A Lumped Parameter Equilibrium Model of a Submerged Body with Mooring Lines

Geoffrey DuBuque

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

University of Washington

2011

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a master's thesis by

Geoffrey DuBuque

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____
Brian Fabien

_____
I-Yeu Shen

_____
Brian Polagye

_____
Santosh Devasia

Date: _____

In presenting this thesis in partial fulfillment of the requirements for a master's degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Any other reproduction for any purposes or by any means shall not be allowed without my written permission.

Signature _____

Date _____

University of Washington

Abstract

A Lumped Parameter Equilibrium Model of a Submerged Body with Mooring Lines

Geoffrey DuBuque

Chair of the Supervisory Committee:
Professor Brian Fabien
Mechanical Engineering

A lumped parameter model is developed to determine the equilibrium of a submerged body and mooring system under different stream flows and load conditions. An underwater turbine system is used as an example test problem. Determining equilibrium is important for systems such as underwater turbines because its position and orientation may affect its performance. The body is modeled as a rigid body and the mooring lines as multiple elastic line segments. All forces in the system including weight, buoyancy, drag, elastic stiffness and applied forces are modeled as conservative forces to find their potential energies. A stable equilibrium is solved for by minimizing the potential energy of the system. Matlab is used to solve the system by using optimization software and all code is included in the Appendix that a user can modify to model and solve their systems. Benchmark tests were run to validate the model and Matlab implementation against analytical solutions for an elastic catenary and a 2D rigid body rotation test. A series of test problems were run to observe trends when finding equilibrium of a system including varying the fluid velocity angle and variation in the location of the body's center of mass (COM) and center of buoyancy (COB). Important observations include body rotation towards the oncoming fluid flow for the example system considered and body rotation if the COM or COB is not centered relative to a symmetric mooring system.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my advisor, Brian Fabien, for giving me the opportunity and support to work on this project. His guidance, teaching and support of my work on this project have shown me what I can accomplish. I would also like to thank the rest of my committee for their useful edits and comments about this thesis to help make it a stronger work. I want to thank the Northwest National Marine Renewable Energy Center (NNMREC) for their pursuit and support of renewable energy projects that pave the way in this very important field. Lastly, I want to give a big thank you to my family for their endless support during my continued education which has allowed me to focus on my education and work.

1 Introduction

1.1 Motivation and Objectives

Renewable energy is a growing industry and research field. Renewable energy technology, such as underwater in-stream turbines, uses a turbine and generator to convert the power of streams or tidal flows to electricity. Most turbines deployed to date use a fixed structure to secure the turbine to a seabed or riverbed, though another design is to make the turbine structure buoyant and secure it to the seabed with mooring lines. These moored systems require a complicated dynamic and static analysis of the system. A static analysis is important to find equilibrium of the system that may be used to determine such things as the role orientation plays in the performance of the turbine.

For deep water applications, submerged bodies like Ocean Renewable Power Company's (ORPC) OCGen™ system design are proposed to be secured to the sea floor with multiple mooring lines (see Figure 1.1) (1). Finding the equilibrium will show the position and orientation of the system subjected to different stream flows and load conditions. The orientation of the system is important because the efficiency or power performance of a turbine may greatly depend on its orientation relative to the stream velocity. One application is to use the equilibrium position to find an optimum configuration of the system's mooring lines to achieve a desired position or orientation of the body to maximize performance.

The objectives of this work are as follows:

- Develop a lumped parameter model to describe a submerged rigid body with multiple mooring lines and derive the potential energy equations used to determine stable equilibrium positions. Equilibrium is found directly and the dynamics of the system is not considered in the scope of this work.
- Develop Matlab code to solve the equilibrium equations and act as a user interface to the modeling and analysis code.

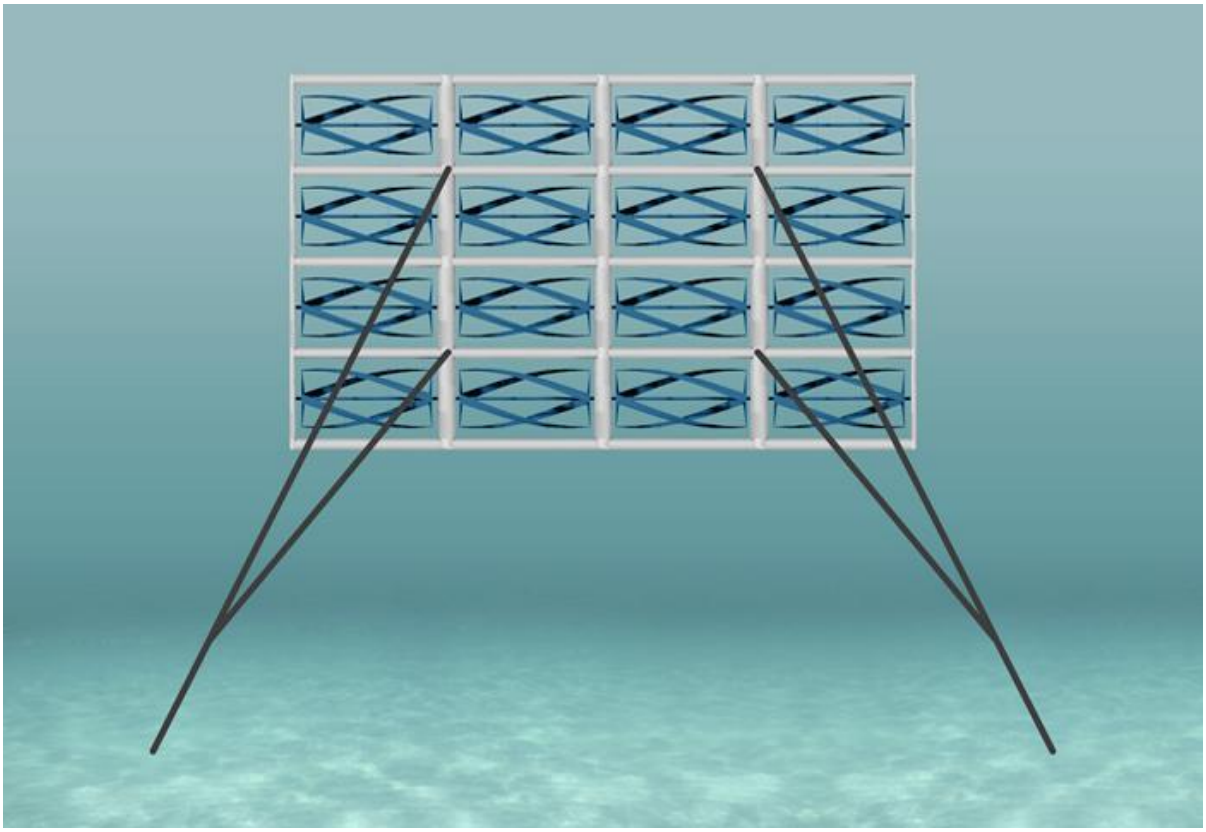


Figure 1.1 Concept image of ORPC's OCGen™ system, submerged body and mooring line system example (1)

1.2 Approach and Methodology

A stable equilibrium of the system will be found by minimizing the potential energy of the system. Lagrange's theorem of stability states: If the potential energy has an isolated minimum in the equilibrium position, then the equilibrium is stable (2). The equilibriums will be solved for numerically by using optimization tools provided in the Matlab software package.

The system will be modeled as follows:

1. Six-degree of freedom rigid body: three translational and three rotational degrees of freedom.
2. Mooring lines: The mooring lines will be modeled as unidirectional elastic springs (the mooring lines are allowed to be slack). Each mooring line will be divided into multiple segments.
3. Lumped parameters: The body and line segments will be described with lumped parameters including weight, buoyancy, drag, applied forces to the body and stiffness for the line segments.

The system will be analyzed as follows:

1. The system will be modeled with conservative forces to develop the potential energy equations to be solved for stable equilibrium.
2. Matlab script will be written to calculate the equations and use optimization software to solve for stable equilibrium positions. Another important aspect of the software development is to provide a simple user interface to the modeling and analysis code.
3. A series of benchmark problems will be used to validate the modeling approach and Matlab code implementation against analytical solutions for an elastic catenary and a 2D rigid body rotation test.
4. A series of test problems will be run to observe any important trends when finding equilibrium of a system including varying the fluid velocity angle and variation in the location of the body's center of mass and center of buoyancy.

1.3 Literature Review

Moored systems of cables and attached bodies provide many interesting and challenging modeling problems. Irvine (3) provides analysis and examples of many different types of cable structures. A lot of research in this area focuses on the dynamics of the cables and attached bodies. Some analysis use differential equations to describe

the mooring system. The work of Leonard et al (4) models an uncoupled three-dimensional analysis of a floating buoy and its mooring, with the buoy and mooring problems solved separately which are coupled together by constraining the solutions at the attachment point of the mooring and buoy. The work of Tjavaras et al (5) models the mechanics of highly extensible cables and can model the dynamics of a near-surface buoy. This work includes modeling bending stiffness in the lines to ensure the problem is solvable with small tension forces.

Other methods to avoid modeling the mooring lines with differential equations or problems with low tension include modeling the lines as lumped springs and masses. The work by Huang (6) uses a lump-mass-and-spring model to describe the three-dimensional dynamics of mooring lines. The work of Raman-Nair and Baddour (7) uses a lumped mass-spring model to describe the three-dimensional dynamics of a buoy with multiple mooring and is discussed further below.

There are works that model systems of submerged bodies and multiple mooring lines but are different than the modeling approach and applications considered in this work in a few key areas. Major differences include how equilibrium is solved for, how constraints such as line slack and touchdown are handled and how the attached body is modeled. The current implementation of this work is suited to model the equilibrium of systems of a single rigid body attached to multiple mooring lines at different locations, and the differences or limitations of other work to model such systems is discussed below.

The work of Raman-Nair and Baddour (7) uses Kane and Levinson's (8) dynamic formulism as a base to model the three-dimensional dynamics of a subsurface buoy and multiple mooring lines. Their model uses a lumped mass-spring model for the mooring lines. To model the mooring line's interaction with the ground a normal reaction force from the seabed is applied to the lines at touchdown. This reaction force is directly proportional to the penetration into the seabed, acting like a linear spring. Touchdown is modeled differently in this work than theirs. Instead of modeling additional forces to

the system, constraints are applied directly to the displacement variables of the system. To model a flat seabed the vertical component of every mooring line node is constrained to be greater than or equal to the elevation of the seabed, so the mooring lines must either be above or lie flat on the seabed.

To model slack in the mooring lines (to only support tension and not compression), Raman-Nair and Baddour define the elongation of a segment in a way such that if the length of the segment becomes less than its unstretched length, the elongation equals zero and the resulting tension force due to stiffness is zero (7). The slack constraint in our work constrains the coordinates of the mooring line nodes such that the length of each line segment must be greater than or equal to its unstretched length (see section 3.4).

To find equilibrium of a system, Raman-Nair and Baddour solve the system for a long simulation time so the system approaches its equilibrium position (7). Our work finds a stable equilibrium of the system directly without a dynamic analysis (see sections 1.2 and 3.1).

A software program has been developed at the Woods Hole Oceanographic Institution (WHOI) called WHOI Cable (9), based on the works of Gobat and Grosenbaugh, et al (10), (11). This program solves two and three-dimensional mooring systems for either dynamic or equilibrium (or steady-state) solutions, including towing and drifting problems. WHOI Cable allows for various connectors, branches, buoys and anchors to be added to a mooring line. The program uses governing differential equations developed in the works by Gobat (11) and Tjavaras (12) which are solved for numerically by the program (10). Touchdown of the cables on the seabed is modeled as a linear elastic foundation as well, and bending stiffness is included in the lines to avoid singularities with slack tension.

Other differences between WHOI Cable and this work include how bodies (or buoys) are attached and modeled in the system. WHOI Cable attaches cables to a buoy at a

single point, while this work allows any number of mooring lines to be attached at different locations to the same body. WHOI Cable does not model the rigid body dynamics of an attached body and the coefficients of drag for a buoy are defined in normal and tangential directions, while the coefficients of drag in this work are defined along the three body-fixed axis and rigid body rotations of the body are modeled.

2 Modeling of the Body and Mooring Line Kinematics

2.1 Rigid Body Kinematics

The coordinate systems are chosen in a way to easily keep track of all points of interest and forces in the system. An inertial frame is used to conveniently describe the whole system relative to a fixed location such as the seabed. A coordinate system that moves and rotates with the body (body-fixed frame) is used to easily describe points fixed to the body. The body is modeled as a rigid body, meaning all points fixed to the body do not move relative to each other. This provides a convenient way to describe body-fixed points in the inertial frame even after complicated movements and rotations of the body.

A fixed rectangular coordinate system, denoted by $\boldsymbol{X-Y-Z}$, with origin $\boldsymbol{O}$ on the seabed is used as an inertial reference frame. A body-fixed rectangular coordinate system with origin $\boldsymbol{o'}$ located at the body's center of mass (COM) is denoted $\boldsymbol{x'-y'-z'}$. The coordinate systems are displayed in Figure 2.1.

The rigid body is modeled with six degrees of freedom (DOF) that includes three translational DOF and three rotational DOF:

- Position of the center of mass in the inertial frame $\bar{\boldsymbol{c}}_{\boldsymbol{o'}} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix}$
- Rotation angles of the body $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$

The three rotation angles, $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$, are used to completely described the body's rotation in 3-dimensional space. The body's final rotation is found by three successive rotations about an axis in the body-fixed frame. These angles are known as Euler angles.

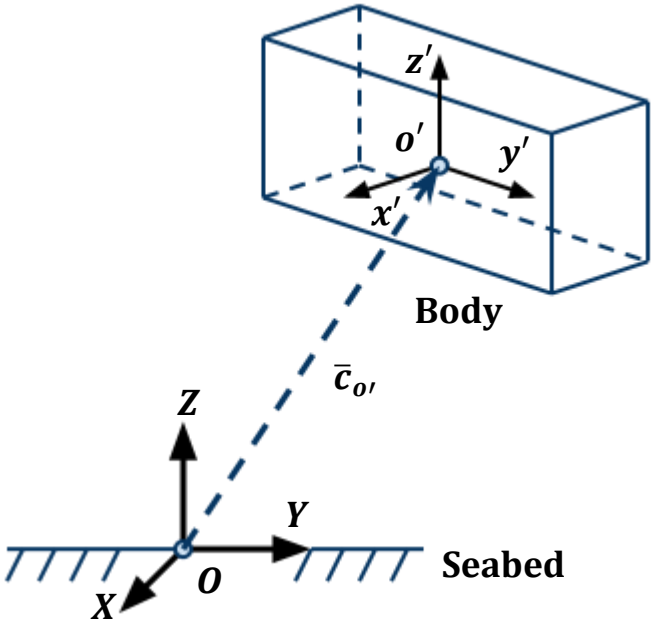


Figure 2.1 Inertial and body-fixed coordinate systems

Intermediate frames are used to follow the rotations and describe how a rotated reference frame is related to the initial reference frame. The coordinates of a point $\bar{\boldsymbol{r}}_{\boldsymbol{b}} = \left[\boldsymbol{r}_{\boldsymbol{bx}}\, \boldsymbol{r}_{\boldsymbol{by}}\, \boldsymbol{r}_{\boldsymbol{bz}}\right]^{\boldsymbol{T}}$ in the rotated frame $\boldsymbol{x_b} - \boldsymbol{y_b} - \boldsymbol{z_b}$ can be written in the coordinates $\bar{\boldsymbol{r}}_{\boldsymbol{a}} = \left[\boldsymbol{r}_{\boldsymbol{ax}}\, \boldsymbol{r}_{\boldsymbol{ay}}\, \boldsymbol{r}_{\boldsymbol{az}}\right]^{\boldsymbol{T}}$ relative to the initial frame $\boldsymbol{x_a} - \boldsymbol{y_a} - \boldsymbol{z_a}$. $\bar{\boldsymbol{r}}_{\boldsymbol{a}}$ can be found from the transformation

$$\bar{\boldsymbol{r}}_{\boldsymbol{a}} = \boldsymbol{R}\bar{\boldsymbol{r}}_{\boldsymbol{b}} \tag{2.1}$$

where $\boldsymbol{R}$ is a rotation transformation matrix.

For rotation about an axis, $\boldsymbol{R}$ has the following forms:

- Rotation of $\boldsymbol{\alpha}$ about the x axis, $\boldsymbol{R_x} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{c_\alpha} & \boldsymbol{-s_\alpha} \\ \mathbf{0} & \boldsymbol{s_\alpha} & \boldsymbol{c_\alpha} \end{bmatrix}$
- Rotation of $\boldsymbol{\beta}$ about the y axis, $\boldsymbol{R_y} = \begin{bmatrix} \boldsymbol{c_\beta} & \mathbf{0} & \boldsymbol{s_\beta} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \boldsymbol{-s_\beta} & \mathbf{0} & \boldsymbol{c_\beta} \end{bmatrix}$
- Rotation of $\boldsymbol{\gamma}$ about the z axis, $\boldsymbol{R_z} = \begin{bmatrix} \boldsymbol{c_\gamma} & \boldsymbol{-s_\gamma} & \mathbf{0} \\ \boldsymbol{s_\gamma} & \boldsymbol{c_\gamma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$

where $\boldsymbol{c_\theta} = \mathbf{cos}\,\boldsymbol{\theta}, \boldsymbol{s_\theta} = \mathbf{sin}\,\boldsymbol{\theta}$.

To describe the rotation of a body in 3-dimensions we will use three successive rotations about the x, y then z-axis of the intermediate frames. Starting with the origin of the body-fixed frame $\boldsymbol{x_a - y_a - z_a}$ coincident and aligned with the inertial reference frame $\boldsymbol{X - Y - Z}$ (Figure 2.2a), rotate $\boldsymbol{\alpha}$ about the $\boldsymbol{x_a}$ axis (Figure 2.2b). The intermediate frame becomes $\boldsymbol{x_b - y_b - z_b}$. Then rotate $\boldsymbol{\beta}$ about the $\boldsymbol{y_b}$ axis and the intermediate frame becomes $\boldsymbol{x_c - y_c - z_c}$ (Figure 2.2c). Finally, rotate $\boldsymbol{\gamma}$ about the $\boldsymbol{z_c}$ axis and the final rotation of the body is aligned with the $\boldsymbol{x_1 - y_1 - z_1}$ frame (Figure 2.2d).

A vector $\bar{\boldsymbol{r}}_{\boldsymbol{a}}$ in the inertial frame can be found from a vector $\bar{\boldsymbol{r}}_{\mathbf{1}}$ in the body-fixed frame with the transformation

$$\bar{\boldsymbol{r}}_{\boldsymbol{a}} = \boldsymbol{T}\bar{\boldsymbol{r}}_{\mathbf{1}} \tag{2.2}$$

$$\boldsymbol{T} = \boldsymbol{R_x R_y R_z} = \begin{bmatrix} \boldsymbol{c_\beta c_\gamma} & \boldsymbol{-c_\beta s_\gamma} & \boldsymbol{s_\beta} \\ \boldsymbol{s_\alpha s_\beta c_\gamma + c_\alpha s_\gamma} & \boldsymbol{-s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma} & \boldsymbol{-s_\alpha c_\beta} \\ \boldsymbol{-c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma} & \boldsymbol{c_\alpha s_\beta s_\gamma + s_\alpha c_\gamma} & \boldsymbol{c_\alpha c_\beta} \end{bmatrix} \tag{2.3}$$

The transformation matrix $\boldsymbol{T}$ uses the 1-2-3 (or XYZ) Euler angle convention.

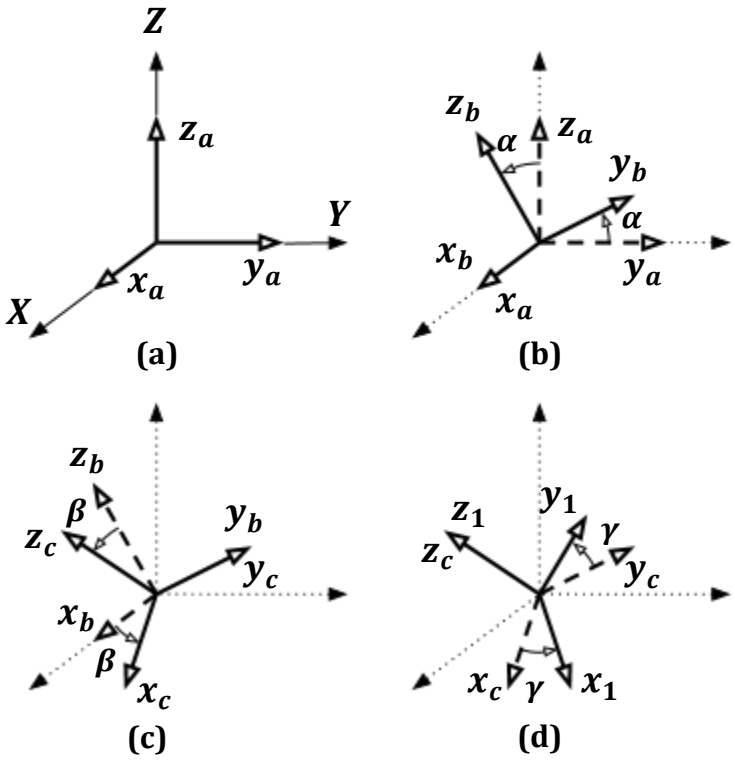


Figure 2.2 Euler rotations of intermediate frames

Coordinates in the body-fixed frame are written with a prime and coordinates in the inertial frame (ground, or fixed frame) are written without a prime.

