
solver = Solver(**solver_args)
main = MainSail(**main_sail_args)
jib = JibSail(**jib_sail_args)
csys = Csys(**csys_args)
keel = Keel(**keel_args)
rig = Rig(**rig_args)

csys_transformations = CSYS_transformations(
    conditions.heel_deg, conditions.leeway_deg,
    v_from_original_xyz_2_reference_csys_xyz=csys.reference_level_for_moments)


w = Wind(conditions)
s = Sail(solver, rig, main, jib, csys_transformations)
sail_set = s.sail_set
hull = HullGeometry(rig.sheer_above_waterline, rig.foretriangle_base, csys_
↪transformations, keel.center_of_lateral_resistance_upright)
myvlm = Vlm(sail_set.panels, solver.n_chordwise, solver.n_spanwise, conditions.rho, w.
↪profile, sail_set.trailing_edge_info, sail_set.leading_edge_info)

inviscid_flow_results = InviscidFlowResults(sail_set, csys_transformations, myvlm)
inviscid_flow_results.estimate_heeling_moment_from_keel(hull.center_of_lateral_
↪resistance)
```

Cell below displays computations and saves integrals to output file.

```
%matplotlib widget
print("Preparing visualization.")
display_panels_xyz_and_winds(myvlm, inviscid_flow_results, myvlm.inlet_conditions,␣
↪hull, show_plot=True) # add show_apparent_induced_wind=True for apparent + induced␣
↪wind
df_components, df_integrals, df_inlet_IC = save_results_to_file(myvlm, csys_
↪transformations, inviscid_flow_results, s.sail_set, out.name, out.file_name)
```

```
Preparing visualization.
```



Lets see our integrals :)

```
print(f"------------------------------------------------------------")
print(f"Notice:\n"
```

(continues on next page)

(continued from previous page)

```
  f"\tThe forces [N] and moments [Nm] are without profile drag.\n"
  f"\tThe the _COG_ CSYS is aligned in the direction of the yacht movement (course␣
↪over ground).\n"
  f"\tThe the _COW_ CSYS is aligned along the centerline of the yacht (course over␣
↪water).\n"
  f"\tNumber of panels (sail s.sail_set with mirror): {s.sail_set.panels.shape}")

df_integrals
```

```
------------------------------------------------------------
```

```
Notice:
        The forces [N] and moments [Nm] are without profile drag.
        The the _COG_ CSYS is aligned in the direction of the yacht movement␣
↪(course over ground).
        The the _COW_ CSYS is aligned along the centerline of the yacht (course␣
↪over water).
        Number of panels (sail s.sail_set with mirror): (600, 4, 3)
```

```
                               Quantity         Value
0                       F_jib_total_COG.x   -288.219785
1                       F_jib_total_COG.y    700.962481
2                       F_jib_total_COG.z    -49.892104
3                 F_main_sail_total_COG.x   -367.334322
4                 F_main_sail_total_COG.y   1098.532327
5                 F_main_sail_total_COG.z   -209.064962
6                     F_sails_total_COG.x   -655.554107
7                     F_sails_total_COG.y   1799.494808
8                     F_sails_total_COG.z   -258.957066
9                     F_sails_total_COW.x   -809.895833
10                    F_sails_total_COW.y   1735.511882
11                    F_sails_total_COW.z   -258.957066
12                      M_jib_total_COG.x  -3732.632148
13                      M_jib_total_COG.y  -1725.312796
14                      M_jib_total_COG.z   -816.618490
15                     M_keel_total_COG.x  -1816.952746
16                     M_keel_total_COG.y   -649.513937
17                     M_keel_total_COG.z     86.168258
18              M_keel_total_COW.x (heel)  -1753.429823
19             M_keel_total_COW.y (pitch)   -805.400206
20     M_keel_total_COW.z (yaw - JG sign)    -86.168258
21               M_keel_total_COW.z (yaw)     86.168258
22                M_main_sail_total_COG.x  -9857.210175
23                M_main_sail_total_COG.y  -3136.325045
24                M_main_sail_total_COG.z   2480.678079
25                    M_sails_total_COG.x -13589.842323
26                    M_sails_total_COG.y  -4861.637842
27                    M_sails_total_COG.z   1664.059590
28             M_sails_total_COW.x (heel) -13114.409213
29            M_sails_total_COW.y (pitch)  -6027.570644
30    M_sails_total_COW.z (yaw - JG sign)  -1664.059590
31              M_sails_total_COW.z (yaw)   1664.059590
32                          M_total_COG.x -15406.795070
33                          M_total_COG.y  -5511.151778
```

(continues on next page)

(continued from previous page)