An arbitrary point on the body with body-fixed coordinates $\overline{\boldsymbol{p}}' = \left[\boldsymbol{p}_{x'}\, \boldsymbol{p}_{y'}\, \boldsymbol{p}_{z'}\right]^{\boldsymbol{T}}$ then has the following coordinates with respect to ground $\overline{\boldsymbol{p}} = [\boldsymbol{p}_{\boldsymbol{X}}\, \boldsymbol{p}_{\boldsymbol{Y}}\, \boldsymbol{p}_{\boldsymbol{Z}}]^{\boldsymbol{T}}$:

$$\overline{\boldsymbol{p}} = \overline{\boldsymbol{c}}_{o'} + \boldsymbol{T}\overline{\boldsymbol{p}}' \tag{2.4}$$

A vector in a rotated frame can be found from the vector in the fixed frame by applying the inverse of the rotation matrix. The inverse of a rotation matrix is equal to its transpose and the inverse of the 1-2-3 transformation matrix is

$$\boldsymbol{T}^{-1} = \boldsymbol{T}^{\boldsymbol{T}} = \begin{bmatrix} \boldsymbol{c}_\beta \boldsymbol{c}_\gamma & \boldsymbol{s}_\alpha \boldsymbol{s}_\beta \boldsymbol{c}_\gamma + \boldsymbol{c}_\alpha \boldsymbol{s}_\gamma & -\boldsymbol{c}_\alpha \boldsymbol{s}_\beta \boldsymbol{c}_\gamma + \boldsymbol{s}_\alpha \boldsymbol{s}_\gamma \\ -\boldsymbol{c}_\beta \boldsymbol{s}_\gamma & -\boldsymbol{s}_\alpha \boldsymbol{s}_\beta \boldsymbol{s}_\gamma + \boldsymbol{c}_\alpha \boldsymbol{c}_\gamma & \boldsymbol{c}_\alpha \boldsymbol{s}_\beta \boldsymbol{s}_\gamma + \boldsymbol{s}_\alpha \boldsymbol{c}_\gamma \\ \boldsymbol{s}_\beta & -\boldsymbol{s}_\alpha \boldsymbol{c}_\beta & \boldsymbol{c}_\alpha \boldsymbol{c}_\beta \end{bmatrix} \tag{2.5}$$

A vector $\bar{r}_1$ in the body-fixed frame can be found from a vector $\bar{r}_a$ in the inertial frame with the transformation

$$\bar{r}_1 = T^T \bar{r}_a \tag{2.6}$$

2.2 Mooring Line Kinematics

The mooring lines are modeled as unidirectional elastic springs with lumped parameters including stiffness, weight, buoyancy and drag forces. The system will be modeled with i mooring lines and each line divided into N_i segments using nodes $\bar{p}_{i,j}$ with $j = 0, 1, 2, \ldots, N_i$ (see Figure 2.3). The mooring lines are divided into multiple segments to accurately describe how the line bends or becomes slack under load. The coordinates of a node in the inertial frame is $\bar{p}_{i,j} = \left[p_{i,j,X}\ p_{i,j,Y}\ p_{i,j,Z}\right]^T$.

If a node is attached to the body its coordinates are determined by the body's position and rotation. Let $\bar{p}'_{i,0}$ denote the coordinates of the attachment point for the i_{th} mooring line on the body in the body-fixed frame. The coordinates of node 0 in the body-fixed frame are $\bar{p}'_{i,0} = \left[p_{i,0,x'}\ p_{i,0,y'}\ p_{i,0,z'}\right]^T$. The coordinates of node 0 in the inertial frame are

$$\bar{p}_{i,0} = \bar{c}_{o'} + T\bar{p}'_{i,0} \tag{2.7}$$

Note that there are three degrees of freedom per node. However, if a node is attached to the body then the degrees of freedom associated with that node are determined by the equation above and therefore shares the same degrees of freedom as the body.

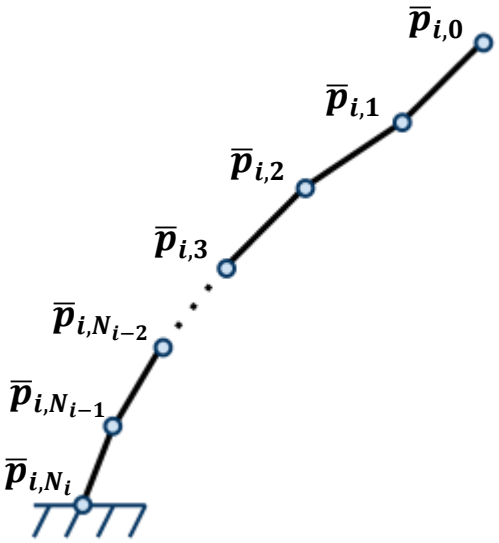


Figure 2.3 Mooring line segments and nodes

Figure 2.3 shows an example of the mooring line segments and node convention. The $\overline{p}_{i,0}$ node is attached to the body and the $\overline{p}_{i,N_i}$ node is attached to the seabed.

Each mooring line will have F_i degrees of freedom, where

$$F_i = 3(N_i + 1) - 3 - 3 = 3N_i - 3 \tag{2.8}$$

Each mooring lines has $(N_i + 1)$ nodes. Three degrees of freedom are subtracted for the first node fixed to the body and another three degrees of freedom are subtracted for the last node fixed to the seabed.

3 Modeling of the Loads and Potential Energies

3.1 Approach

Lumped parameters and forces are used to easily describe and keep track of the forces on the system. All forces are modeled to be conservative (the work done by the force is path-independent) in order to find their potential energy used to solve for equilibrium. The potential energy $\boldsymbol{V}$ of a conservative force $\boldsymbol{F}$ is defined as

$$\boldsymbol{V}(\boldsymbol{q}) = -\int \boldsymbol{F}(\boldsymbol{q}) \cdot \boldsymbol{dq} \tag{3.1}$$

where $\boldsymbol{q}$ is a displacement variable (2). The total potential energy of the system is then found by summing the potential energy of every force in the system. The total potential energy equation is minimized to find a stable equilibrium of the system.

3.2 Mooring Line Buoyancy and Weight

In this model the mass of the mooring line is distributed uniformly at the center of each segment (see Figure 3.1). For example, the $\boldsymbol{j_{th}}$ segment of the $\boldsymbol{i_{th}}$ mooring line will have mass $\boldsymbol{m_{i,j}, j = 1, 2, \dots, N_i}$. The $\boldsymbol{j_{th}}$ segment is defined by nodes $\overline{\boldsymbol{p}}_{\boldsymbol{i,j-1}}$ and $\overline{\boldsymbol{p}}_{\boldsymbol{i,j}}$. The mass of the segment is lumped at its center $\overline{\boldsymbol{q}}_{\boldsymbol{i,j}}$. The coordinates of the center are determined by consecutive nodes as

$$\overline{\boldsymbol{q}}_{\boldsymbol{i,j}} = \frac{\boldsymbol{1}}{\boldsymbol{2}}\left(\overline{\boldsymbol{p}}_{\boldsymbol{i,j-1}} + \overline{\boldsymbol{p}}_{\boldsymbol{i,j}}\right) \tag{3.2}$$

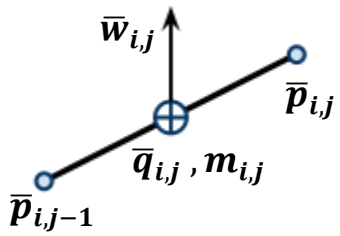


Figure 3.1 Segment center of mass and net buoyant and gravity force

Let L_i denote the free (unstretched) length of the i_{th} mooring line. Let ρ_i denote the linear density (mass per unit length) of the i_{th} mooring line. The mass of the j_{th} segment is $m_{i,j} = \rho_i L_{i,j}$, where $L_{i,j}$ is the unstretched length of the j_{th} segment. Note that $L_{i,j}$ is not the instantaneous length of the segment, $\left(L_{i,j} \neq \left|\bar{p}_{i,j} - \bar{p}_{i,j-1}\right|\right)$, because the length can vary with time if the mooring line stretches.

The buoyant and gravity forces only act in the vertical direction, which is along the **Z** axis in the fixed frame. The net force of the buoyant and gravity forces acting on the j_{th} segment of the i_{th} mooring line with respect to the fixed frame is

$$\bar{w}_{i,j} = \left[0\ 0\ w_{i,j,Z}\right]^T \tag{3.3}$$

The **Z** component of the net force is

$$w_{i,j,Z} = -m_{i,j}g + \rho_f V_{i,j} g \tag{3.4}$$

where g is acceleration due to gravity, ρ_f is the fluid density and $V_{i,j}$ is the volume of the j_{th} segment of the i_{th} mooring line. This net force acts at the segment's center of mass (see Figure 3.1).

The potential energy of the net buoyant and gravity force on each segment is

$$V\left(\bar{w}_{i,j}\right) = -w_{i,j,Z} q_{i,j,Z} \tag{3.5}$$

3.3 Body Buoyancy and Weight

The force due to gravity acts at the center of mass (COM) of the body which is the origin **o'** in the body-fixed frame. Recall that the position of **o'** in the inertial frame is $\bar{c}_{o'} = [x\ y\ z]^T$. The buoyant force acts the body's center of buoyancy (COB). The location of the COB in the body-fixed frame is $\bar{c}'_b = [x'_b\ y'_b\ z'_b]^T$ (see Figure 3.2). The COB in the inertial frame is

$$\bar{c}_b = \bar{c}_{o'} + T\bar{c}'_b \tag{3.6}$$

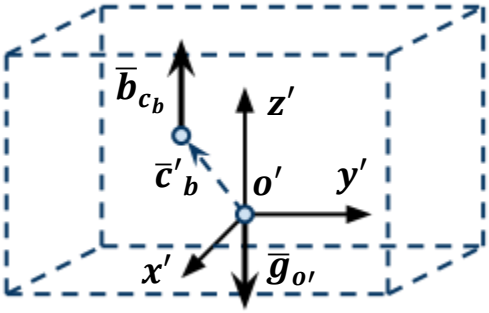


Figure 3.2 Gravity and buoyant forces on the body

The force due to gravity in the inertial frame is $\bar{g}_{o'} = [0\ 0\ -Mg]^T$, where M is the mass of the body.

The potential energy of the weight of the body is

$$V(\bar{g}_{o'}) = Mgz \tag{3.7}$$

where z is the **Z** component of $\bar{c}_{o'}$.

The buoyancy force on the body is $\bar{b}_{c_b} = \left[0\ 0\ \rho_f V_{body} g\right]^T$, where V_{body} is the volume of the body.

The potential energy of the buoyancy force on the body is

$$V(\bar{b}_{c_b}) = -\rho_f V_{body} g c_{b,Z} \tag{3.8}$$

where $c_{b,Z}$ is the **Z** component of $\bar{c}_b$.

3.4 Slack Mooring Model

Each segment of the mooring line is modeled as a spring that applies a force only if the segment elongates relative to its unstretched length (see Figure 3.3). The stiffness of the j_{th} segment of the i_{th} mooring line is

$$K_{i,j} = \frac{E_{i,j} A_{i,j}}{L_{i,j}} \tag{3.9}$$

$E_{i,j}$ is the modulus of elasticity, $A_{i,j}$ is the effective cross-sectional area and $L_{i,j}$ is the unstretched length of the segment.

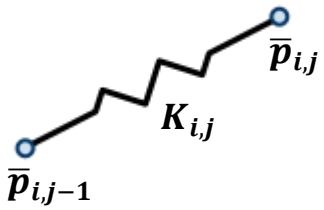


Figure 3.3 Segment with stiffness

A segment is defined as a line between two adjacent nodes, written as $\bar{l}_{i,j} = \bar{p}_{i,j} - \bar{p}_{i,j-1}$. The length of the segment is the magnitude of this line:

$$l_{i,j} = |\bar{l}_{i,j}| = \sqrt{l_{i,j,X}^2 + l_{i,j,Y}^2 + l_{i,j,Z}^2} \tag{3.10}$$

where $l_{i,j,X} = p_{i,j,X} - p_{i,j-1,X}$, etc.

The lines are modeled to only support tension forces when the segment is stretched and not compressive forces or allow the line to be compressed, instead the line becomes slack. To model slack and only tension forces the change in length of the segment is constrained to be greater than or equal to zero:

$$\Delta \boldsymbol{l}_{i,j} = \boldsymbol{l}_{i,j} - \boldsymbol{L}_{i,j} \geq \boldsymbol{0} \tag{3.11}$$

When a segment is elongated there are reaction forces at each node that point inwards along the segment (see Figure 3.4).

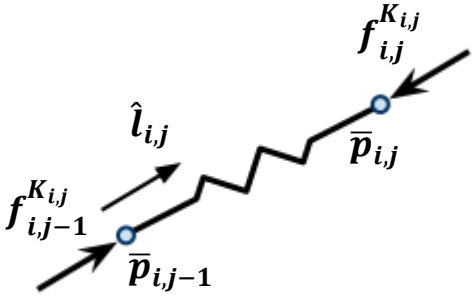


Figure 3.4 Segment with stiffness forces

The spring force at each node acts along the line of the segment, in the same direction as the segment's unit vector. The unit vector in the direction from $\bar{\boldsymbol{p}}_{\boldsymbol{i,j-1}}$ to $\bar{\boldsymbol{p}}_{\boldsymbol{i,j}}$ is the line vector divided by its magnitude:

$$\hat{\boldsymbol{l}}_{\boldsymbol{i,j}} = \bar{\boldsymbol{l}}_{\boldsymbol{i,j}} / \boldsymbol{l}_{\boldsymbol{i,j}} \tag{3.12}$$

The force at node $\bar{\boldsymbol{p}}_{\boldsymbol{i,j}}$ due to the extension of the segment is then

$$\boldsymbol{f}_{\boldsymbol{i,j}}^{\boldsymbol{K_{i,j}}} = -\boldsymbol{K}_{\boldsymbol{i,j}} \Delta \boldsymbol{l}_{\boldsymbol{i,j}} \hat{\boldsymbol{l}}_{\boldsymbol{i,j}} \tag{3.13}$$

The force at node $\bar{\boldsymbol{p}}_{\boldsymbol{i,j-1}}$ is

$$\boldsymbol{f}_{i,j-1}^{K_{i,j}} = \boldsymbol{K}_{i,j}\Delta\boldsymbol{l}_{i,j}\hat{\boldsymbol{l}}_{i,j} \tag{3.14}$$

The potential energy from the stiffness of each segment is

$$\boldsymbol{V}(\boldsymbol{K}_{i,j}) = \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{K}_{i,j}\Delta\boldsymbol{l}_{i,j}^{\mathbf{2}} \tag{3.15}$$

3.5 Drag on the Mooring Lines

The drag forces from the fluid velocity are modeled to act on the center of each segment of the mooring lines. The magnitude of the drag force can be found from the equation of the form

$$\boldsymbol{f}_{\boldsymbol{D}} = \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\rho}_{\boldsymbol{f}}\boldsymbol{C}_{\boldsymbol{D}}\boldsymbol{A}\boldsymbol{u}^{\mathbf{2}} \tag{3.16}$$

where $\boldsymbol{\rho}_{\boldsymbol{f}}$ is the fluid density, $\boldsymbol{C}_{\boldsymbol{D}}$ is the coefficient of drag, $\boldsymbol{A}$ is the reference area and $\boldsymbol{u}$ is the speed of the object relative to the fluid. In equilibrium the velocity of each object is zero and therefore $\boldsymbol{u}$ is equal to the speed of the fluid.

To model drag as a conservative force it is modeled as a constant force (with constant magnitude and direction) applied to the system. Because the drag forces depend on the system's orientation relative to the fluid velocity, drag forces are updated between iterative solutions as discussed in section 5.1

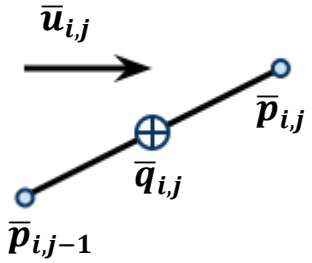
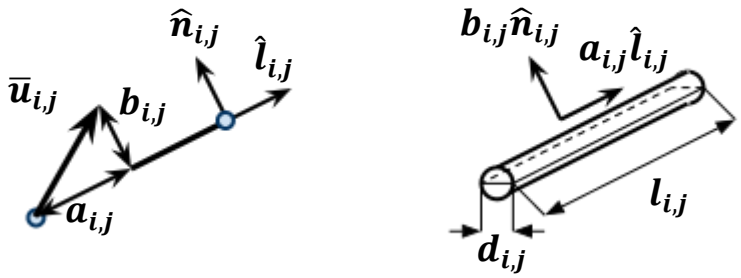


Figure 3.5 Stream velocity on line segment

The velocity of the fluid on each segment's center is denoted $\bar{\boldsymbol{u}}_{\boldsymbol{i,j}}$ (Figure 3.5). Depending on the orientation of the segment relative to the stream velocity, the fluid flows over different areas of the segment. In particular, part of the fluid can flow across, or perpendicular, to the segment and part of the fluid can flow along, or parallel, to the segment. The relative fluid velocity can be written in terms of its components parallel and perpendicular to the segment as

$$\bar{\boldsymbol{u}}_{\boldsymbol{i,j}} = \boldsymbol{a}_{\boldsymbol{i,j}}\hat{\boldsymbol{l}}_{\boldsymbol{i,j}} + \boldsymbol{b}_{\boldsymbol{i,j}}\hat{\boldsymbol{n}}_{\boldsymbol{i,j}} \tag{3.17}$$

where $\hat{\boldsymbol{l}}_{\boldsymbol{i,j}}$ is the unit vector in the direction from $\bar{\boldsymbol{p}}_{\boldsymbol{i,j-1}}$ to $\bar{\boldsymbol{p}}_{\boldsymbol{i,j}}$, and $\hat{\boldsymbol{n}}_{\boldsymbol{i,j}}$ is the unit vector perpendicular to the segment (see Figure 3.6a).



(a) **(b)**

Figure 3.6 (a) Segment unit vectors; **(b)** Segment dimensions and relative fluid velocities

The coefficient $\boldsymbol{a}_{\boldsymbol{i,j}}$ is the speed of the fluid velocity parallel to the segment and is found by the equation

$$\boldsymbol{a}_{\boldsymbol{i,j}} = \bar{\boldsymbol{u}}_{\boldsymbol{i,j}} \cdot \hat{\boldsymbol{l}}_{\boldsymbol{i,j}} \tag{3.18}$$

Because the magnitude of the relative velocity is $\left|\bar{\boldsymbol{u}}_{\boldsymbol{i,j}}\right| = \sqrt{\boldsymbol{a}_{\boldsymbol{i,j}}^2 + \boldsymbol{b}_{\boldsymbol{i,j}}^2}$, the coefficient $\boldsymbol{b}_{\boldsymbol{i,j}}$ can be found from the equation

$$\boldsymbol{b_{i,j}} = \sqrt{\left|\overline{\boldsymbol{u}}_{\boldsymbol{i,j}}\right|^2 - \boldsymbol{a_{i,j}^2}} \tag{3.19}$$

The unit vector $\hat{\boldsymbol{n}}_{\boldsymbol{i,j}}$ is then

$$\hat{\boldsymbol{n}}_{\boldsymbol{i,j}} = \frac{\mathbf{1}}{\boldsymbol{b_{i,j}}}\left(\overline{\boldsymbol{u}}_{\boldsymbol{i,j}} - \boldsymbol{a_{i,j}}\hat{\boldsymbol{l}}_{\boldsymbol{i,j}}\right) \tag{3.20}$$

The area of the segment projected in the $\hat{\boldsymbol{n}}_{\boldsymbol{i,j}}$ direction (see Figure 3.6b) is

$$\boldsymbol{A_{\hat{n}_{i,j}}} = \boldsymbol{l_{i,j}d_{i,j}} \tag{3.21}$$

where $\boldsymbol{d_{i,j}}$ is the diameter of the segment.

The area of the segment projected in the $\hat{\boldsymbol{l}}_{\boldsymbol{i,j}}$ direction (surface area, see Figure 3.6b) is

$$\boldsymbol{A_{\hat{l}_{i,j}}} = \boldsymbol{\pi d_{i,j}l_{i,j}} \tag{3.22}$$

The drag force acting on the segment is approximated as

$$\overline{\boldsymbol{F}}_{\boldsymbol{D,i,j}} = \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\rho_f C_{D\hat{l}}A_{\hat{l}_{i,j}}a_{i,j}}\left|\boldsymbol{a_{i,j}}\right|\hat{\boldsymbol{l}}_{\boldsymbol{i,j}} + \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\rho_f C_{D\hat{n}}A_{\hat{n}_{i,j}}b_{i,j}}\left|\boldsymbol{b_{i,j}}\right|\hat{\boldsymbol{n}}_{\boldsymbol{i,j}} \tag{3.23}$$

where $\boldsymbol{C_{D\hat{l}}}$ and $\boldsymbol{C_{D\hat{n}}}$ are the coefficients of drag in the parallel and normal directions respectively.

The potential energy of the drag force on each segment is

$$\boldsymbol{V}\left(\overline{\boldsymbol{F}}_{\boldsymbol{D,i,j}}\right) = -\boldsymbol{F_{D,i,j,X}q_{i,j,X}} - \boldsymbol{F_{D,i,j,Y}q_{i,j,Y}} - \boldsymbol{F_{D,i,j,Z}q_{i,j,Z}} \tag{3.24}$$

3.6 Drag on the Body

The drag forces on the body are found in a similar way to that on the mooring lines with the same form as Equation 3.16. The fluid velocity is first found in terms of the body-

fixed coordinates to find the drag forces in the body-fixed frame. The drag forces are then transformed into the inertial frame coordinates.