```
34                        M_total_COG.z    1750.227847
35                 M_total_COW.x (heel) -14867.839036
36                M_total_COW.y (pitch)  -6832.970850
37        M_total_COW.z (yaw - JG sign)  -1750.227847
38                  M_total_COW.z (yaw)   1750.227847
```

Compute aerodynamic parameters:

```
sails_Cxyz = myvlm.get_Cxyz(w, 1.0)
print(f"Cxyz for {rig.sails_def}")
for idx, c in enumerate(sails_Cxyz):
    print(f"C[{idx}]: {c}")
```

```
Cxyz for jib_and_main
C[0]: [-2.08353396  5.06724107 -0.36066883]
C[1]: [-1.38511347  4.14225362 -0.78832463]
```

Make model plot in 2D colored by pressure coefficients:

```
plot_cp(sail_set.zero_mesh, myvlm.p_coeffs, out.name)
```

Thats all. Experiment and play with this code on your own.

5.5 Sweep cambered main sail

```
# varaibles.py for jupyter
import os
import numpy as np
import time

half_wing_span = 8
sweep_angle_deg = 5.
chord_length = 4
AoA_deg = 8.
mgirths =  np.array([0.00, 1./8, 1./4, 1./2, 3./4, 7./8, 1.00])
mchords = np.array([chord_length]* len(mgirths))


output_args = {
    'case_name': 'my_case_name',  # get name of the current file
    'case_dir': os.path.abspath(''), # get dir of the current file
    'name': os.path.join("results_example_jib_and_mainsail_vlm", time.strftime("%Y-%m-
↪%d_%Hh%Mm%Ss")),
    'file_name': 'my_fancy_results', # name of xlsx excel file
}

solver_args = {
    'n_spanwise':  5,  # No of control points (above the water) per sail,␣
↪recommended: 50
    'n_chordwise': 5, # No of control points (above the water) per sail, recommended:␣
↪50
```

(continues on next page)

(continued from previous page)

```
    'interpolation_type': "linear",  # either "spline" or "linear"
    'LLT_twist': "real_twist",  # defines how the Lifting Line discretize the sail␣
↪twist.
}

conditions_args = {
    'leeway_deg': 0.,     # [deg]
    'heel_deg': 0.,     # [deg]
    'SOG_yacht': 1.,   # [m/s] yacht speed - speed over ground (leeway is a separate␣
↪variable)
    'tws_ref': 1.0,     # [m/s] true wind speed
    'alpha_true_wind_deg': AoA_deg,   # [deg] true wind angle (with reference to␣
↪course over ground) => Course Wind Angle to the boat track = true wind angle to␣
↪centerline + Leeway
    'reference_water_level_for_wind_profile': -0.,  # [m] this is an attempt to␣
↪mimick the deck effect
    # by lowering the sheer_above_waterline
    # while keeping the wind profile as in original geometry
    # this shall be negative (H = sail_ctrl_point - water_level)
    'wind_exp_coeff': 0.,  # [-] coefficient to determine the exponential wind profile
    'wind_reference_measurment_height': 10.,  # [m] reference height for exponential␣
↪wind profile
    'rho': 1.,  # air density [kg/m3]
    'wind_profile': 'flat', # allowed: 'exponential' or 'flat' or 'logarithmic'
    'roughness': 0.05, # for logarithmic profile only
}


rig_args = {
    'main_sail_luff': half_wing_span / np.cos(np.deg2rad(sweep_angle_deg)),  # [m]
    'jib_luff': 10.0,  # [m]
    'foretriangle_height': 11.50,  # [m]
    'foretriangle_base': 3.90,  # [m]
    'sheer_above_waterline': 1.2,#[m]
    'boom_above_sheer': 1.3, # [m],
    'rake_deg': 90. + sweep_angle_deg,  # rake angle [deg]
    'mast_LOA': 0.,  # [m]
    'sails_def': 'main', # definition of sail set, possible: 'jib' or 'main' or 'jib_
↪and_main'
}


# INFO for camber:
# First digit describing maximum camber as percentage of the chord.
# Second digit describing the distance of maximum camber from the airfoil leading␣
↪edge in tenths of the chord.
main_sail_args = {
    'girths' : mgirths,
    'chords': mchords,
    'centerline_twist_deg': 0*mgirths,
    'camber': 10*np.array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]),
}


jgirths = np.array([0.00, 1./4, 1./2, 3./4, 1.00])
```

(continues on next page)

(continued from previous page)