The fluid velocity in the vicinity of the body's center of mass in the inertial frame is denoted $\bar{\boldsymbol{u}}_{\boldsymbol{fo'}}$. The fluid velocity in the body-fixed frame is

$$\bar{\boldsymbol{u}}'_{\boldsymbol{fo'}} = \boldsymbol{T}^{\boldsymbol{T}}\bar{\boldsymbol{u}}_{\boldsymbol{fo'}} \tag{3.25}$$

The drag on the body in the body-fixed frame is approximated as

$$\bar{\boldsymbol{F}}'_{\boldsymbol{D,o'}} = \begin{bmatrix} \boldsymbol{F}_{\boldsymbol{D,o'x'}} \\ \boldsymbol{F}_{\boldsymbol{D,o'y'}} \\ \boldsymbol{F}_{\boldsymbol{D,o'z'}} \end{bmatrix}; \quad \begin{aligned} \boldsymbol{F}_{\boldsymbol{D,o'x'}} &= \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{\rho}_{\boldsymbol{f}}\boldsymbol{C}_{\boldsymbol{Dx'}}\boldsymbol{A}_{\boldsymbol{o'x'}}\boldsymbol{u}_{\boldsymbol{fo'x'}}\left|\boldsymbol{u}_{\boldsymbol{fo'x'}}\right| \\ \boldsymbol{F}_{\boldsymbol{D,o'y'}} &= \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{\rho}_{\boldsymbol{f}}\boldsymbol{C}_{\boldsymbol{Dy'}}\boldsymbol{A}_{\boldsymbol{o'y'}}\boldsymbol{u}_{\boldsymbol{fo'y'}}\left|\boldsymbol{u}_{\boldsymbol{fo'y'}}\right| \\ \boldsymbol{F}_{\boldsymbol{D,o'z'}} &= \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{\rho}_{\boldsymbol{f}}\boldsymbol{C}_{\boldsymbol{Dz'}}\boldsymbol{A}_{\boldsymbol{o'z'}}\boldsymbol{u}_{\boldsymbol{fo'z'}}\left|\boldsymbol{u}_{\boldsymbol{fo'z'}}\right| \end{aligned} \tag{3.26}$$

- $\boldsymbol{C}_{\boldsymbol{Dx'}}, \boldsymbol{C}_{\boldsymbol{Dy'}}, \boldsymbol{C}_{\boldsymbol{Dz'}}$ – Coefficient of drag in the $\boldsymbol{x'}, \boldsymbol{y'}$ and $\boldsymbol{z'}$ directions respectively
- $\boldsymbol{A}_{\boldsymbol{o'x'}}, \boldsymbol{A}_{\boldsymbol{o'y'}}, \boldsymbol{A}_{\boldsymbol{o'z'}}$ – Effective surface areas in the $\boldsymbol{x'}, \boldsymbol{y'}$ and $\boldsymbol{z'}$ directions respectively

Note that the reference areas used in the body drag force equations are referred to as effective surface areas. This is because the drag coefficients and reference areas depend on how the drag forces are found and the effective surface areas may not be a projected area of the body. For modeling purposes the drag coefficients or the effective surface areas can be changed to achieve a desired drag force.

The drag force on the body in the inertial frame is

$$\bar{\boldsymbol{F}}_{\boldsymbol{D,o'}} = \boldsymbol{T}\bar{\boldsymbol{F}}'_{\boldsymbol{D,o'}} \tag{3.27}$$

The potential energy of the drag force on the body is

$$\boldsymbol{V}\left(\bar{\boldsymbol{F}}_{\boldsymbol{D,o'}}\right) = -\boldsymbol{F}_{\boldsymbol{D,o'X}}\boldsymbol{x} - \boldsymbol{F}_{\boldsymbol{D,o'Y}}\boldsymbol{y} - \boldsymbol{F}_{\boldsymbol{D,o'Z}}\boldsymbol{z} \tag{3.28}$$

3.7 Applied Body Loads

This model allows for constant forces and torques to be applied to the body. Examples of applied body loads could be forces or torques on the body due to the rotation or operation of the turbine and generator. The potential energy of the applied body force is

$$V(\overline{F}_{A,O'}) = -F_{A,O'X}x - F_{A,O'Y}y - F_{A,O'Z}z \tag{3.29}$$

The potential energy of the applied torque in the body-fixed frame $(\overline{T}'_{A,O'} = T^T \overline{T}_{A,O'})$ is

$$V(\overline{T}'_{A,O'}) = -T'_{A,O'\alpha}\alpha - T'_{A,O'\beta}\beta - T'_{A,O'\gamma}\gamma \tag{3.30}$$

4 Benchmark Tests

For all test problems the tolerance on the objective function, constraints and variables specified for the Matlab minimization function *fmincon* were set to their default values of 10^{-6}.

4.1 Elastic Catenary and Error Rate of Convergence

To test the accuracy of the profile of a suspended elastic cable the minimized equilibrium solution was compared to an analytical solution given by Irvine (3). For easier application the solution can be rewritten for the case where one end of the cable is fixed at the origin and a horizontal and vertical force are applied at the other end (7), (Figure 4.1).

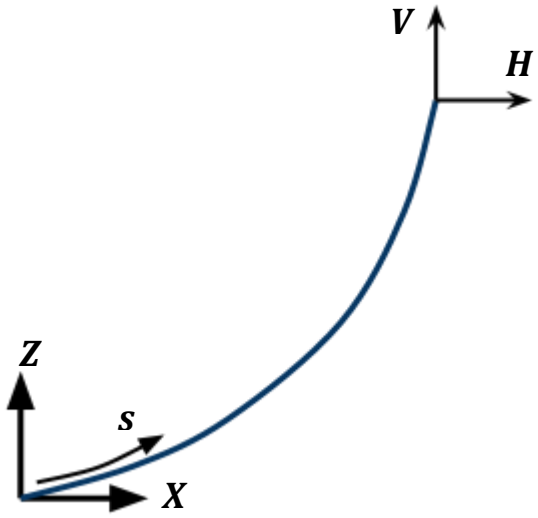


Figure 4.1 Elastic catenary setup

The solutions for the $\boldsymbol{X}$ and $\boldsymbol{Z}$ profile of the cable as a function of the unstretched arc length $\boldsymbol{s}$ using the setup shown are then (7)

$$\boldsymbol{X(s) = \frac{Hs}{EA_0} + \frac{H}{\rho g}\left[\sinh^{-1}\left(\frac{V - \rho g L_0 + \rho g s}{H}\right) - \sinh^{-1}\left(\frac{V - \rho g L_0}{H}\right)\right]} \quad (4.1)$$

$$\boldsymbol{Z(s) = \frac{s}{EA_0}\left(V - \rho g L_0 + \frac{1}{2}\rho g s\right) + \frac{H}{\rho g}\left[\sqrt{1 + \left(\frac{V - \rho g L_0 + \rho g s}{H}\right)^2} - \sqrt{1 + \left(\frac{V - \rho g L_0}{H}\right)^2}\right]} \quad (4.2)$$

$\boldsymbol{E}$ is the elastic modulus, $\boldsymbol{A_0}$ cross-sectional area, $\boldsymbol{L_0}$ unstretched line length, $\boldsymbol{\rho}$ mass per unit length of the line, $\boldsymbol{g}$ gravity, $\boldsymbol{H}$ applied horizontal force and $\boldsymbol{V}$ applied vertical force.

The catenary test problem was setup with the following properties: $E = 10^8\ Pa$, $line\ diameter\ =\ 0.01\ m,\ L_0 = 10\ m, \rho = 0.7\ kg/m, H = 50\ N$ and $V = 75\ N$.

The initial displacements for the minimization solution were defined by creating a line from the origin to the coordinates (8 m, 6 m), near the equilibrium of the free end based on the analytical solution, and dividing the line into 5 equal segments. The solution using 5 segments was used to double the number of segments of the line by using the coordinates of the nodes and segment centers as the new initial node coordinates for a solution using 10 segments. Each successive solution used the previous solution as initial coordinates to double the number of nodes used until the problem was solved using 40 line segments. The analytical and minimized solutions of the catenary test are shown in Figure 4.2.

The error between the minimized and analytical solutions was calculated by comparing the coordinates of the line nodes to the analytical solution at the corresponding unstretched arc length. The error is calculated as the difference between the analytical solution and the minimized solution.

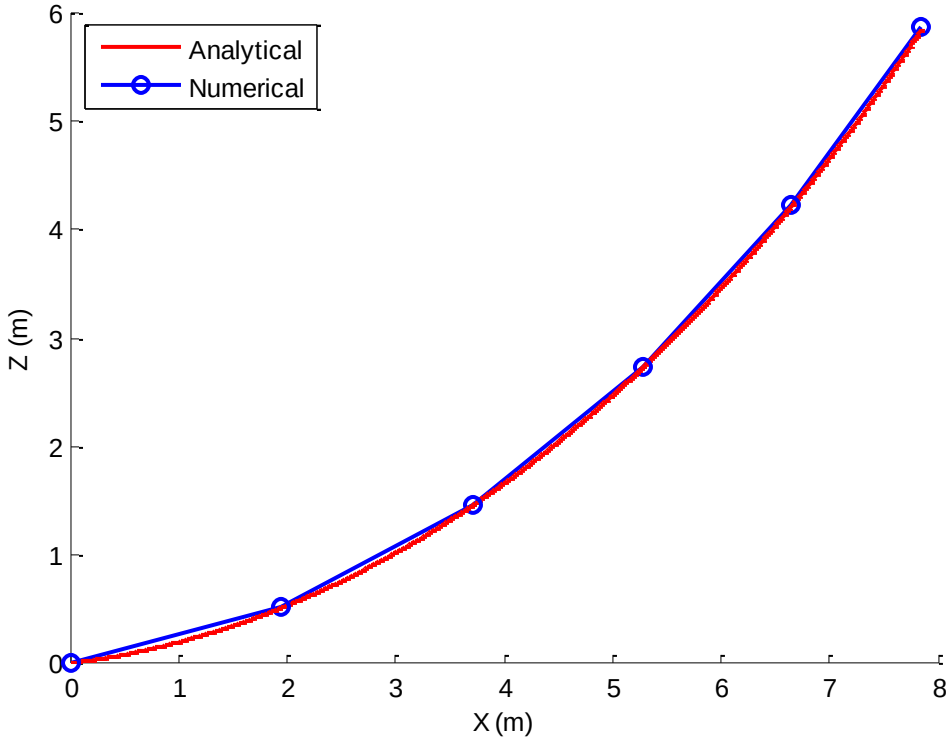
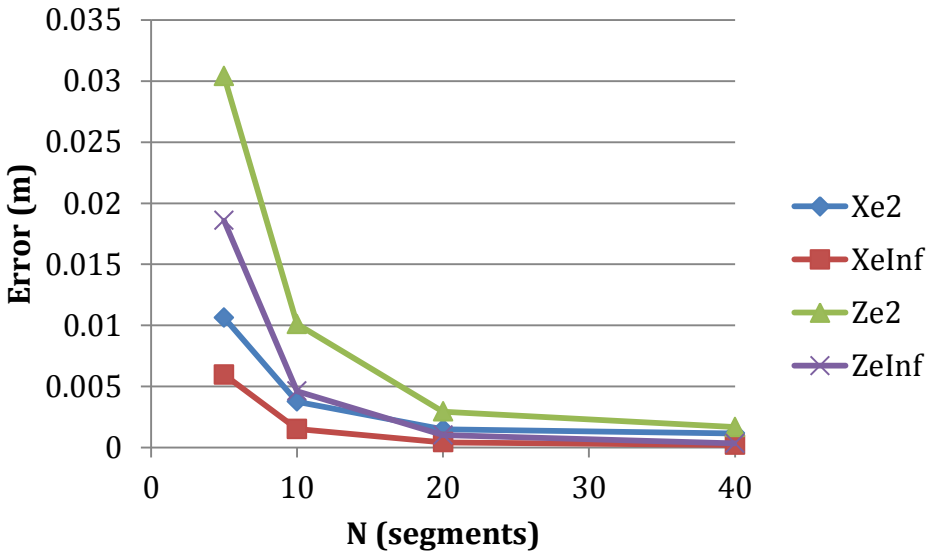
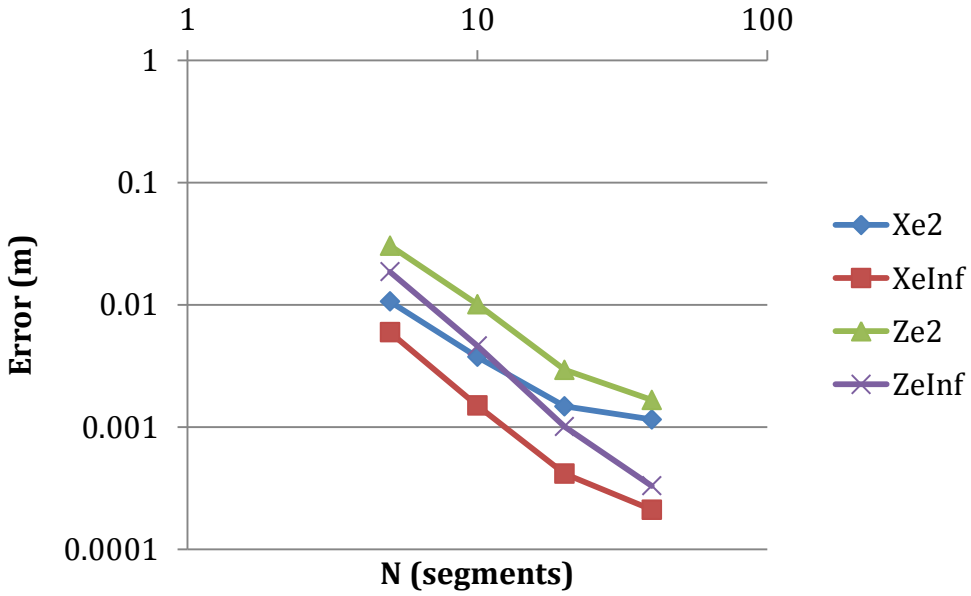


Figure 4.2 Elastic catenary analytical and minimized solution using 5 line segments

The 2-norm and the infinite-norm of the $\boldsymbol{X}$ and $\boldsymbol{Z}$ components of the errors were found. The elastic catenary errors are plotted in Figure 4.3a and on a log-log scale in Figure 4.3b. The 2-norm and infinite-norm of the $\boldsymbol{X}$ component errors are labeled Xe2 and XeInf respectively, for example. Table 4.1 lists the slopes of the catenary errors calculated between N=5 and 20 segments, using the log(Error) versus the log(N). The slopes on the log-log scale give the exponent on N and describe how the error varies with the number of line segments. For 5, 10 and 20 segments the infinite-norm error decreases on the order near $1/N^2$ and the 2-norm error near $1/N^{1.5}$, seen as close to straight lines on the log-log scale. At 40 segments the errors start to level out on the log-log scale and do not follow the trend as closely. The error could be limited by the tolerance set on the solver *fmincon*.



(a) Catenary errors



(b) Catenary errors, log-log scale

Figure 4.3 Catenary errors versus number of line segments

Table 4.1 Slope of catenary errors on log-log scale, between N=5 and N=20

$\|Xe\|_2$	$\|Xe\|_\infty$	$\|Ze\|_2$	$\|Ze\|_\infty$
-1.42	-1.92	-1.69	-2.10

4.2 Rigid Body Rotations and Stable Equilibrium

To test large rigid body rotations, the minimized equilibrium solution was compared to an analytical solution of a 2D problem. The test problem consists of a rigid body of width $\boldsymbol{W}$ attached to two linear springs at its ends. The left and right springs have springs constants $\boldsymbol{K_1}$ and $\boldsymbol{K_2}$ respectively, have the same unstretched length $\boldsymbol{L}$ and are fixed to the ground separated by the same width as the body. A constant vertical force $\boldsymbol{F}$ is applied to the body a distance $\boldsymbol{l_3}$ measured along the body from the left end and a constant torque $\boldsymbol{T}$ is applied to the body in the positive $\boldsymbol{\theta}$ direction (counterclockwise). The problem setup is shown in Figure 4.4a. The 2D rigid body test problem has 3 degrees of freedom. The coordinates of the left end of the body $(\boldsymbol{x_1}, \boldsymbol{y_1})$ and the angle $\boldsymbol{\theta}$ measured from the positive $\boldsymbol{X}$ direction to the body were chosen as the independent displacements. The coordinate system and degrees of freedom are shown in Figure 4.4b.

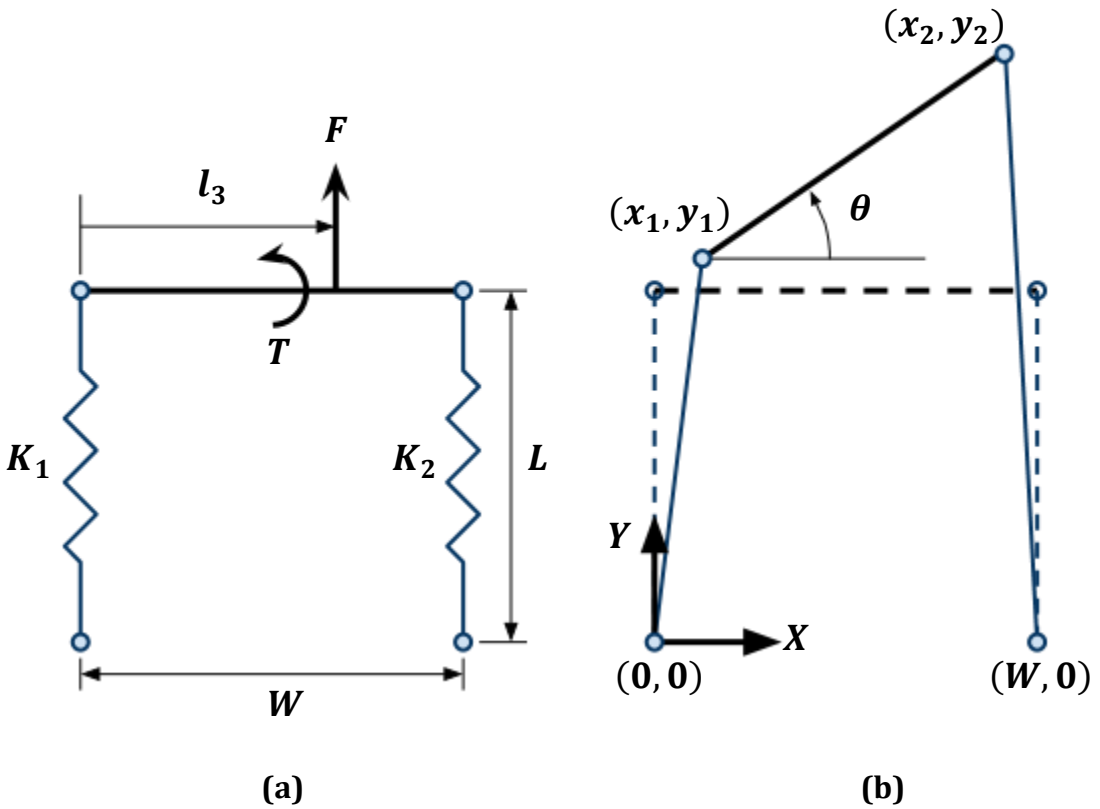


Figure 4.4 (a) Rigid body test setup**; (b)** Deformed shape and coordinate system

The analytical solution was found by setting the gradients of the potential energy of the system to zero and solving for the independent displacements. The total potential energy of the system is

$$\boldsymbol{V = \frac{1}{2}K_1(\Delta l_1)^2 + \frac{1}{2}K_2(\Delta l_2)^2 - Fy_3 - T\theta} \tag{4.3}$$

$$\boldsymbol{\Delta l_1 = \sqrt{x_1^2 + y_1^2} - L}$$
$$\boldsymbol{\Delta l_2 = \sqrt{(x_1 + W\cos\theta - W)^2 + (y_1 + W\sin\theta)^2} - L}$$
$$\boldsymbol{y_3 = y_1 + l_3\sin\theta}$$

The analytical equations to be solved for are:

$$\begin{aligned} \boldsymbol{\delta V/\delta x_1 = 0} \\ \boldsymbol{\delta V/\delta y_1 = 0} \\ \boldsymbol{\delta V/\delta \theta = 0} \end{aligned} \tag{4.4}$$

The analytical equations were solved using the *fsolve* function in Matlab. The problem was set up with the following properties: $K_1 = 10\ N/m$, $K_2 = 5\ N/m$, $L = 3\ m$, $W = 5\ m$, $l_3 = 3\ m$, $F = 20\ N$ and $T = 25\ Nm$. The errors were calculated as the difference between the coordinates of the left spring's node attached to the body, the body's rotation and the independent displacements from the analytical solution.

The initial displacements for the minimization solution were defined by making the body initially horizontal at a height L. The minimized solution gives the results: $x_1 = 3.262\ m$, $y_1 = 1.431\ m$, $\theta = 1.596\ rad$, and total potential energy $V = -92.8041\ J$.

If the initial guess of the displacements for the analytical solution were the same as for the minimization solution ($x_{1,0} = 0$, $y_{1,0} = L$, $\theta_0 = 0$), the analytical solution gives the results: $x_1 = 1.299\ m$, $y_1 = 2.923\ m$, $\theta = 0.8446\ rad$, $V = -90.5237$ J. This analytical solution along with the minimized solution is shown in Figure 4.5.

The stability of an equilibrium solution can be related to the eigenvalues of the Hessian of the unconstrained potential energy function. The Hessian, $\boldsymbol{H}$, is a matrix of the second partial derivatives of the potential energy function $\boldsymbol{V}$:

$$\boldsymbol{H} = \boldsymbol{\nabla^2 V} = \begin{bmatrix} \boldsymbol{\delta V^2/\delta x_1 \delta x_1} & \boldsymbol{\delta V^2/\delta x_1 \delta y_1} & \boldsymbol{\delta V^2/\delta x_1 \delta \theta} \\ \boldsymbol{\delta V^2/\delta y_1 \delta x_1} & \boldsymbol{\delta V^2/\delta y_1 \delta y_1} & \boldsymbol{\delta V^2/\delta y_1 \delta \theta} \\ \boldsymbol{\delta V^2/\delta \theta \delta x_1} & \boldsymbol{\delta V^2/\delta \theta \delta y_1} & \boldsymbol{\delta V^2/\delta \theta \delta \theta} \end{bmatrix} \tag{4.5}$$

The Eigen values of the Hessian evaluated at this analytical solution include an Eigen value near zero and is considered a marginally stable equilibrium: $\lambda_1 = 81.87$, $\lambda_2 = 12.88$, $\lambda_3 = 3.875 * 10^{-7}$.

The results are different because of how the analytical and minimization methods find equilibrium. The analytical solution finds equilibrium near the initial guess while the minimization solution finds a stable equilibrium by minimizing the potential energy.

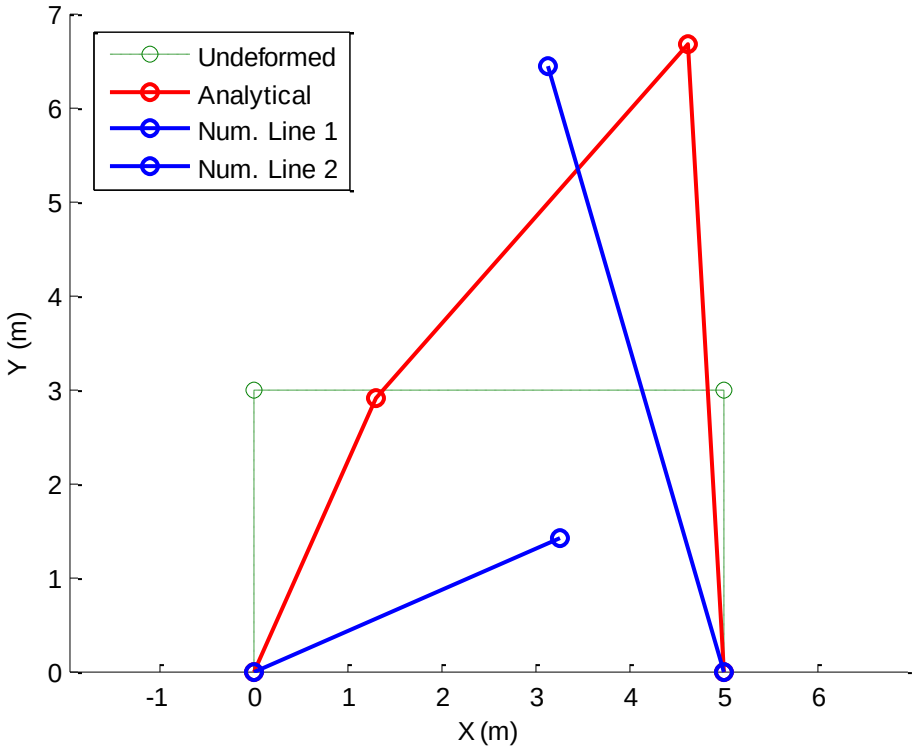


Figure 4.5 Stable numerical and unstable analytical solutions of rigid body test

Given an initial guess closer to the minimized solution, $x_{1,0} = 3, y_{1,0} = 1, \theta_0 = 1.5$, the analytical solution gives the same stable equilibrium as the minimized solution. The errors of these results are listed in Table 4.2 and are on the order of magnitude of 10^{-6}, the same order of magnitude as the tolerances specified for the Matlab solver *fmincon*. The Eigen values of the Hessian evaluated at these results, $\lambda_1 = 48.98,\ \lambda_2 = 10.12, \lambda_3 = 3.480$, show a more stable equilibrium than the analytical solution given the previous initial guess. The stable equilibrium of the system is shown in Figure 4.6.

Table 4.2 Rigid body rotation errors

$x_1\ (m)$	$y_1\ (m)$	$\theta\ (rad)$
$7.084 * 10^{-6}$	$4.358 * 10^{-6}$	$1.375 * 10^{-6}$

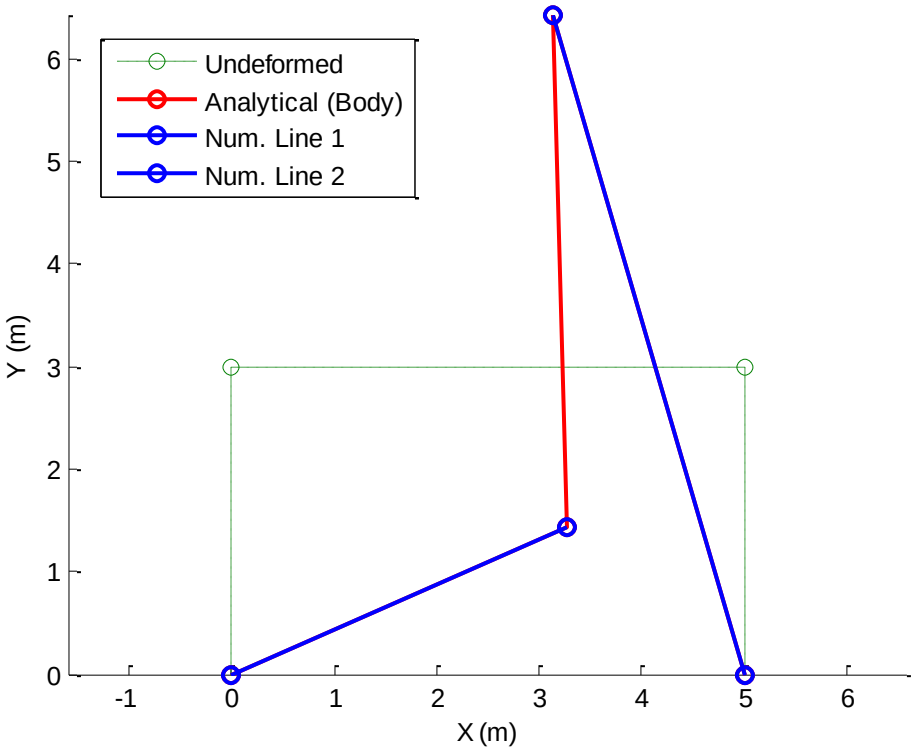


Figure 4.6 Undeformed and stable equilibrium positions of rigid body test

4.3 Slack and Ground Constraints

To observe that the slack and ground constraints can be satisfied a test problem was setup consisting of two elastic lines and an applied force. The lines are fixed to each other at one end and the other ends are fixed to the ground at separate points. A vertical force $\boldsymbol{F}$ is applied at the end of the lines where they are connected. The unstretched length of the left line is long enough such that when the force is applied the left line must become slack with part of it laying on the ground and the rest hanging vertically below the applied force. The right line must stretch and is under tension from the applied force. The right line was defined to have a stiffness of $K_1 = 100\ N/m$ and unstretched length $L_1 = 1\ m$, and $K_2 = 40\ N/m$ and $L_2 = 2.5\ m$ for the left line. Both lines have a linear density of $\rho_L = 0.1\ kg/m$. A force of $F = 10\ N$ was applied to the system. Drag and buoyancy were not modeled in the test. The lines were separated by a distance of 1 m on the ground. See Figure 4.7 for the setup.

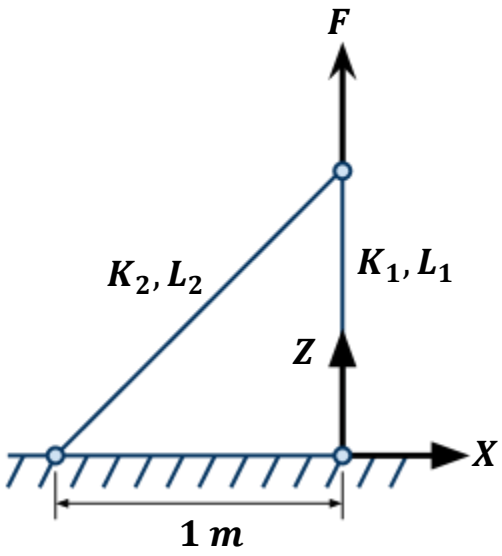


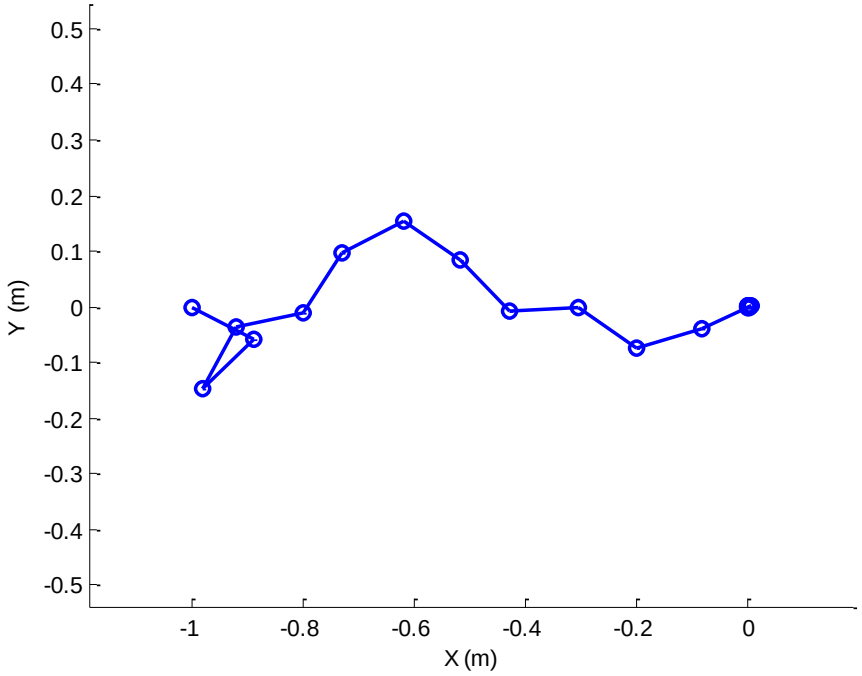
Figure 4.7 Slack test setup

The initial positions of the elastic lines were defined as straight lines from their ground attachment point to the coordinates $(0\ m, 0\ m, 2.5\ m)$, where the force is applied, such that the left line is initially stretched. The problem was solved with one solution using

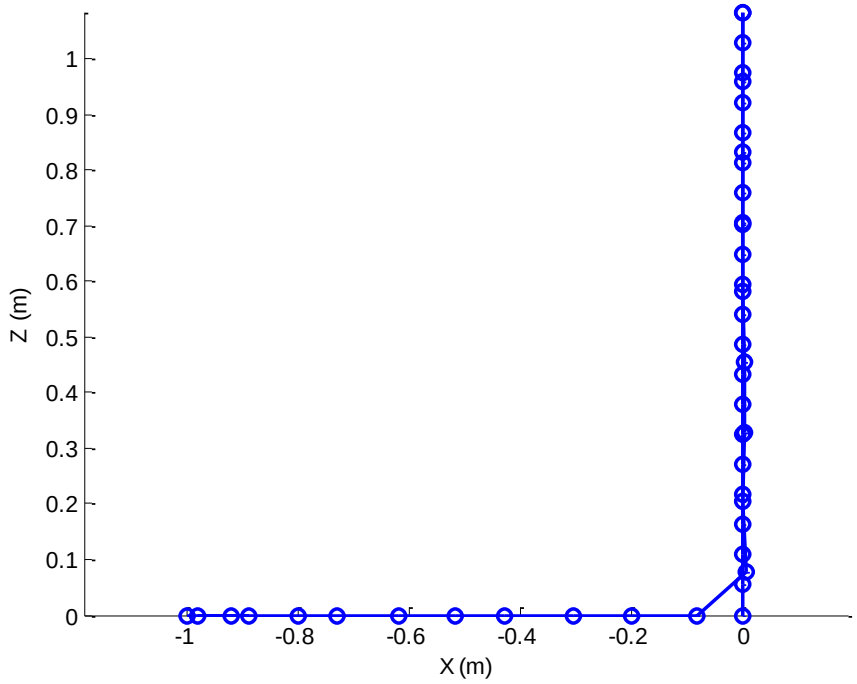
20 line segments for each mooring line. The solution is plotted in Figure 4.8. The top view shows how the left line became slack and 11 of its segments scattered flat on the ground. One of segments has one end on the ground and the other end suspended vertically below the applied force with the rest of its segments, as seen in the front view.

Table 4.3 lists the tension forces in each line segment, numbered with the first segment at the applied force toward the last segment attached to the ground. Segment numbers 9 through 20 of the left line are the segments that touch the ground have a tension force near zero indicating they are slack with a length near its unstretched length. Segment numbers 1 through 8 of the left line are suspended below the applied force and linearly increase in tension from the lowest to highest segment as the line supports its own weight. The right line is taut from the applied force, also linearly increasing in tension from its lowest to highest segment due to its own weight.

Systems with slack or grounded lines may be difficult to solve because the solver *fmincon* can take many iterations to adjust the system before it converges to a minimum. In some tests the solver may not converge to minimum, as discussed in section 5.2. It may be necessary to increase the number of iterations or function evaluations allowed by the solver in order for it to converge.



(a) Top view, showing slack line segments scattered on ground



(b) Front view, showing line segments suspended vertically below applied force

Figure 4.8 Plots of the elastic lines of the slack test solution

Table 4.3 Line segment tensions (N) from slack test solution

Segment No.	Left Line (Slack Line)	Right Line (Taut Line)
1	0.9841	8.934
2	0.8587	8.886
3	0.7365	8.834
4	0.6124	8.784
5	0.4909	8.736
6	0.3697	8.689
7	0.2482	8.638
8	0.1250	8.590
9	$1.742 * 10^{-6}$	8.542
10	$4.108 * 10^{-6}$	8.490
11	$5.147 * 10^{-6}$	8.441
12	$4.045 * 10^{-7}$	8.388
13	$3.329 * 10^{-6}$	8.343
14	$2.356 * 10^{-6}$	8.296
15	$8.545 * 10^{-6}$	8.242
16	$2.904 * 10^{-5}$	8.197
17	$2.824 * 10^{-6}$	8.149
18	$2.378 * 10^{-5}$	8.097
19	$3.868 * 10^{-4}$	8.050
20	$1.176 * 10^{-5}$	7.999

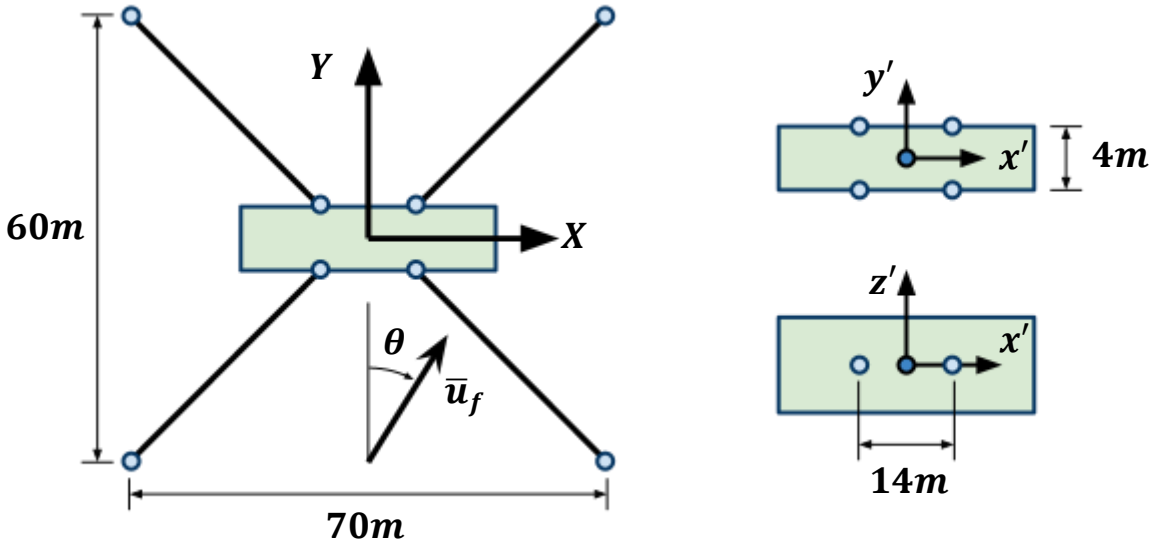
5 Test Problems

5.1 Iterative Solutions and Drag Forces

A test problem consisting of a body and four mooring lines representing a basic underwater turbine system was run to observe the equilibrium solutions using iterative solutions of the same problem. Iterative solutions are necessary when drag forces are involved because of how the drag forces are calculated and change between each solution if the equilibrium solutions are different. Drag forces are calculated for each solution process according to the initial position of the body and mooring line segments. The drag forces are used with a constant magnitude and direction for the solution process. As the equilibrium position of the body and line segments differ from their initial positions their drag forces no longer correspond with their current position and can differ greatly in direction and magnitude. The drag forces are recalculated using the previous solution's equilibrium position to more accurately represent the next solutions actual drag forces. The problem is solved again using the recalculated drag forces and previous solution as the new initial position. The iterative process can be repeated until there is a desired small enough change between solutions.

The setup used for these test problems consists of a symmetric mooring system centered at the inertial frame origin (Figure 5.1a). A constant fluid velocity of magnitude $|\bar{u}_f| = 4\,m/s$ acts on the system in the positive $\boldsymbol{Y}$ direction ($\theta = 0$ in Figure 5.1a). The fluid velocity is constant with depth ($\boldsymbol{Z}$ direction), although actual stream or current profiles may vary with depth and decrease to zero near the seabed. A constant velocity profile may be a harder problem to solve because it causes greater drag forces near the seabed and was used for test purposes.

The attachment points of the mooring lines to the body are symmetric with respect to the body-fixed frame and are level with the body's COM (Figure 5.1b).



(a) Inertial frame and system setup, top view, showing mooring line ground attachment points and fluid velocity

(b) Body-fixed frame and mooring line attachment points

Figure 5.1 Test system mooring line and fluid velocity setup

The following tables list the property values used in these test problems. The values in Table 5.2 and Table 5.3 are generally guesses and do not represent an actual system. There are, however, some relationships between values and physical properties that can be used to estimate some values. The volume of the body was chosen such that its buoyant force is greater than the weight of the body and mooring lines combined. The body's dimensions are modeled as a rectangular box with a width of 28 m, depth of 4 m, and a height of 8.5 m. Because an actual turbine is not a solid rectangular shape the effective surface areas of the body are only a small percentage of the rectangular area. For areas of the turbine in which the water is designed to flow through (the y' and z' directions) the effective areas are about 10% of the rectangular area and larger for other areas (50% for the x' direction). The body's initial COM coordinates must be chosen such that all the mooring lines are slightly stretched to avoid invalid initial conditions of compressed mooring lines (discussed in Appendix 1).

Table 5.1 System constant properties

Property	Value
Gravity	$g = 9.81\, m/s^2$
Fluid density	$\rho_f = 1020\, kg/m^3$
Fluid speed	$\left\|\bar{u}_f\right\| = 4\, m/s$

Table 5.2 Body properties

Property	Value
Mass	$M = 5{,}000\ kg$
Volume	$V_{body} = 10\, m^3$
Center of Buoyancy	$\bar{c'}_b = [x'_b\ y'_b\ z'_b]^T = [0 \quad 0 \quad 0]^T\ (m)$
Coefficients of Drag	$\bar{C}_D = \left[C_{Dx\prime}\ C_{Dy\prime}\ C_{Dz\prime}\right]^T = [0.5 \quad 0.5 \quad 0.5]^T$
Effective Surface Areas	$\bar{A}_{o\prime} = \left[A_{o\prime x\prime}\ A_{o\prime y\prime}\ A_{o\prime z\prime}\right]^T = [16 \quad 24 \quad 11]^T\ (m^2)$
Applied Forces	$\bar{F}_{A,o\prime} = \left[F_{A,o\prime X}\ F_{A,o\prime Y}\ F_{A,o\prime Z}\right]^T = [0 \quad 0 \quad 0]^T\ (N)$
Applied Torques	$\bar{T}_{A,o\prime} = \left[T_{A,o\prime X}\ T_{A,o\prime Y}\ T_{A,o\prime Z}\right]^T = [0 \quad 0 \quad 0]^T\ (Nm)$
Initial COM coordinates	$\bar{c}_{o\prime} = [x \quad y \quad z]^T = [0 \quad 0 \quad 20]^T$ (m)
Initial Euler angles	$[\alpha_i \quad \beta_i \quad \gamma_i]^T = [0 \quad 0 \quad 0]^T$ (radians)

The tangential and normal coefficients of drag for the mooring lines are approximated by the coefficients for a cylinder. The actual diameter of the mooring lines should be used because this value is used to calculate the surface areas and volume of the mooring lines. With a diameter of 5 *cm*, the elastic stiffness of the mooring lines is based on an elastic modulus of *10 Gpa*.