```
jib_sail_args = {
    'centerline_twist_deg': 0*(10+5)  + 0*15. * jgirths,
    'girths': jgirths,
    'chords': 0* np.array([3.80, 2.98, 2.15, 1.33, 0.5]),
    'camber': 0*np.array([0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5]), # starting from␣
↪leading edge
}

# REFERENCE CSYS
# The origin of the default CSYS is located @ waterline level and aft face of the mast
# The positive x-coord: towards stern
# The positive y-coord: towards leeward side
# The positive z-coord: above the water
# To shift the default CSYS, adjust the 'reference_level_for_moments' variable.
# Shifted CSYS = original + reference_level_for_moments
# As a results the moments will be calculated around the new origin.

# yaw_reference [m] - distance from the aft of the mast towards stern, at which the␣
↪yawing moment is calculated.
# sway_reference [m] - distance from the aft of the mast towards leeward side. 0 for␣
↪symmetric yachts ;)
# heeling_reference [m] - distance from the water level,  at which the heeling moment␣
↪is calculated.
csys_args = {
    'reference_level_for_moments': np.array([0, 0, 0]),  # [yaw_reference, sway_
↪reference, heeling_reference]
}

# GEOMETRY OF THE KEEL
# to estimate heeling moment from keel, does not influence the optimizer.
# reminder: the z coord shall be negative (under the water)
keel_args={
    'center_of_lateral_resistance_upright': np.array([0, 0, -1.0]),  # [m] the␣
↪coordinates for a yacht standing in upright position
}
```

```
import shutil
from pySailingVLM.rotations.csys_transformations import CSYS_transformations
from pySailingVLM.yacht_geometry.hull_geometry import HullGeometry
from pySailingVLM.results.save_utils import save_results_to_file
from pySailingVLM.solver.panels_plotter import display_panels_xyz_and_winds
from pySailingVLM.results.inviscid_flow import InviscidFlowResults
from pySailingVLM.solver.vlm import Vlm
from pySailingVLM.runner.sail import Wind, Sail
from pySailingVLM.runner.container import Output, Rig, Conditions, Solver, MainSail,␣
↪JibSail, Csys, Keel

from pySailingVLM.solver.panels_plotter import plot_cp
```

```
import numpy as np
from pySailingVLM.solver.coefs import get_vlm_Cxyz

C_results = []
a_vlm_results = []
```

(continues on next page)

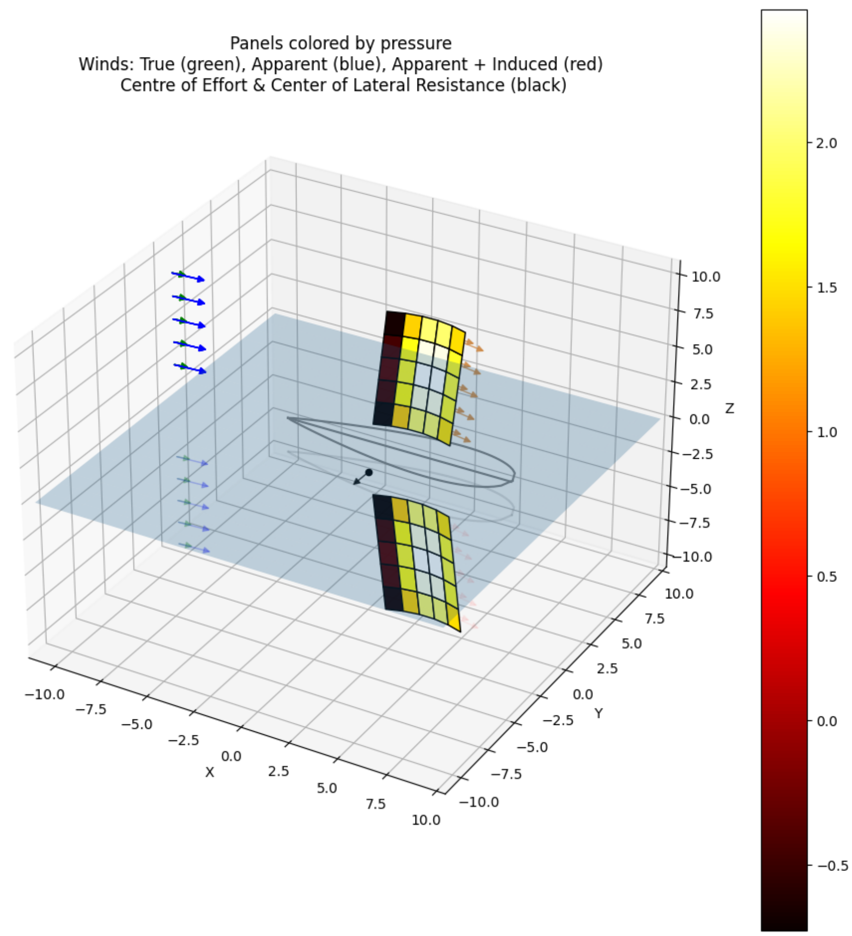
(continued from previous page)