Table 5.3 Mooring Line properties
(The same values were used for all mooring lines)

Property	Value
Tangential Coefficient of Drag	$C_{D\hat{l}} = 0.3$
Normal Coefficient of Drag	$C_{D\hat{n}} = 1$
Linear Density	$\rho_i = 10\, kg/m$
Undeformed Length	$L_i = 44\, m$
Diameter	$d_i = 0.05\, m$
Stiffness	$K_i = 446{,}250\, N/m$

Table 5.4 Mooring Line attachment points

	Ground Attachment (m)	**Body Attachment (m)**
Line #	$\overline{\boldsymbol{p}}_{i,G} = \left[\boldsymbol{p}_{i,G,X}\ \boldsymbol{p}_{i,G,Y}\ \boldsymbol{p}_{i,G,Z}\right]^{\boldsymbol{T}}$	$\overline{\boldsymbol{p}}'_{i,B} = \left[\boldsymbol{p}_{i,B,x'}\ \boldsymbol{p}_{i,B,y'}\ \boldsymbol{p}_{i,B,z'}\right]^{\boldsymbol{T}}$
1	$[35 \quad -30 \quad 0]^T$	$[7 \quad -2 \quad 0]^T$
2	$[35 \quad 30 \quad 0]^T$	$[7 \quad 2 \quad 0]^T$
3	$[-35 \quad 30 \quad 0]^T$	$[-7 \quad 2 \quad 0]^T$
4	$[-35 \quad -30 \quad 0]^T$	$[-7 \quad -2 \quad 0]^T$

The test problem was solved with four iterative solutions. The first two solutions used 10 segments per mooring line. The segment centers of the second solution were used to define new segment nodes to double the number of line segments. 20 segments were then used for the third and fourth solutions. Figure 5.2 through Figure 5.5 show a top view of each equilibrium solution with the tangential and normal drag forces on each mooring line segment.

The segment drag forces are the most inaccurate after the first solution. The initial position used for the first solution centers the body's COM above the center of the seabed attachment points and draws a straight line from the seabed attachment point to the body attachment point to define the line nodes. As the body is pushed back from the drag force the mooring lines downstream of the flow become curved from the seabed to the body. Figure 5.2 shows how the initial drag forces on the downstream line segments do not nearly match the solutions current normal and tangential directions.

The second solution shows a much closer alignment of the initial drag forces to the segments solution orientations (Figure 5.3) because the system is displaced less between the initial and final positions. The third and fourth iterations show increasingly more accurate alignments of drag forces between the initial and final positions (Figure 5.4 and Figure 5.5).

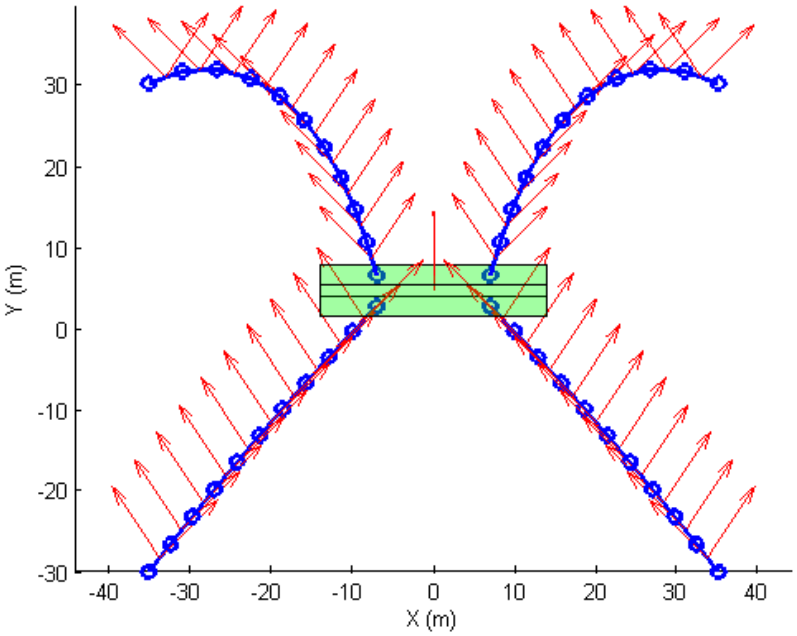


Figure 5.2 First solution and segment drag forces

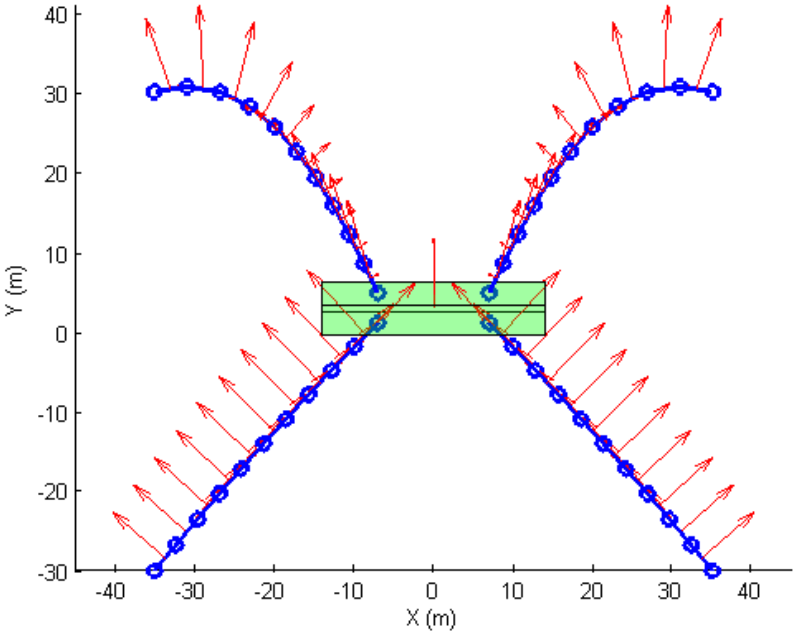


Figure 5.3 Second solution and segment drag forces

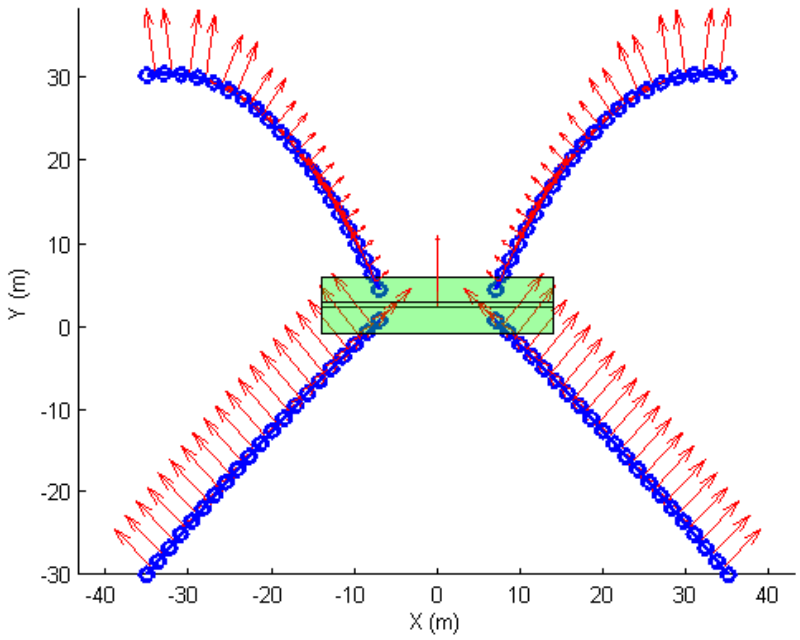


Figure 5.4 Third solution and segment drag forces

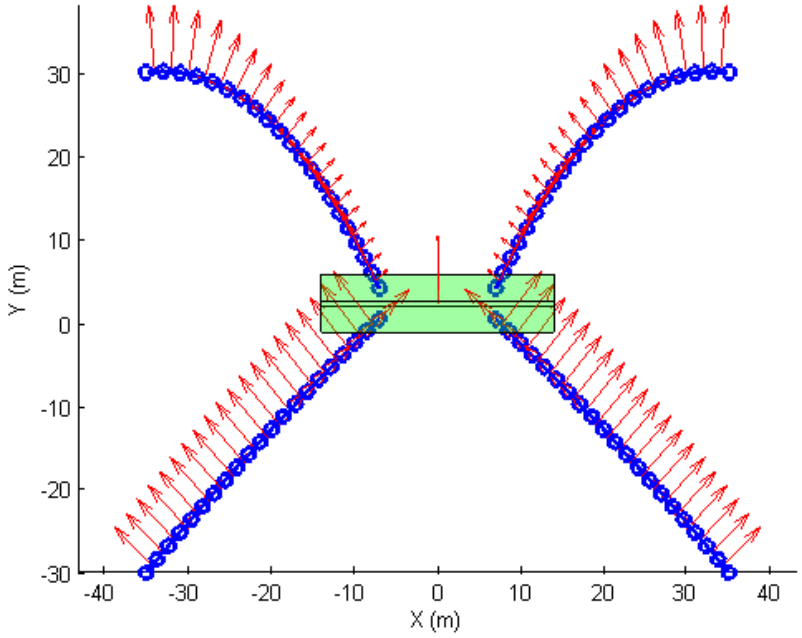


Figure 5.5 Fourth solution and segment drag forces

Figure 5.6 through Figure 5.9 show a side view of each iterative solution. Table 5.5 lists the body's COM position and Euler angles for each iterative solution.

Table 5.5 Body's COM position and Euler angles, $|\bar{u}_f| = 4\,m/s\,, \theta = 0°$

Solution	$x\ (m)$	$y\ (m)$	$z\ (m)$	$\alpha\ (rad)$	$\beta\ (rad)$	$\gamma\ (rad)$
1	10^{-8}	4.552	10.67	0.2732	10^{-7}	10^{-7}
2	10^{-6}	2.915	14.98	0.3482	10^{-7}	10^{-7}
3	10^{-6}	2.410	16.02	0.3747	10^{-7}	10^{-7}
4	10^{-5}	2.263	16.30	0.3809	10^{-6}	10^{-6}

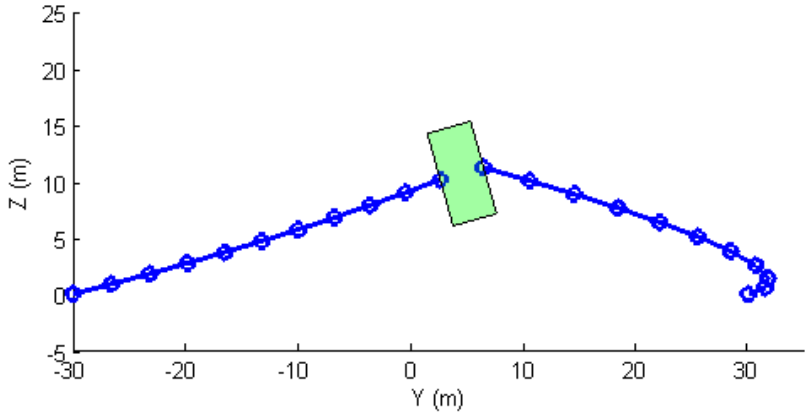


Figure 5.6 First solution

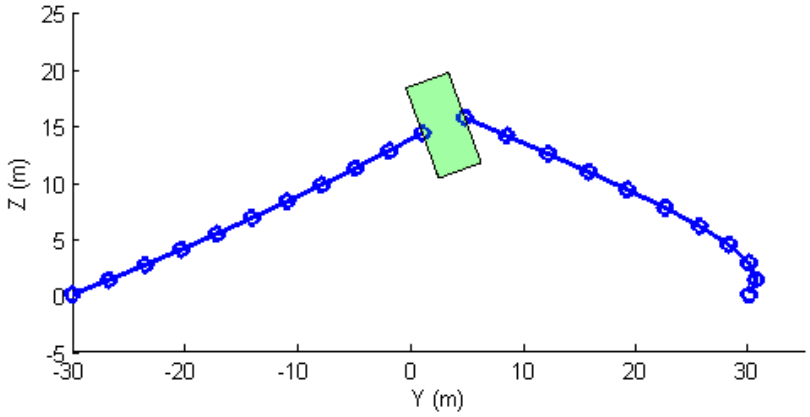


Figure 5.7 Second solution

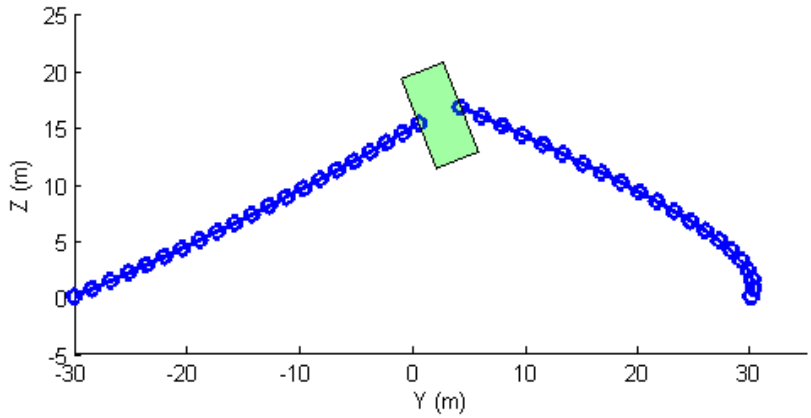


Figure 5.8 Third solution

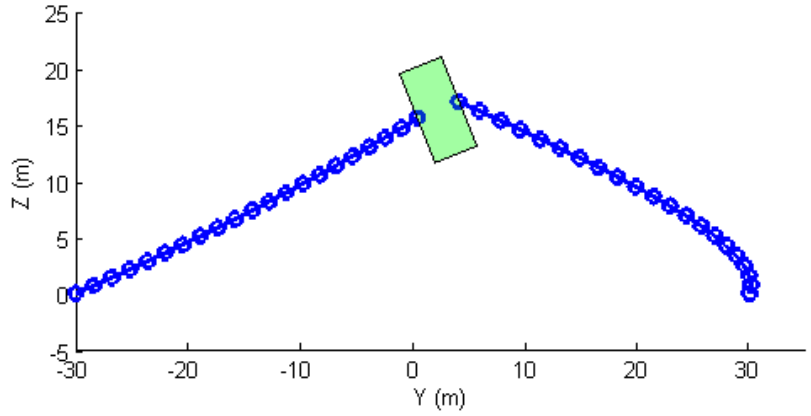


Figure 5.9 Fourth solution

Figure 5.10 shows a 3D view of the fourth iterative solution.

These test solutions show the body is rotated significantly toward the oncoming stream flow. This rotation can be important to the power performance of a hydrokinetic turbine if the turbine is orientated in line with the body and, therefore, off-axis to the incoming flow, or if part of turbine's support structure now blocks part of the fluid flow over the turbine. The rotation is caused by torques applied to the body from the tension forces in the mooring lines.

Considering a 2D model of the body and tensions forces from the mooring lines (Figure 5.11), a rotation will occur if one tension force is larger than the other and/or at

different angles relative to the body. In this test the mooring lines upstream are stretched more and have a greater tension force ($\overline{T}_1$) than the mooring lines downstream ($\overline{T}_2$).

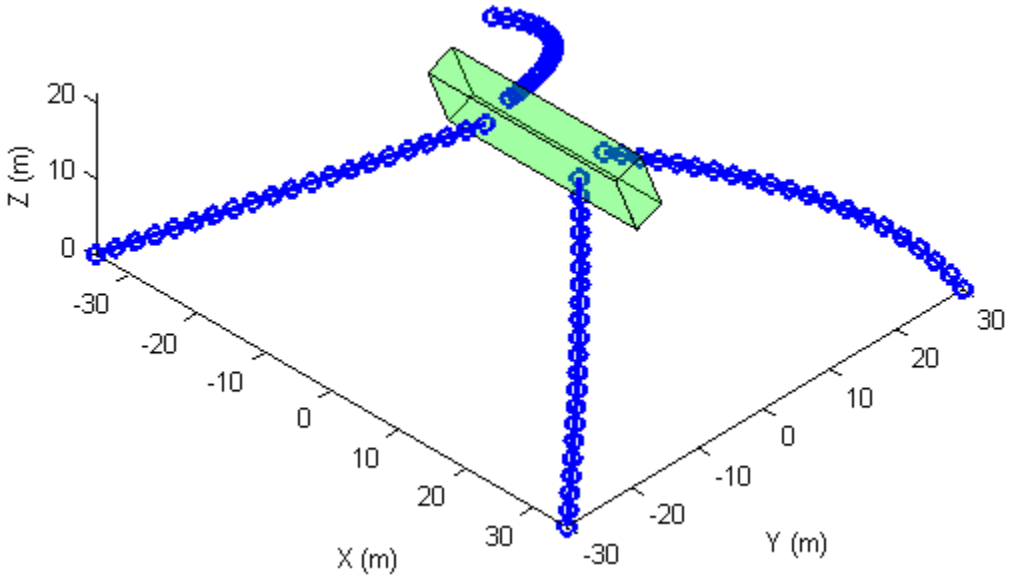


Figure 5.10 Perspective of fourth solution

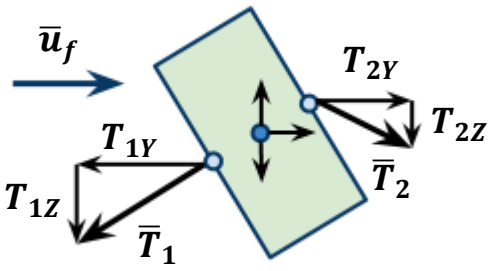


Figure 5.11 2D model of tension forces on the body

Table 5.6 lists the components and magnitudes of the tension forces in the mooring line segments attached to the body from the fourth iterative solution.

Table 5.6 Mooring line segment tensions at body (N)

Line #	**1**	**2**	**3**	**4**
T_X	55,225	6,091.5	-6,080.9	-55,226
T_Y	-53,783	12,603	12,581	-53,784
T_Z	-32,305	-5,705.8	-5,696.0	-32,305
Magnitude	83,582	15,116	15,090	83,584

For the 2D model of the line tensions on the body (Figure 5.11), only forces in the $\boldsymbol{Y-Z}$ plane are considered because the solution is nearly symmetric about the $\boldsymbol{Y}$ axis. The average tension in the upstream mooring lines (lines # 1 and 4) were taken to model the $\overline{\boldsymbol{T}}_{\boldsymbol{1}}$ tension force and the average tension in the downstream mooring lines (lines # 2 and 3) to model the $\overline{\boldsymbol{T}}_{\boldsymbol{2}}$ tension. Table 5.7 lists the components of the $\overline{\boldsymbol{T}}_{\boldsymbol{1}}$ and $\overline{\boldsymbol{T}}_{\boldsymbol{2}}$ forces.

Table 5.7 Tension forces on body for 2D model (N)

Force	$\overline{\boldsymbol{T}}_{\boldsymbol{1}}$	$\overline{\boldsymbol{T}}_{\boldsymbol{2}}$
Y-component	-53,784	12,592
Z-component	-32,305	-5,700.9

Taking the moments about the body's COM can show why the body will rotate towards the oncoming stream flow if $\overline{\boldsymbol{T}}_{\boldsymbol{1}}$ is greater than $\overline{\boldsymbol{T}}_{\boldsymbol{2}}$**.** The weight, buoyancy and drag force on the body act at the body's COM in this test and do not cause a torque on the body. Because the attachment points of the mooring lines are symmetric in the body-fixed frame, the two tension forces have the same length of moment arm $\boldsymbol{r}$ (for this test $\boldsymbol{r = 2\ m}$). When the body is rotated to equilibrium, the $\boldsymbol{T_{2Y}}$, $\boldsymbol{T_{2Z}}$, and $\boldsymbol{T_{1Y}}$ components of force create clockwise torques that balance out the large counterclockwise torque of the $\boldsymbol{T_{1Z}}$ force. For the 2D model only the $\boldsymbol{\alpha}$ rotation is considered to model the rotation of the body, with $\boldsymbol{\alpha = 0.3809\ rad}$ from the fourth iterative solution, measured positive

counterclockwise from the $\boldsymbol{Y}$ axis. The sum of moments about the body's center of mass for the 2D model is

$$\circlearrowleft \sum \boldsymbol{M} = -|\boldsymbol{T_{2Y}}|\boldsymbol{r}\sin\boldsymbol{\alpha} - |\boldsymbol{T_{2Z}}|\boldsymbol{r}\cos\boldsymbol{\alpha} - |\boldsymbol{T_{1Y}}|\boldsymbol{r}\sin\boldsymbol{\alpha} + |\boldsymbol{T_{1Z}}|\boldsymbol{r}\cos\boldsymbol{\alpha} = \boldsymbol{43.423\ Nm} \tag{5.1}$$

The net torque on the body from Equation 5.1 is not zero but is several orders of magnitude smaller than the tension forces on the body. The difference of the net torque from zero is due to the approximations made for the 2D model and that the solution has a small rotation of the body about the $\boldsymbol{Z}$ axis and is not perfectly symmetric about the $\boldsymbol{Y}$ axis.

5.2 Varying Fluid Velocity Angle

A series of test problems were run with the same set up as the previous section but varying the angle the fluid velocity makes with the $\boldsymbol{Y}$ axis (Figure 5.1). The angle $\boldsymbol{\theta}$ was varied from 0° to 90° in increments of 10° with a constant speed of $\left|\overline{\boldsymbol{u}}_{\boldsymbol{f}}\right| = \boldsymbol{4\ m/s}$. These correspond to cases for off-axis flow such as might occur in an elliptical tidal current. Three iterative solutions were used to solve the problem, using 10 segments per mooring line for the first two solutions and 20 segments for the third. Three iterative solutions were run instead of four as with the previous test only for convenience to take less time to run the tests. For comparison the system was solved with zero fluid velocity and three iterative solutions. The COM displacement is calculated as the magnitude of displacement of the body's center of mass with fluid velocity compared to the solution with zero fluid velocity. Figures Figure 5.12 and Figure 5.13 plot the body's position and Euler angles versus the fluid velocity angle respectively.

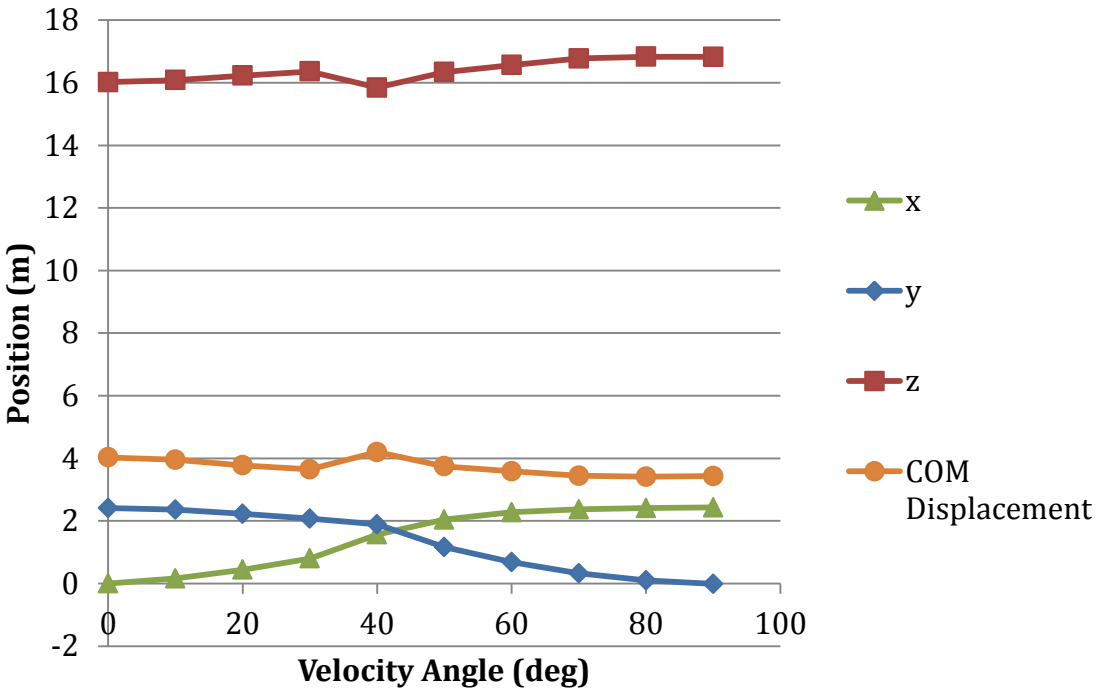


Figure 5.12 Body's position and displacement versus velocity angle

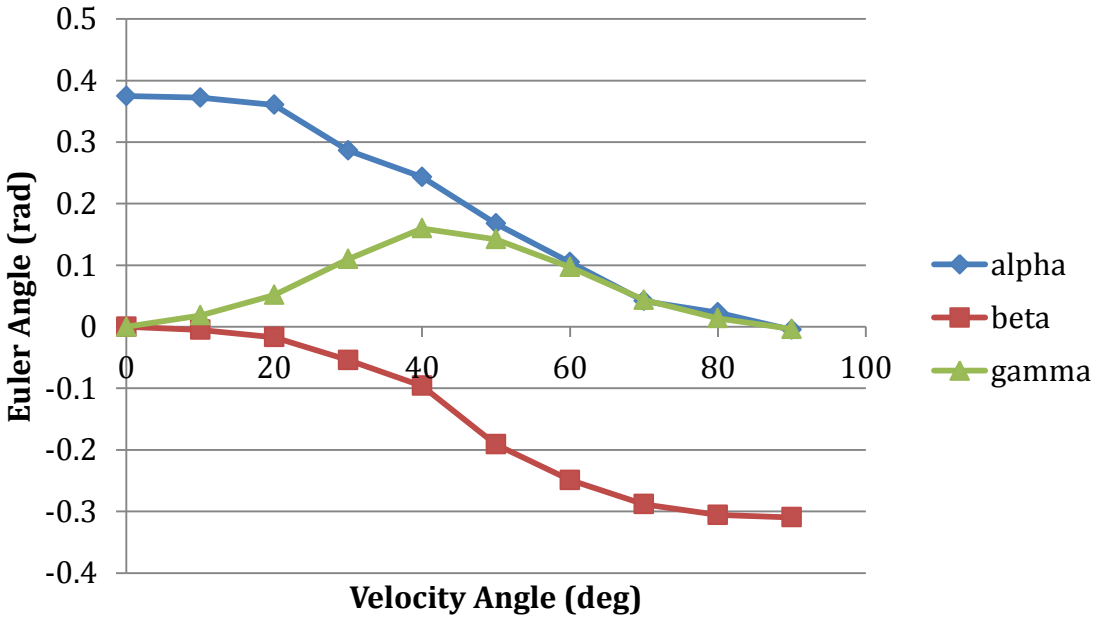


Figure 5.13 Body's Euler angles versus velocity angle

These solutions show the body consistently rotates toward the oncoming fluid flow as the fluid velocity angle changes. The body's COM displacement and depth stay fairly constant, with their difference between tests within one meter. Table 5.8 lists the body's COM position and Euler angles for zero fluid velocity and for each angle with fluid velocity. Figure 5.14 shows a front view of the third iterative solution when $\boldsymbol{\theta = 90^{\circ}}$.

Table 5.8 Body's COM position and Euler angles for varying fluid velocity angles

$\boldsymbol{\theta}$ $(\boldsymbol{deg})$	$\boldsymbol{x}$ $(\boldsymbol{m})$	$\boldsymbol{y}$ $(\boldsymbol{m})$	$\boldsymbol{z}$ $(\boldsymbol{m})$	$\boldsymbol{\alpha}$ $(\boldsymbol{rad})$	$\boldsymbol{\beta}$ $(\boldsymbol{rad})$	$\boldsymbol{\gamma}$ $(\boldsymbol{rad})$	$\boldsymbol{COM\ Displ.}$ $(\boldsymbol{m})$
0 Vel.	10^{-8}	10^{-6}	19.25	10^{-6}	10^{-8}	10^{-7}	---
0	10^{-6}	2.410	16.02	0.3747	10^{-7}	10^{-7}	4.032
10	0.1607	2.360	16.08	0.3722	-0.00499	0.01849	3.951
20*	0.4434	2.223	16.23	0.3604	-0.01699	0.05161	3.776
30*	0.7982	2.073	16.36	0.2864	-0.05405	0.1099	3.645
40*	1.566	1.886	15.84	0.2432	-0.09602	0.1598	4.197
50*	2.041	1.162	16.34	0.1675	-0.1912	0.1421	3.741
60*	2.276	0.6803	16.56	0.1046	-0.2491	0.09694	3.584
70*	2.371	0.3224	16.78	0.04188	-0.2882	0.04369	3.440
80	2.408	0.09787	16.83	0.02296	-0.3057	0.01371	3.413
90*	2.430	-0.01124	16.82	-0.00494	-0.3097	-0.00371	3.432

* Some or all iteration solutions did not converge

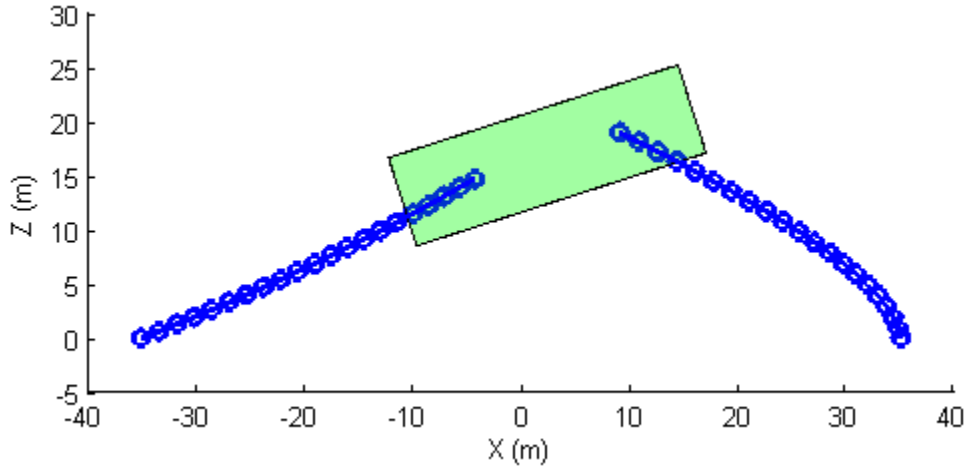


Figure 5.14 Front view of equilibrium solution, $|\bar{u}_f| = 4\,m/s, \theta = 90°$

For many of these tests the Matlab minimization solver *fmincon* did not converge to a minimum. The solver will stop prematurely if it reaches a limit on the number of iterations or function evaluations before minimizing the objective function within a specified tolerance. For iterations that do not converge the solver displayed a maximum constraint value that can vary by several orders of magnitude, sometimes increasing to the order of 10^0, indicating that not all the constraints are satisfied at that solver iteration. Some of the constraints may need to be relaxed in such cases to converge to a minimum.

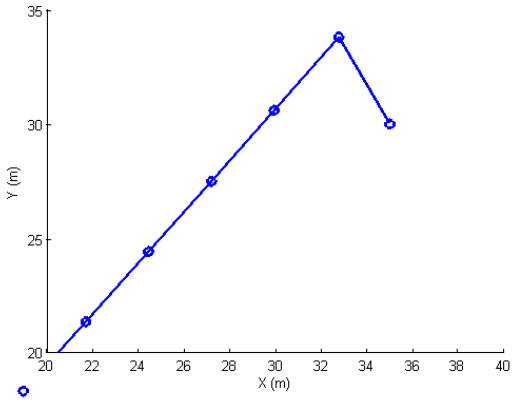
It should be noted that the tolerances specified for the *fmincon* solver of 10^{-6} (default values) is very tight for a problem of this scale with displacements measured in meters. Some constraints such as the ground constraint may realistically allow softer constraints than others because the moorings lines and body may sink into the ground, but the solver only allows the same tolerance to be applied to all constraints.

A possible cause for the solver not converging or constraints being satisfied may be the mooring lines being unable to take the shape needed to minimize the potential energy. The shape of the mooring lines is dependent on its number and size of line segments. Bends in the mooring line are only approximated as straight lines by the line segments.

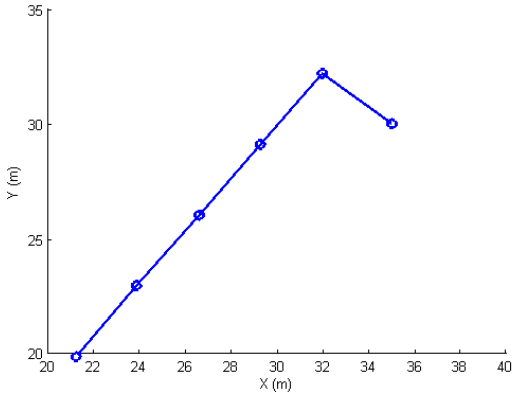
A minimum potential energy may require the line segments or nodes to be in a position they cannot be because of constraints such as the slackness or ground constraints.

Figure 5.15 shows how the shape of the mooring line furthest downstream (line #2) changes between each solution iteration for the test where $\left|\bar{\boldsymbol{u}}_{\boldsymbol{f}}\right| = \boldsymbol{4\,m/s}$, $\boldsymbol{\theta} = \boldsymbol{40}°$. For this test the solver did not converge for each iterative solution.

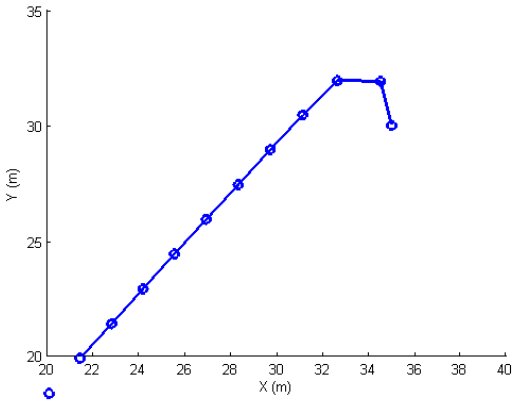
Smaller line segments may be required to accurately describe the bend of the mooring line near the ground attachment point to find a minimum in the potential energy. Another condition that could require smaller line segments includes a case where some of the mooring line segments touch the ground. A minimum in the potential energy may require only a fraction of a segment to touch ground, but the solver cannot reach this minimum because either all or none of the line segment will touch the ground. A possible fix to these problems is to use mesh refinement and increase the number of line segments near the ground or bends and is to be considered in the future work of this research.



(a) First solution



(b) Second solution



(c) Third solution

Figure 5.15 Top view of mooring line #2 near its ground attachment point $|\bar{u}_f| = 4\,m/s\,, \theta = 40°$

5.3 Sensitivity to Variation in the Location of the COM and COB

The location of the body's center of mass (COM) or center of buoyancy (COB) relative to the body-fixed frame may not be known exactly or can only be estimated with a low tolerance. The system may not behave as expected if the COM or COB is not located where they were expected to be, or the system may behave more desirably if the COM or COB are moved to different locations. To observe the system's sensitivity to these variations multiple solutions were compared varying the locations of the COM and COB.

The models used in these test problems assume the body's COM is located at its geometric center and the body mooring line attachment points are symmetric about the body-fixed frame's origin. A variation in the body's COM results in the attachment points locations to vary in the body-fixed frame relative to the actual COM. The difference of the body's actual COM compared to the expected COM in the body-fixed frame is represented as $\Delta\boldsymbol{x}, \Delta\boldsymbol{y}, \Delta\boldsymbol{z}$, in the body-fixed $\boldsymbol{x}'$, $\boldsymbol{y}'$, $\boldsymbol{z}'$ directions respectively. The body mooring line attachment points relative to the actual COM in the body-fixed frame become

$$\bar{\boldsymbol{p}}'_{\boldsymbol{i,B}} = \left[\boldsymbol{p_{i,B,x'}} - \Delta\boldsymbol{x}, \boldsymbol{p_{i,B,y'}} - \Delta\boldsymbol{y}, \boldsymbol{p_{i,B,z'}} - \Delta\boldsymbol{z}\right]^{\boldsymbol{T}} \tag{5.2}$$

where $\boldsymbol{p_{i,B,x'}}$, etc. are the components of the attachment points relative to the body's expected COM or geometric center.

Variations on the COM and COB were chosen to be 10% of the body's dimensions. The body dimension used were a width of 28 m ($\boldsymbol{x}'$ direction), depth of 4 m ($\boldsymbol{y}'$ direction) and height of 8.5 m ($\boldsymbol{z}'$ direction), giving the variation magnitudes of $\Delta\boldsymbol{x} = \boldsymbol{2.8\ m}$, $\Delta\boldsymbol{y} = \boldsymbol{0.4\ m}$ and $\Delta\boldsymbol{z} = \boldsymbol{0.85\ m}$.

The test problems have the same setup as section 5.1 but with zero fluid velocity. Three iterative solutions were run, using 10 segments per mooring line for the first two solutions and 20 segments for the third.

The COM and COB locations were varied in the positive and negative directions along one axis at a time keeping all other variations set to zero. When the COM was varied the COB remained fixed to the COM ($\bar{\boldsymbol{c}}'_{\boldsymbol{b}} = [\boldsymbol{0} \quad \boldsymbol{0} \quad \boldsymbol{0}]^{\boldsymbol{T}}$). When the COB was varied the COM remained at the body's geometric center (the mooring line body attachment points did not change) and the COB location in the body-fixed frame changed according to

$$\bar{\boldsymbol{c}}'_{\boldsymbol{b}} = [\Delta\boldsymbol{x} \quad \Delta\boldsymbol{y} \quad \Delta\boldsymbol{z}]^{\boldsymbol{T}} \tag{5.3}$$

Figures Figure 5.16 and Figure 5.17 plot the change in the body's position and change in Euler angles dues to variations in the body's COM and COB respectively compared to the solution with no variations.

The body's COM position moves in the same direction as its variation. The body rotates when there is a COM $\Delta \boldsymbol{x}$ and $\Delta \boldsymbol{y}$ variance, with the side the COM is moved toward rising higher than the opposite end. Figure 5.18 plots the equilibrium solution for a variation of the COM in the positive $\boldsymbol{x}'$ direction. The body's COM equilibrium moved about 4 m in the positive $\boldsymbol{X}$ direction and rotated about $-0.25\ rad$ in the $\boldsymbol{\beta}$ direction.

A variation in the COB causes a smaller change in the body's translational position but a larger change in the body's rotation compared to a variation in the COM. Figure 5.19 plots the equilibrium solution for a variation of the COB in the positive $\boldsymbol{x}'$ direction. The body's COM equilibrium moved about 1.8 m in the positive $\boldsymbol{X}$ direction and rotated about $-0.4\ rad$ in the $\boldsymbol{\beta}$ direction. The body's $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ rotations are almost doubled compared to the rotations from the variance in the COM. The larger rotations are caused by the additional torque on the body when the COB is offset laterally from the COM.

Because of the symmetry of the system the results are symmetric for variances in the positive and negative $\boldsymbol{x}'$ and $\boldsymbol{y}'$ directions, though the results are not exactly symmetric because of small numerical differences in the solutions. Variances in the $\boldsymbol{z}'$ direction do not cause rotations as expected and did not change the results of the overall system.

It should be noted that a location of a submerged body's COB below the COM is not a stable equilibrium position for a body by itself. The COB would be located above the COM in stable equilibrium positions as any lateral offset between the COM and COB would cause a torque on the body in a direction to orient itself in the stable position. For this test, the solution did converge to equilibrium when the COB was located below the COM. The existence of the mooring lines would physically prevent the body from rotating upside down and could also prevent the solver from finding a minimum in the

potential energy such that the body had rotated upside down because a local minimum could be found leaving the COB below the COM.

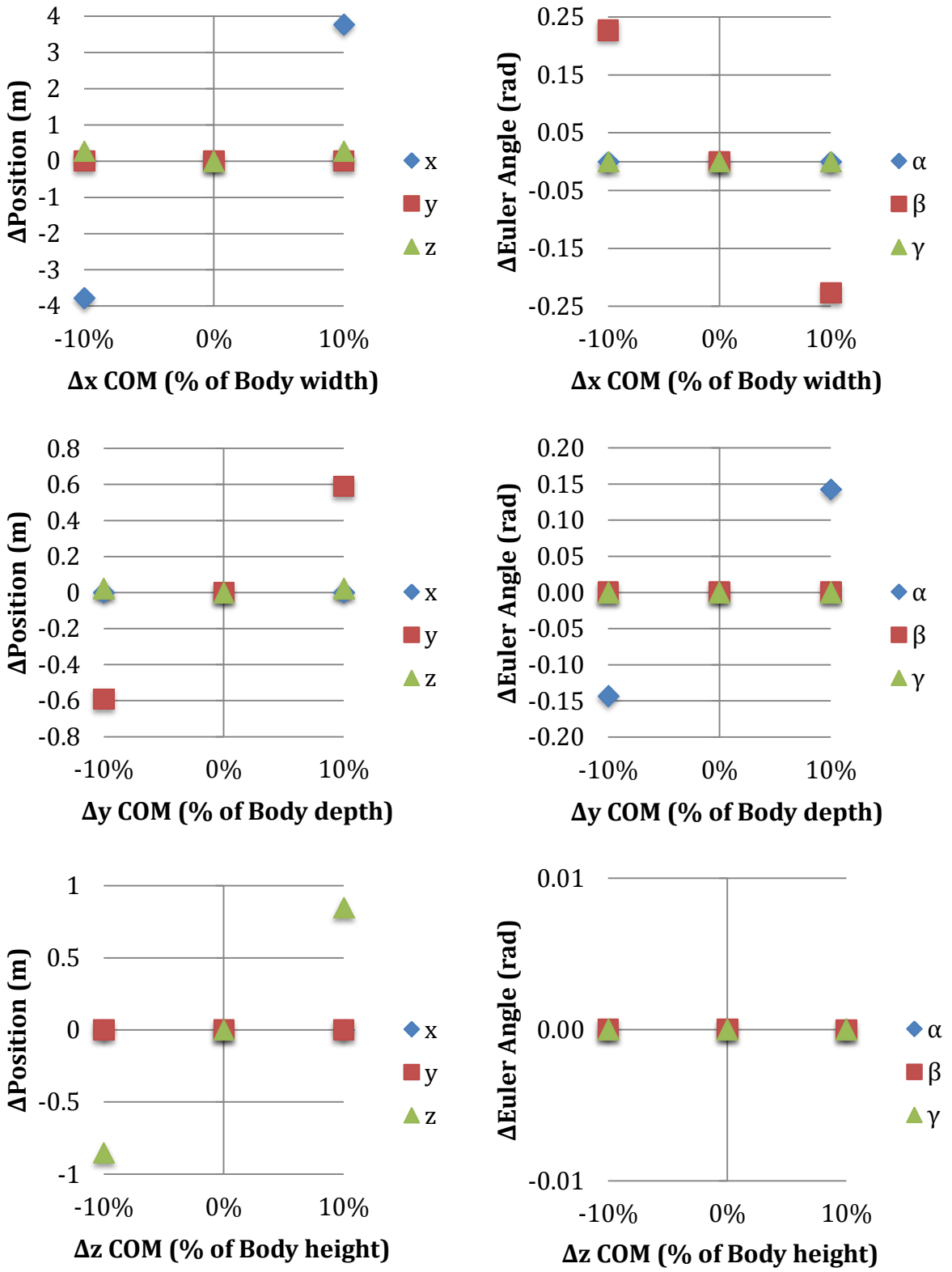


Figure 5.16 Change in body's position and Euler angles due to variation of body's COM