```
out = Output(**output_args)
conditions = Conditions(**conditions_args)
solver = Solver(**solver_args)
main = MainSail(**main_sail_args)
jib = JibSail(**jib_sail_args)
csys = Csys(**csys_args)
keel = Keel(**keel_args)
rig = Rig(**rig_args)
csys_transformations = CSYS_transformations(conditions.heel_deg, conditions.leeway_
↪deg, v_from_original_xyz_2_reference_csys_xyz=csys.reference_level_for_moments)


w = Wind(conditions)
s = Sail(solver, rig, main, jib, csys_transformations)
sail_set = s.sail_set
myvlm = Vlm(sail_set.panels, solver.n_chordwise, solver.n_spanwise, conditions.rho, w.
↪profile, sail_set.trailing_edge_info, sail_set.leading_edge_info)

height = 1.0
sails_Cxyz = myvlm.get_Cxyz(w, height)

# enumerate through sails
# in this example we have only main
print(f"Cxyz for {rig.sails_def}")
for idx, c in enumerate(sails_Cxyz):
    print(f"C[{idx}]: {c}")

hull = HullGeometry(rig.sheer_above_waterline, rig.foretriangle_base, csys_
↪transformations, keel.center_of_lateral_resistance_upright)
inviscid_flow_results = InviscidFlowResults(sail_set, csys_transformations, myvlm)
inviscid_flow_results.estimate_heeling_moment_from_keel(hull.center_of_lateral_
↪resistance)
```

```
Cxyz for main
C[0]: [ 0.06028863  2.83686792 -0.00527457]
```

```
%matplotlib widget
print("Preparing visualization.")
display_panels_xyz_and_winds(myvlm, inviscid_flow_results, myvlm.inlet_conditions,␣
↪hull, show_plot=True, show_apparent_induced_wind=True) # add show_apparent_induced_
↪wind=True for apparent + induced wind
df_components, df_integrals, df_inlet_IC = save_results_to_file(myvlm, csys_
↪transformations, inviscid_flow_results, s.sail_set, out.name, out.file_name)
```

```
Preparing visualization.
```




```
print(f"-------------------------------------------------------------")
print(f"Notice:\n"
  f"\tThe forces [N] and moments [Nm] are without profile drag.\n"
  f"\tThe the _COG_ CSYS is aligned in the direction of the yacht movement (course␣
↪over ground).\n"
  f"\tThe the _COW_ CSYS is aligned along the centerline of the yacht (course over␣
↪water).\n"
  f"\tNumber of panels (sail s.sail_set with mirror): {s.sail_set.panels.shape}")

df_integrals
```

```
-------------------------------------------------------------
Notice:
        The forces [N] and moments [Nm] are without profile drag.
        The the _COG_ CSYS is aligned in the direction of the yacht movement␣
↪(course over ground).
```

(continues on next page)

(continued from previous page)

```
        The the _COW_ CSYS is aligned along the centerline of the yacht (course
↪over water).
        Number of panels (sail s.sail_set with mirror): (50, 4, 3)
```

```
                              Quantity       Value
0               F_main_sail_total_COG.x    0.988981
1               F_main_sail_total_COG.y   46.536280
2               F_main_sail_total_COG.z   -0.086525
3                   F_sails_total_COG.x    0.988981
4                   F_sails_total_COG.y   46.536280
5                   F_sails_total_COG.z   -0.086525
6                   F_sails_total_COW.x    0.988981
7                   F_sails_total_COW.y   46.536280
8                   F_sails_total_COW.z   -0.086525
9                    M_keel_total_COG.x  -46.536280
10                   M_keel_total_COG.y    0.988981
11                   M_keel_total_COG.z    0.000000
12            M_keel_total_COW.x (heel)  -46.536280
13           M_keel_total_COW.y (pitch)    0.988981
14   M_keel_total_COW.z (yaw - JG sign)   -0.000000
15             M_keel_total_COW.z (yaw)    0.000000
16              M_main_sail_total_COG.x -302.514742
17              M_main_sail_total_COG.y    6.902970
18              M_main_sail_total_COG.z  137.446498
19                  M_sails_total_COG.x -302.514742
20                  M_sails_total_COG.y    6.902970
21                  M_sails_total_COG.z  137.446498
22           M_sails_total_COW.x (heel) -302.514742
23          M_sails_total_COW.y (pitch)    6.902970
24  M_sails_total_COW.z (yaw - JG sign) -137.446498
25            M_sails_total_COW.z (yaw)  137.446498
26                        M_total_COG.x -349.051021
27                        M_total_COG.y    7.891951
28                        M_total_COG.z  137.446498
29                 M_total_COW.x (heel) -349.051021
30                M_total_COW.y (pitch)    7.891951
31        M_total_COW.z (yaw - JG sign) -137.446498
32                  M_total_COW.z (yaw)  137.446498
```