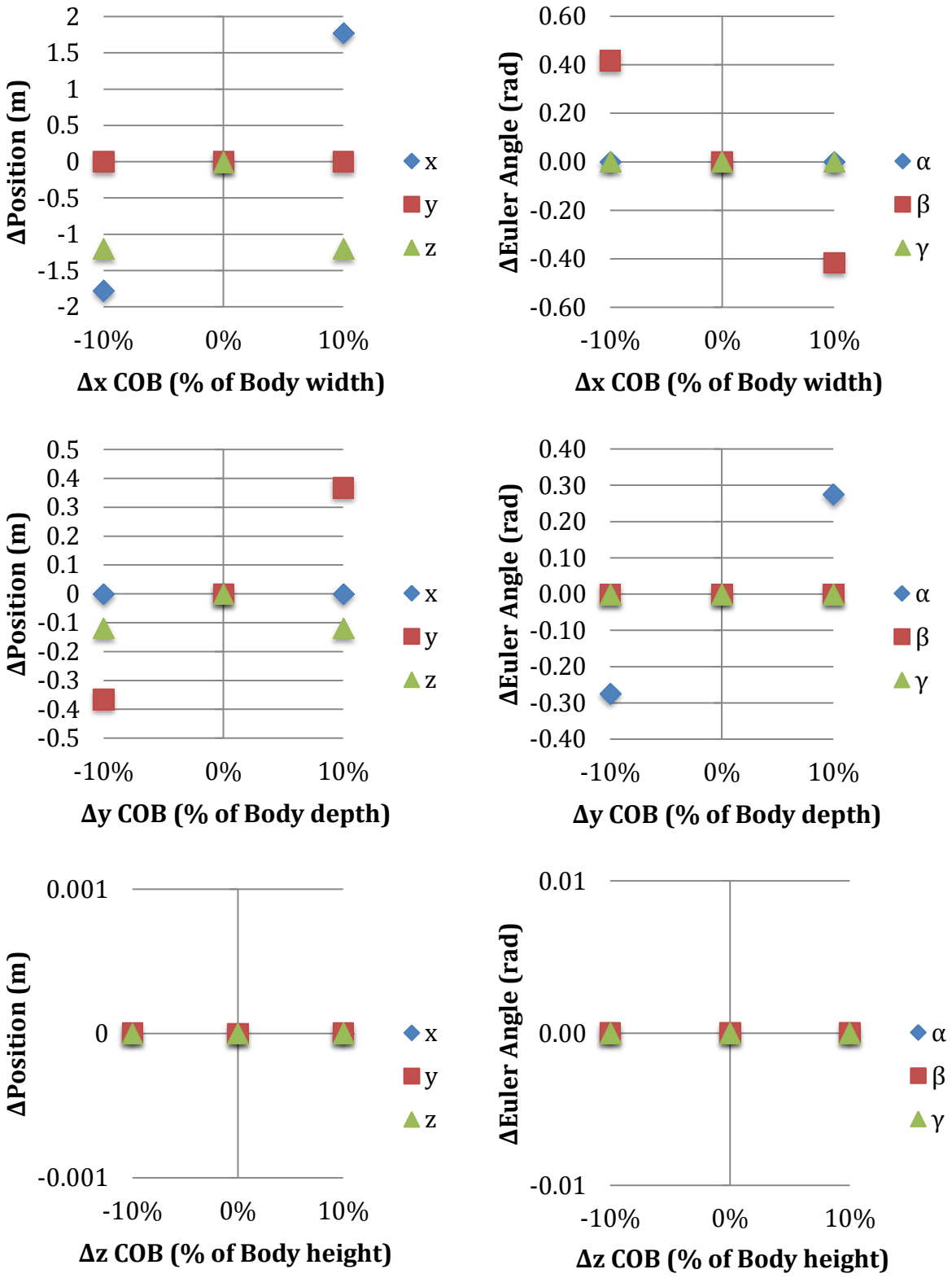


Figure 5.17 Change in body's position and Euler angles due to variation of body's COB

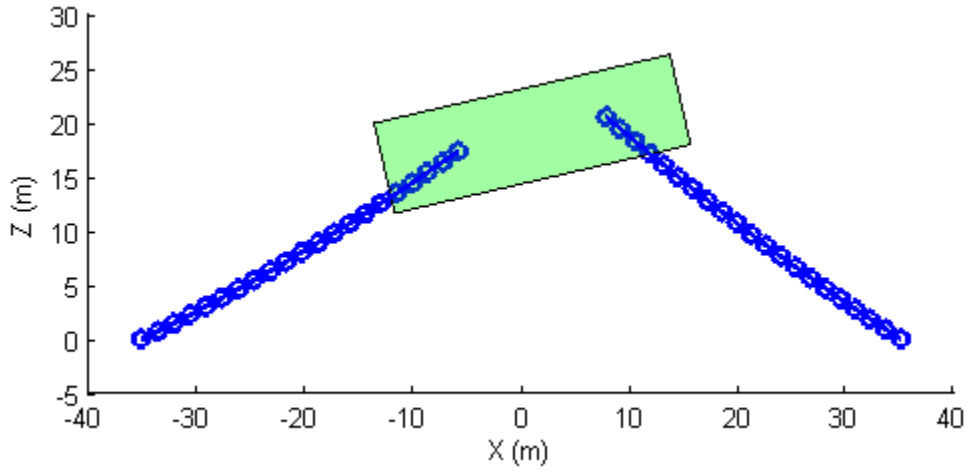


Figure 5.18 Equilibrium position due to $\Delta \boldsymbol{x}$ variation in body's COM

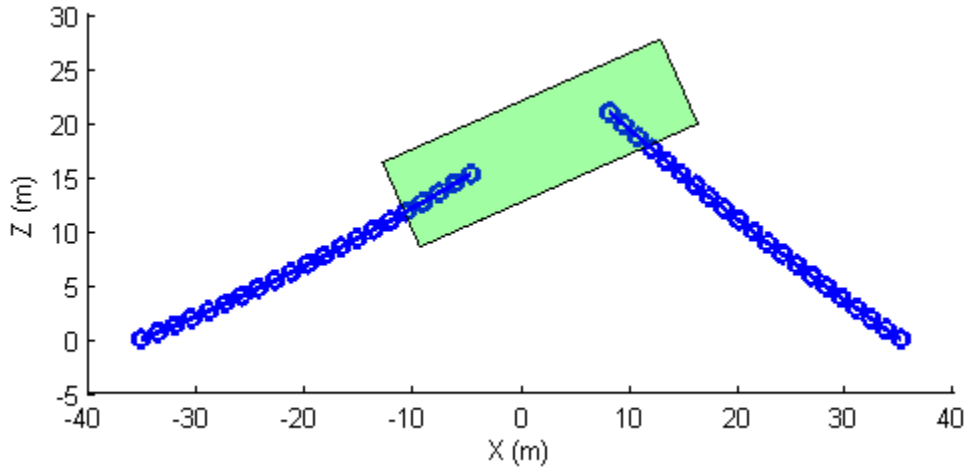


Figure 5.19 Equilibrium position due to $\Delta \boldsymbol{x}$ variation in body's COB

6 Matlab Implementation and Solver

All the code necessary to model and solve these problems was written in Matlab with the intent that a user can easily modify the code to model and solve their problem. All the Matlab script files and functions written fall under the following categories:

- Script files to define the properties and set up everything necessary to solve a problem
- Functions to calculate the potential energies in the system
- Function to set up the constraints and options for the built-in Matlab function *fmincon* to minimize the potential energy to solve for equilibrium
- Functions to plot the solutions
- Various functions for calculations such as defining the fluid velocity profile and calculating the Euler angle rotation transformation matrix

The Matlab function *fmincon*, which is part of the Matlab Optimization Toolbox, is used to minimize the potential energy. *Fmincon* finds the minimum of a constrained nonlinear multivariable function, also known as nonlinear programming. The current version of code for this research uses *fmincon's* active-set algorithm that uses a sequential quadratic programming method and computes the Hessian using finite differences.

The *fmincon* function allows various types of constraints on the variables to be defined. For the type of systems considered in these problems the following constraints were defined:

- **Lower Bound Constraints:** To model the ground constraint, the lower bound of the $\boldsymbol{Z}$ coordinate of every line node and the body is set to zero, so that $z \geq 0$. The lower bound for every other variable is set to $-\infty$ so that it is not constrained. To model other seabed topographies, the elevation of the seabed can be defined as a function of $\boldsymbol{X}$ and $\boldsymbol{Y}$.

- **Linear Equality Constraints**: To model the ground attachment point or anchor of the mooring line, the last node of the mooring line $(\bar{\boldsymbol{p}}_{i,N})$ is set to be equal to its ground attachment point $(\bar{\boldsymbol{p}}_{i,G})$.
- **Non-linear Inequality Constraints:** The slack constraint for the mooring lines such that they can only stretch and not compress requires that a line segment's instantaneous length $(|\bar{\boldsymbol{p}}_{i,j+1} - \bar{\boldsymbol{p}}_{i,j}|)$ is greater than or equal to its unstretched length $(\boldsymbol{L}_{i,j})$. For the solver this must written in the form $-|\bar{\boldsymbol{p}}_{i,j+1} - \bar{\boldsymbol{p}}_{i,j}| + \boldsymbol{L}_{i,j} \leq \boldsymbol{0}$.
- **Non-linear Equality Constrains:** The coordinates of the first node of a mooring line $(\bar{\boldsymbol{p}}_{i,0})$ are constrained to be fixed to its body attachment point $(\bar{\boldsymbol{p}}'_{i,B})$. For the solver this written in the form $\bar{\boldsymbol{p}}_{i,0} - \bar{\boldsymbol{c}} - \boldsymbol{T}\bar{\boldsymbol{p}}'_{i,B} = \boldsymbol{0}$.

For more information about *fmincon*, the optimization algorithms used and how to use Matlab's optimization functions see Matlab's Optimization Toolbox help documentation which is also available on the MathWorks website (13).

All the Matlab script files to solve these benchmark and test problems are included in the Appendices. A script file that sets up the general test problem with multiple mooring lines is included with an explanation of how to setup the code in Appendix 1. The setups for the Elastic Catenary and Rigid Body Rotation benchmarks are included in their own script files. The user can modify these setup files to define the system's properties as they wish. Only the setup script files need to be run to solve the problem and display the results.

Object oriented programming is used to keep track of potential energies associated with the body and mooring lines. This is done so the way the body and mooring lines are modeled can be easily modified. All of the code is commented throughout to describe all the variables, calculations, functions and their inputs and outputs.

The code was written using Matlab version R2009a. The user needs the Optimization Toolbox installed with Matlab in order to run the code.

7 Future Work

The future work involving this research can be divided into two categories: improvements to the current work, and further or additional research. Improvements to the current work can include changes to the code and how the system is modeled. Some improvements include:

- Mesh refinement – The number of line segments in the mooring lines is locally increased in problems areas such where the mooring line touches the ground or around kinks so the solver may converge or provide a more accurate solution.
- Split mooring lines and multiple bodies – The mooring lines can be attached to other mooring lines or a different body instead of having one end attached to the ground and the other end attached to the body for each mooring line.

Additional forces or the way they are modeled can be added to the current code by adding the appropriate properties and potential energy calculation functions to the body and mooring line classes.

Further research includes work that can use the equilibrium solutions as a basis or work to analyze different properties of the system. Some of the further research areas include:

- Dynamic analysis – Behavior of the system over time and can include time-varying forces such as the effect of waves or currents.
- Vibration analysis – Vibration of the body and mooring lines from fluid or turbine forces (e.g., vortex induced vibration).
- Control issues – Possible methods to control the body's position and orientation, and optimal mooring configurations that allow better control of the body.

References

1. **Ocean Renewable Power Company, LLC.** ORPC POWER SYSTEMS. [Online] 2011. http://www.oceanrenewablepower.com/orpcpowersystems.htm.

2. **Meirovitch, Leonard.** *Methods of Analytical Dynamics.* s.l. : McGraw-Hill, 1970.

3. **Irvine, H. M.** *Cable Structures.* Cambridge : MIT Press, 1981.

4. *Large angular motions of tethered surface buoys.* **Leonard, J. W., Idris, K. and Yim, S. C. S.** 12, s.l. : Ocean Engineering, 2000, Vol. 27.

5. *The Mechanics of Highly-Extensible Cables.* **Tjavaras, A. A., et al., et al.** 4, s.l. : Journal of Sound and Vibration, 1998, Vol. 213.

6. *Dynamic analysis of three-dimensional marine cables.* **Huang, Shan.** 6, s.l. : Ocean Engineering, 1994, Vol. 21.

7. *Three-dimensional coupled dynamics of a buoy and multiple mooring lines: formulation and algorithm.* **W. Raman-Nair, R. E. Baddour.** s.l. : Oxford University Press, 2002.

8. **Kane, Thomas R. and Levinson, David A.** *Dynamics, Theory and Applications.* s.l. : McGraw-Hill, 1985.

9. **Gobat, Jason I and Grosenbaugh, Mark A.** *WHOI Cable v2.0: Time Domain Numerical Simulation of Moored.* s.l. : Woods Hole Oceanographic Institution, 2000.

10. **Gobat, Jason I., Grosenbaugh, Mark A. and Triantafyllou, Michael S.** *WHOI cable : time domain numerical simulation of moored and towed oceanographic systems.* s.l. : Woods Hole Oceanographic Institution, 1997.

11. **Gobat, Jason I.** *The dynamics of geometrically compliant mooring systems.* Woods Hole, MA : PhD thesis, Massachusetts Institute of Technology and Woods Hole Oceanographic Institution Joint Program, 2000.

12. **Tjavaras, Athanassios A.** *Dynamics of highly extensible cables.* Cambridge, MA : PhD thesis, Massachusetts Institute of Technology, 1996.

13. *MathWorks.* [Online] The MathWorks, Inc., 2011. http://www.mathworks.com/help/toolbox/optim/ug/fmincon.html.

Appendix 1 – Example of a Body and Multiple Mooring Line System Setup Script

This example explains how to setup a Matlab script file to model a general body and multiple mooring line system to solve for equilibrium. The complete script file referenced in this example (Sys_test.m) is included in Appendix 2, which the user can modify to model their specific system.

Lines 9 through 30 in the script file define the properties necessary to create a body object. Each property of the body is assigned a variable. Above each variable is a comment describing what the variable represents, units of the variable if applicable in parenthesis and the format of the variable, for example: the variable must be a column vector of the form [CDx'; CDy'; CDz'].

This version of code requires all of these properties to be defined before creating a body object. The user can use a different system of units but they must make sure all the other properties and all calculations using these properties use a consistent set of units. For this example the user only needs to define a value for each variable.

The body's COM initial position is defined on line 28. Note that not any initial position can be chosen because the solver may not start if inappropriate initial positions are chosen. As stated in the comments above line 28 the initial length of the mooring lines must be greater than or equal to its unstretched length. The user should choose an initial body position that slightly stretches each mooring line.

All the body property variables are passed to the **newBody** function in line 33 to create a new body object labeled **Body** in this example. This variable **Body** is what is passed to other functions to perform calculations that require the body's properties.

Lines 37 through 71 define the properties necessary to create a line object for each mooring line in the system. Above each variable is a comment describing what the variable represents in the same manner as the body properties.

Because this example uses the same property values for all mooring lines only one value was assigned for the linear density (line 41), unstretched length (line 44) and diameter (line 51) variables and were not defined as arrays.

To assign different values of the same property to each mooring line the property is defined as an array (column or row vector) with each element containing the value for a single mooring line. For example, the stiffness of the mooring lines is defined as a row vector (line 53) with four numbers representing the stiffness value for each of the four mooring lines. This example used the same value for all mooring lines but the user can specify different values for each mooring line. The length of the arrays must be the same as the number of mooring lines in the system (in this example the length is 4).

Lines 56 through 59 define the coordinates of the mooring lines' body attachment points as column vectors. A separate variable must be defined for each mooring line. Line 62 places all the body attachment point column vectors into a matrix to be used later in a for loop to create a line object for each mooring line.

Lines 65 through 68 define the coordinates of the mooring lines' ground attachment points as column vectors and line 71 places all the ground attachment point column vectors into a matrix.

Line 71 is the last line the in this example code that the user needs to modify in order to set up a problem just to solve for equilibrium of the system. Constants such as gravity, fluid density and the mooring line coefficients of drag are defined in the **newBody.m** and **newLine.m** class files and need to be changed by the user as necessary. The fluid velocity profile needs to be defined by the user by modifying the **FluidVelocity.m** file.

If the user wants to plot the drag forces on the system they can uncomment the lines that begin with the PlotDrag3D functions (Ex. Uncomment line 115 to plot the drag forces from the first solution).

Line 74 sets up the initial displacement column vector by placing the body's initial COM coordinates in rows 1 through 3 and the initial Euler angles in rows 4 through 6.

Lines 76 through 107 define straight lines for each mooring line from its body attachment point to its ground attachment point. The lines are divided into equal length segments and the coordinates of each node are successively added to the initial displacement vector.

Line 103 creates a new line object and adds them to a line object array called **Lines** within a for loop that uses the variable **i** that goes from one to the number of mooring lines in the system. The variable **i** is used as an index to keep track of the variables associated for each mooring line. For example, an array was created for the stiffness values (line 53) labeled **K**. When each line object is created the i^{th} element of the **K** array is passed to the **newLine** function with the syntax **K(i)**.

The index of the mooring line properties must be kept consistent between properties. For example, the first mooring line has its body attachment point as the first column in the body attachment point matrix and its stiffness value as the first element in the **K** array. The second mooring line has its body attachment point as the second column in the body attachment point matrix and its stiffness value as the second element in the **K** array, and so on.

Line 110 solves the system by passing the body object (**Body**), line object array (**Lines**) and initial displacement vector to the equilibrium solver function **EquilSolver**. This function returns the solution of displacement variables in column vector that is assigned to the variable **qf**.

Lines 112 through 115 plot the solution. The user can uncomment line 115 to plot the drag forces on the system as red arrows on the plot of the solution. Lines 117 through 123 solves and plots the system a second time using the **SysUpdate** function (line 118) to update the drag forces on the system. Lines 125 through 131 solves and plots the system a third time and doubles the number of line segments by using the **SysRefine**

function (line 126). The user can uncomment lines 134 through 139 to update the system drag forces and solve and plot the solution for a fourth time.