```
plot_cp(sail_set.zero_mesh, myvlm.p_coeffs, out.name)
```

CHAPTER

SIX

VALIDATION

6.1 Rectangular flat plate

Using pySailingVLm a rectangular flat plate was modeled with a span of 10 units and a constant chord of 1 unit. The geomery of sail is discretized into 16 spanwise and 8 chordwise panels. The free-stream strength is 1 unit and the angle of attack is 10°. Then the results was compared with [AH13] and analitically derived results in the table below.

Table 6.1: Results from pySailingVLM code of the rectangular flat plate.

	Analitical	[AH13]	pySailingVLM
$C_{L,\alpha}$	4.896	4.786	4.846

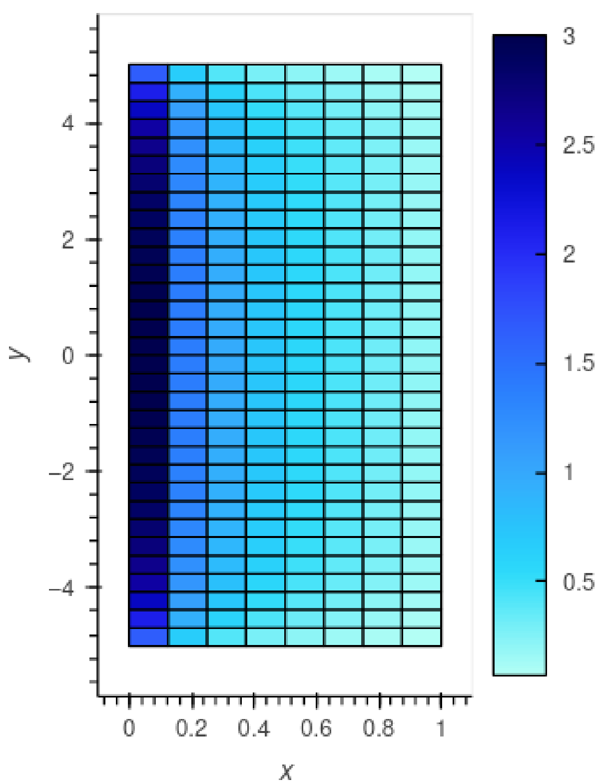


Fig. 6.1: Pressure coefficients for flat plate. Figure generated by pySailingVLM.

6.1.1 results

pySailingVLM gives very similar results in comparison to [AH13] and analitically derived results.

6.2 Sweep wing

Swept wing by Bertin [JJB09] is a flat plate with a 45° leading edge sweeping angle. The span of wing is 1 unit and a chord of 0.2 units. Model was discretized into 4 spanwise panels and 1 chordwise panel. The free stream is aligned with the x-axis and has a magnitude of 1 units.

Table 6.2: Comparison between swept wing by Bertin and pySailingVLM results.

	Bertin	pySailingVLM
$C_{L,\alpha}$	3.443	3.434

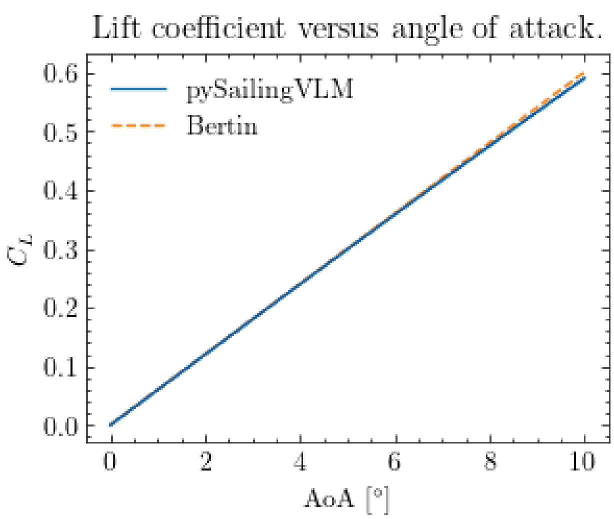


Fig. 6.2: Lift coefficient versus angle of attack. Figure generated by pySailingVLM.

TODO Dodatkowy wykres za Katzem: https://www.researchgate.net/publication/245355607_Calculation_of_the_Aerodynamic_Forces_or
Fig4

6.3 results

Results obtained by pySailingVLM are close to teoretical results taken from [JJB09].

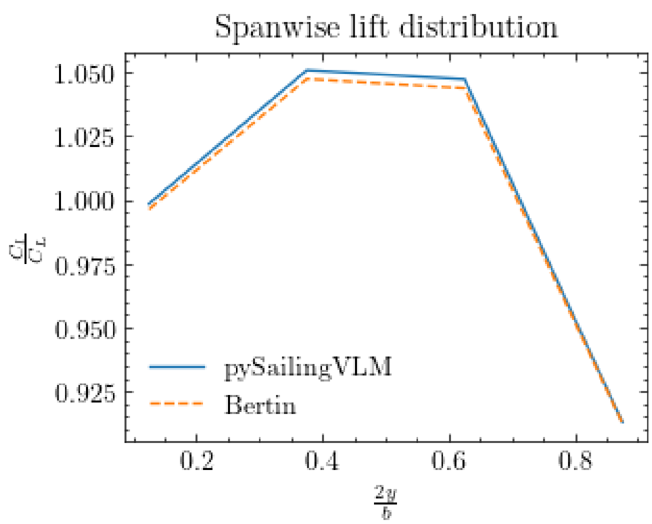


Fig. 6.3: Comparison of spanwise lift distributionfor the Bertin wing. Figure generated by pySailingVLM.

CHAPTER SEVEN

CONCLUSIONS

7.1 Future outlooks

TODO

CHAPTER

EIGHT

BIBLIOGRAPHY

BIBLIOGRAPHY

[AH13] T.Larsson A. Helmstad. *An Aeroelastic Implementation for Yacht Sails and Rigs, Degree project in Naval Architecture*. KTH Engeneering Sciences, 2013.

[JK01] A. Plotkin J. Katz. *Low-Speed Aerodynamics, Second Edition*. Cambridge University Press, 2001.

[JJB09] Russell M. Cummings John J. Bertin. *Aerodynamics for engineers, Fifth Edition*. Pearson Education International, 2009.

[NJS+21] Sheharyar Nasir, M Tariq Javaid, M. Usman Shahid, Asad Raza, Waseeq Siddiqui, and Shuaib Salamat. Applicability of vortex lattice method and its comparison with high fidelity tools. *Pakistan Journal of Engineering and Technology, PakJET*, 04(01):207– 211, 2021. URL: https://www.hpej.net/journals/pakjet/article/view/754/564, doi:DOI: 10.51846/vol4iss1pp207-211.

[PS00] W. F. Phillips and D. O. Snyder. Modern adaptation of prandtl's classic lifting-line theory. *JOURNAL OF AIRCRAFT*, 37(4):662–670, July - August 2000. URL: http://arc.aiaa.org, doi:DOI: 10.2514/2.2649.

[SHR+19] A. Septiyana, K. Hidayat, A. Rizaldi, M. L. Ramadiansyah, R. A. Ramadhan, P. A. P. Suseno, E. B. Jayanti, N. Atmasari, and A. Rasyadi. Analysis of aerodynamic characteristics using the vortex lattice method on twin tail boom unmanned aircraft. *7TH INTERNATIONAL SEMINAR ON AEROSPACE SCIENCE AND TECHNOLOGY - ISAST 2019*, 2226(April):536, 24-25 September 2019. URL: https://pubs.aip.org/aip/acp/article/2226/1/020003/814546/Analysis-of-aerodynamic-characteristics-using-the, doi:10.1063/5.0002337.

[Tec05] Prof. A.H. Techet. Potential flow theory. In *Lecture: 2.016 Hydrodynamics*. Massachusetts Institute of Technology, 2005. MIT. URL: https://web.mit.edu/2.016/www/handouts/2005Reading4.pdf.