Appendix 2 – Sys_test.m – Basic Body and Mooring Line Test Setup

```
clear
close all
clc

% Example test problem with multiple mooring lines

% Specify Body properties

% Mass of body (kg)
M = 5000;
% Volume of body (m^3)
V = 10;
% Center of buoyancy relative to center of mass in body-fixed frame (m),
% column vector [cbx'; cby'; cbz']
cB = [0; 0; 0];
% Coefficients of Drag in the body-fixed frame [CDx'; CDy'; CDz']
CD = [0.5; 0.5; 0.5];
% Effective surface areas of the Body in the body-fixed frame (m^2)
% [Ax'; Ay'; Az']
AB = [16; 24; 11];
% Applied forces to the Body in global frame (N) [FaX; FaY; FaZ]
Fa = [0; 0; 0];
% Applied torques to the Body in global frame (N*m) [TaX; TaY; TaZ]
Ta = [0; 0; 0];
% Initial position of center of mass in global frame (m) c = [x; y; z]
% Must be chosen such that the initial elongations of the mooring lines
% are stretched and not shorter than the unstretched lengths of the lines.
c_i = [0; 0; 20];
% Initial xyz Euler angles (radians) angles_i = [alpha; beta; gamma]
angles_i = [0; 0; 0];

% Creates new Body object
Body = newBody(M,V,cB,CD,AB,Fa,Ta,c_i,angles_i)

% Specify Mooring Line properties

% Number of mooring lines
nl = 4;
% Linear mass density of the line (kg/m)
% The same property value is used for all mooring lines in this example
rho_l = 10;
% Unstretched length of the entire mooring line (m)
% The same property value is used for all mooring lines in this example
L = 44;
% Number of segments per mooring line
% This version of code only allows all mooring lines to have the same
% number of line segments.
N = 10;
% Diameter of the line (m)
% The same property value is used for all mooring lines in this example
d = .05;
```

```
% Stiffness of mooring lines (N/m) K = [K1, K2, ..., Knl]
K = [446250, 446250, 446250, 446250];
% Attachment point of mooring lines to body in body-fixed frame (m)
% pBi = [pBx'; pBy'; pBz']
pB1 = [7; -2; 0];
pB2 = [7; 2; 0];
pB3 = [-7; 2; 0];
pB4 = [-7; -2; 0];
% Matrix of all body attachment points, each column is a lines attachment
% point vector
pB = [pB1, pB2, pB3, pB4];
% Attachment point of mooring lines to ground in global frame (m)
% pGi = [pGX; pGY; pGZ]
pG1 = [35; -30; 0];
pG2 = [35; 30; 0];
pG3 = [-35; 30; 0];
pG4 = [-35; -30; 0];
% Matrix of all ground attachment points, each column is a lines
attachment
% point vector
pG = [pG1, pG2, pG3, pG4];

% Set up initial position vector
q_i = [c_i;angles_i];

% Adds the coordinates of the line nodes to the initial position vector
for i=1:nl
    % Initial coordinates of the nodes
    % Sets up nodes along a straight line from the Body attachment point
to
    % the ground attachment point

    % Body node in global frame
    p0 = c_i + Txyz(angles_i)*pB(:,i);
    % Line vector from Body node to Ground node
    Lvec = pG(:,i) - p0;
    % Initialize node coords.
    ql_i = zeros((N+1)*3,1);
    % Body node
    ql_i(1:3) = p0;
    % Ground node
    ql_i((N+1)*3 - 2:(N+1)*3) = pG(:,i);
    % Sets up segment nodes
    if N>1
        for j=1:(N-1)
            ql_i(1+3*j:3+3*j) = p0 + Lvec*j/N;
        end
    end

    % Sorts the coords. of the line nodes into column vectors
    P_i = reshape(ql_i,3,length(ql_i)/3);

    % Create new line object
```

```
    Lines(i) = newLine(rho_l,L,N,d,K(i),pB(:,i),pG(:,i),P_i)

    % Add initial node coords. to initial position vector
    q_i = [q_i;ql_i];
end

% Solve the system, 1st iteration
qf = EquilSolver(Body,Lines,q_i);

% Plot the Body and Mooring Lines
PlotSolu(qf,Body,Lines)
% Plot the drag force vectors on the Body and Mooring Lines
%PlotDrag3D(qf,Body,Lines)

% 2nd iteration, updates system drag forces
[Body2, Lines2] = SysUpdate(Body,Lines,qf);
qf2 = EquilSolver(Body2,Lines2,qf);

figure
PlotSolu(qf2,Body2,Lines2)
%PlotDrag3D(qf2,Body2,Lines2)

% 3rd iteration, doubles number of line segments
[Body3, Lines3, q3] = SysRefine(Body2,Lines2,qf2);
qf3 = EquilSolver(Body3,Lines3,q3);

figure
PlotSolu(qf3,Body3,Lines3)
%PlotDrag3D(qf3,Body3,Lines3)

%% 4th iteration, updates system
% [Body4, Lines4] = SysUpdate(Body3,Lines3,qf3);
% qf4 = EquilSolver(Body4,Lines4,qf3);
%
% figure
% PlotSolu(qf4,Body4,Lines4)
% PlotDrag3D(qf4,Body4,Lines4)
```

Appendix 3 – Catenary_test.m – Elastic Catenary Test Setup

```
clear
close all
clc

% Elastic Catenary test problem, compares the numerical solution to an
% analytical solution.
% The fluid velocity function must return zero fluid velocity, and the
% fluid density in the newLine class must be set to zero.

% Specify Body properties, only used to specify the horizontal and
vertical
% applied force, and the initial position of the free end, all other
% properties must be set to zero.

% Mass of body (kg)
M = 0;
% Volume of body (m^3)
V = 0;
% Center of buoyancy relative to center of mass in body-fixed frame (m),
% column vector [cbx'; cby'; cbz']
cB = [0; 0; 0];
% Coefficients of Drag in the body-fixed frame [CDx'; CDy'; CDz']
CD = [0; 0; 0];
% Effective surface areas of the Body in the body-fixed frame (m^2)
% [Ax'; Ay'; Az']
AB = [0; 0; 0];
% Applied forces to the Body in global frame (N) [FaX; FaY; FaZ]
Fa = [50; 0; 75];
% Applied torques to the Body in global frame (N*m) [TaX; TaY; TaZ]
Ta = [0; 0; 0];
% Initial position of center of mass in global frame (m) c = [x; y; z]
c_i = [8; 0; 6];
% Initial xyz Euler angles (radians) angles_i = [alpha; beta; gamma]
angles_i = [0; 0; 0];

% Creates new Body object
Body = newBody(M,V,cB,CD,AB,Fa,Ta,c_i,angles_i)

% Specify Line properties

% Linear mass density of the line (kg/m)
rho_l = 0.7;
% Unstretched length of the entire mooring line (m)
L = 10;
% Number of segments per mooring line
N = 5;
% Diameter of the line (m)
d = .01;
% Elastic Modulus of the line (Pa, N/m^2)
E = 1e8;
% Stiffness of mooring line (N/m)
```

```
K = E*pi*(d/2)^2/L;
% Attachment point of mooring line to body in body-fixed frame (m)
% [pBx';pBy';pBz'], must be set to zero for this test
pB = [0;0;0];
% Attachment point of mooring line to ground in global frame (m)
% [pGX; pGY; pGZ]
pG = [0;0;0];

% Initial coordinates of the nodes
% Sets up nodes along a straight line from the Body node to the Ground
node
p0 = c_i + Txyz(angles_i)*pB;    % Body node in global frame
Lvec = pG - p0;                  % Line vector from Body node to Ground
node
ql_i = zeros((N+1)*3,1);         % Initialize node coords.
ql_i(1:3) = p0;                  % Body node
ql_i((N+1)*3 - 2:(N+1)*3) = pG; % Ground node
if N>1                           % Sets up segment nodes
    for j=1:(N-1)
        ql_i(1+3*j:3+3*j) = p0 + Lvec*j/N;
    end
end

% Sorts the coords. of the line nodes into column vectors
P_i = reshape(ql_i,3,length(ql_i)/3);

% Create new line object
Lines(1) = newLine(rho_l,L,N,d,K,pB,pG,P_i)

% Set up initial displacements
q_i = [c_i;angles_i;ql_i];

% Plot analytical solution
A = pi*(d/2)^2; % Cross-sectional area
H = Fa(1);      % X-component of force
V = Fa(3);      % Z-component of force
s = 0:L/1000:L; % Unstretched arclength
% Analytical X and Z solutions
[Xa,Za] = Catenary_Solu(E,A,L,rho_l,H,V,s);
Ya = zeros(size(Xa));
axis equal
plot3(Xa,Ya,Za,'r','LineWidth',2)

% Solve the system
qf = EquilSolver(Body,Lines,q_i);

% Plot the solution
PlotSolu(qf,Body,Lines)
view(0,0)
xlim([0 8]); xlabel('X (m)')
zlim([0 6]); zlabel('Z (m)')
legend('Analytical','Numerical','Location','NorthWest')
```

```
% Calc. Errors
s = 0:L/N:L;    % Arc length at nodes
Zerror1 = zeros(1,N+1);  % Z error at nodes
Xerror1 = zeros(1,N+1);  % X error at nodes
% Analytical solution at nodes
[Xa1,Za1] = Catenary_Solu(E,A,L,rho_l,H,V,s);
for i=0:N
    Xerror1(length(Xerror1)-i) = qf(7+3*i)-Xa1(length(Xa1)-i);
    Zerror1(length(Zerror1)-i) = qf(9+3*i)-Za1(length(Za1)-i);
end
Xerror1
Zerror1
Xe1 = norm(Xerror1)          % 2 norm of Xerror
Ze1 = norm(Zerror1)          % 2 norm of Zerror
Xe1inf = norm(Xerror1,inf)    % Inf. norm of Xerror
Ze1inf = norm(Zerror1,inf)    % Inf. nrom of Zerror

%% 2nd iteration, doubles number of nodes
[Body2, Lines2, q2] = SysRefine(Body,Lines,qf);
qf2 = EquilSolver(Body2,Lines2,q2);

figure
plot3(Xa,Ya,Za,'r','LineWidth',2)
PlotSolu(qf2,Body2,Lines2)
view(0,0)
xlim([0 8]); xlabel('X (m)')
zlim([0 6]); zlabel('Z (m)')
legend('Analytical','Numerical','Location','NorthWest')

% Calc. Errors
N2 = Lines2(1).N;
s = 0:L/N2:L;    % Arc length at nodes
Zerror2 = zeros(1,N2+1);  % Z error at nodes
Xerror2 = zeros(1,N2+1);  % X error at nodes
% Analytical solution at nodes
[Xa2,Za2] = Catenary_Solu(E,A,L,rho_l,H,V,s);
for i=0:N2
    Xerror2(length(Xerror2)-i) = qf2(7+3*i)-Xa2(length(Xa2)-i);
    Zerror2(length(Zerror2)-i) = qf2(9+3*i)-Za2(length(Za2)-i);
end
Xe2 = norm(Xerror2)          % 2 norm of Xerror
Ze2 = norm(Zerror2)          % 2 norm of Zerror
Xe2inf = norm(Xerror2,inf)    % Inf. norm of Xerror
Ze2inf = norm(Zerror2,inf)    % Inf. nrom of Zerror

% 3rd iteration, doubles number of nodes
[Body3, Lines3, q3] = SysRefine(Body2,Lines2,qf2);
qf3 = EquilSolver(Body3,Lines3,q3);

figure
plot3(Xa,Ya,Za,'r','LineWidth',2)
PlotSolu(qf3,Body3,Lines3)
view(0,0)
```

```
xlim([0 8]); xlabel('X (m)')
zlim([0 6]); zlabel('Z (m)')
legend('Analytical','Numerical','Location','NorthWest')

% Calc. Errors
N3 = Lines3(1).N;
s = 0:L/N3:L;    % Arc length at nodes
Zerror3 = zeros(1,N3+1);  % Z error at nodes
Xerror3 = zeros(1,N3+1);  % X error at nodes
% Analytical solution at nodes
[Xa3,Za3] = Catenary_Solu(E,A,L,rho_l,H,V,s);
for i=0:N3
    Xerror3(length(Xerror3)-i) = qf3(7+3*i)-Xa3(length(Xa3)-i);
    Zerror3(length(Zerror3)-i) = qf3(9+3*i)-Za3(length(Za3)-i);
end
Xe3 = norm(Xerror3)          % 2 norm of Xerror
Ze3 = norm(Zerror3)          % 2 norm of Zerror
Xe3inf = norm(Xerror3,inf)    % Inf. norm of Xerror
Ze3inf = norm(Zerror3,inf)    % Inf. nrom of Zerror

% 4th iteration, doubles number of nodes
[Body4, Lines4, q4] = SysRefine(Body3,Lines3,qf3);
qf4 = EquilSolver(Body4,Lines4,q4);

figure
plot3(Xa,Ya,Za,'r','LineWidth',2)
PlotSolu(qf4,Body4,Lines4)
view(0,0)
xlim([0 8]); xlabel('X (m)')
zlim([0 6]); zlabel('Z (m)')
legend('Analytical','Numerical','Location','NorthWest')

% Calc. Errors
N4 = Lines4(1).N;
s = 0:L/N4:L;    % Arc length at nodes
Zerror4 = zeros(1,N4+1);  % Z error at nodes
Xerror4 = zeros(1,N4+1);  % X error at nodes
% Analytical solution at nodes
[Xa4,Za4] = Catenary_Solu(E,A,L,rho_l,H,V,s);
for i=0:N4
    Xerror4(length(Xerror4)-i) = qf4(7+3*i)-Xa4(length(Xa4)-i);
    Zerror4(length(Zerror4)-i) = qf4(9+3*i)-Za4(length(Za4)-i);
end
Xe4 = norm(Xerror4)          % 2 norm of Xerror
Ze4 = norm(Zerror4)          % 2 norm of Zerror
Xe4inf = norm(Xerror4,inf)    % Inf. norm of Xerror
Ze4inf = norm(Zerror4,inf)    % Inf. nrom of Zerror

% Plot errors versus N
Nvals = [N;N2;N3;N4];
figure
plot(Nvals,[Xe1;Xe2;Xe3;Xe4],Nvals,[Xe1inf;Xe2inf;Xe3inf;Xe4inf],...
    Nvals,[Ze1;Ze2;Ze3;Ze4],Nvals,[Ze1inf;Ze2inf;Ze3inf;Ze4inf])
```

```
xlabel('N'); ylabel('Error (m)');
legend('||Xe||_2','||Xe||_\infty','||Ze||_2','||Ze||_\infty')
figure
loglog(Nvals,[Xe1;Xe2;Xe3;Xe4],Nvals,[Xe1inf;Xe2inf;Xe3inf;Xe4inf],...
    Nvals,[Ze1;Ze2;Ze3;Ze4],Nvals,[Ze1inf;Ze2inf;Ze3inf;Ze4inf])
xlabel('N'); ylabel('Error (m)');
legend('||Xe||_2','||Xe||_\infty','||Ze||_2','||Ze||_\infty')
```

Appendix 4 – Catenary_Solu.m – Analytical Catenary Solution Equations

```
function [Xa,Za] = Catenary_Solu(E,A,L,rho,H,V,s)
% Calculates the 2D analytical solution of an elastic catenary line fixed
% at one end with a horizontal and vertical force applied at the other
end.
% Inputs: Elastic modulus E, unstrained cross-section area A, unstrained
% length L, linear mass density rho, applied horizontal force H, applied
% vertical force V, and arc length s measured from the fixed end.
% Output: The horizontal (Xa) and vertical position (Za) of the line
% measured from the fixed end, returned in a column matrix Solu = [Xa,Za]
% Ex. Solu = Catenary_Solu(E,A,L,rho,H,V,s)

g = newLine.g;  % gravity

Xa = H*s/(E*A) + H/(rho*g)*(asinh((V-rho*g*L+rho*g*s)/H)-asinh((V-
rho*g*L)/H));
Za = s/(E*A)*(V-rho*g*L)+0.5*rho*g*s.*s/(E*A)+H/(rho*g)*(sqrt(1+((V-
rho*g*L+rho*g*s)/H).^2)-sqrt(1+((V-rho*g*L)/H)^2));

end
```

Appendix 5 – RigidBody_test.m – Rigid Body Rotation Test Setup

```
clear
close all
clc

% Rigid Body rotation test problem
% The fluid velocity function must return zero fluid velocity, and the
% fluid density in the newLine class must be set to zero.

% Define parameters for analytical solution
K1 = 10;     % Spring constant 1 (N/m)
K2 = 5;      % Spring constant 2 (N/m)
L = 3;       % Free length of both springs
% Width of body, and separation of attachment points on ground (m)
W = 5;
% Length along body of applied force, measured from left end (m)
l3 = 3;
F = 20;      % Applied Force (N), acts only in Y direction
% Applied Moment (N*m), gamma direction in the body-fixed frame, Z
% direction in global frame
T = 25;

% Specify Body properties, only used to specify applied force, torque and
% initial body position, all other properties set to zero.

% Mass of body (kg)
M = 0;
% Volume of body (m^3)
V = 0;
% Center of buoyancy relative to center of mass in body-fixed frame (m),
% column vector [cbx'; cby'; cbz']
cB = [0; 0; 0];
% Coefficients of Drag in the body-fixed frame [CDx'; CDy'; CDz']
CD = [0; 0; 0];
% Effective surface areas of the Body in the body-fixed frame (m^2)
% [Ax'; Ay'; Az']
AB = [0; 0; 0];
% Applied forces to the Body in global frame (N) [FaX; FaY; FaZ]
Fa = [0; F; 0];
% Applied torques to the Body in global frame (N*m) [TaX; TaY; TaZ]
Ta = [0; 0; T];
% Initial position of center of mass in global frame (m) c = [x; y; z]
c_i = [l3; L; 0];
% Initial xyz Euler angles (radians) angles_i = [alpha; beta; gamma]
angles_i = [0; 0; 0];

% Creates new Body object
Body = newBody(M,V,cB,CD,AB,Fa,Ta,c_i,angles_i)

% Specify Mooring Line properties, used as linear springs, the linear
% density must be set to zero for massless springs
```

```
% Linear mass density of the line (kg/m)
rho_l = 0;
% Unstretched length of the entire mooring line (m)
% L
% Number of segments per mooring line
N = 1;
% Diameter of the line (m)
d = .01;
% Stiffness of mooring lines (N/m)
K = [K1, K2];
% Attachment point of mooring line to body in body-fixed frame (m)
% pBi = [pBx';pBy';pBz']
pB1 = [-l3;0;0];
pB2 = [W-l3;0;0];
pB = [pB1, pB2];
% Attachment point of mooring line to ground in global frame (m)
% pGi = [pGX; pGY; pGZ]
pG1 = [0;0;0];
pG2 = [W;0;0];
pG = [pG1, pG2];

% Set up initial 2D displacements
q_i2D = [c_i(1:2);angles_i(3)];

nl = 2; % # mooring lines

for i=1:nl
    % Initial coordinates of the nodes
    % Sets up nodes along a straight line from the Body attachment point
to
    % the ground attachment point

    % Body node in global frame
    p0 = c_i + Txyz(angles_i)*pB(:,i);
    % Line vector from Body node to Ground node
    Lvec = pG(:,i) - p0;
    % Initialize node coords.
    ql_i = zeros((N+1)*3,1);
    % Body node
    ql_i(1:3) = p0;
    % Ground node
    ql_i((N+1)*3 - 2:(N+1)*3) = pG(:,i);
    % Sets up segment nodes
    if N>1
        for j=1:(N-1)
            ql_i(1+3*j:3+3*j) = p0 + Lvec*j/N;
        end
    end

    % Sorts the coords. of the line nodes into column vectors
    P_i = reshape(ql_i,3,length(ql_i)/3);

    % Create new line object
```

```
    Lines(i) = newLine(rho_l,L,N,d,K(i),pB(:,i),pG(:,i),P_i)

    % 2D node coords.
    ql_i2D = reshape(P_i(1:2,:),2*(N+1),1);

    % Add initial node coords. to initial displacements
    q_i2D = [q_i2D;ql_i2D];
end

% Solve the system with the 2D equilibrium solver setup
[qf,hessian] = EquilSolver2D(Body,Lines,q_i2D);

% Set up properties to solve the analytical solution

% Sets up parameters for the analytical equations
param = [K1; K2; L; L; W; l3; F; T];

% Define initial guess for Y = [x1; z1; theta]
Y = [3; 1; 1.5];

% Solve the analytical equilibrium equations
[Y,Vf] = fsolve(@(Y)RigidBodyfuns(Y,param),Y)

% Calc x2 and y2
x2 = Y(1) + W*cos(Y(3))
y2 = Y(2) + W*sin(Y(3))
x3 = Y(1) + l3*cos(Y(3))
y3 = Y(2) + l3*sin(Y(3))

% Plot original and deformed shapes
Xorig = [0; 0; W; W];
Yorig = [0; L; L; 0];
Zorig = zeros(size(Xorig));
Xnew = [0; Y(1); x2; W];
Ynew = [0; Y(2); y2; 0];
Znew = zeros(size(Xnew));
hold on
plot3(Xorig,Yorig,Zorig,'go--')
plot3(Xnew,Ynew,Znew,'ro-')
xlim([-.5 5.5])
% Plot the solution
PlotSolu2D(qf,Body,Lines)

% Calc. errors, 2D
X1e = qf(4) - Y(1)
Y1e = qf(5) - Y(2)
Theta_e = qf(3) - Y(3)
X2e = qf(8) - x2
Y2e = qf(9) - y2

% Eigen values of the Hessian
eig(hessian)
```

Appendix 6 – RigidBodyfuns.m – Analytical Rigid Body Rotation Equilibrium Equations

```
% Rigid Body rotation test problem, analytical equations
% Define the functions to be solved by 'fsolve'
% These are the partial derivatives of the potential energy function with
% respect to each generalized coordinate
% Where Y = [x1; y1; theta]
% param = [k1; k2; l1; l2; W; l3; F; M]
function Func = RigidBodyfuns(Y,param)
Func = zeros(3,1);
k1 = param(1);
k2 = param(2);
l1 = param(3);
l2 = param(4);
W = param(5);
l3 = param(6);
F = param(7);
M = param(8);

Func(1) = k1*Y(1)*(sqrt(Y(1)^2 + Y(2)^2) - l1)/sqrt(Y(1)^2 + Y(2)^2) + ...
    k2*(sqrt((Y(1)+W*cos(Y(3))-W)^2 + (Y(2)+W*sin(Y(3)))^2) -
l2)*(Y(1)+W*cos(Y(3))-W)/sqrt((Y(1)+W*cos(Y(3))-W)^2 +
(Y(2)+W*sin(Y(3)))^2);

Func(2) = k1*Y(2)*(sqrt(Y(1)^2 + Y(2)^2) - l1)/sqrt(Y(1)^2 + Y(2)^2) + ...
    k2*(sqrt((Y(1)+W*cos(Y(3))-W)^2 + (Y(2)+W*sin(Y(3)))^2) -
l2)*(Y(2)+W*sin(Y(3)))/sqrt((Y(1)+W*cos(Y(3))-W)^2 + (Y(2)+W*sin(Y(3)))^2)
- F;

Func(3) = -F*l3*cos(Y(3)) -M + ...
    k2*(sqrt((Y(1)+W*cos(Y(3))-W)^2 + (Y(2)+W*sin(Y(3)))^2) - l2)*(-
2*W*(Y(1)+W*cos(Y(3))-W)*sin(Y(3)) +
2*(Y(2)+W*sin(Y(3)))*W*cos(Y(3)))/(2*sqrt((Y(1)+W*cos(Y(3))-W)^2 +
(Y(2)+W*sin(Y(3)))^2));
end
```

Appendix 7 – FluidVelocity.m – Fluid Velocity Profile Functions

```
function Uf = FluidVelocity(p)
% Returns the fluid velocity in the global frame as a column vector
% Uf = [UfX; UfY; UfZ], given the global coordinates of the point of
% interest p = [x; y; z]
% Ex. Uf = FluidVelocity(p)

x = p(1);
y = p(2);
z = p(3);

Uf = zeros(3,1);

% Defines fluid velocity profile for example test problem
% Magnitude of fluid velocity (m/s)
U = 4;

% Angle fluid velocity makes from Y-axis, + clockwise (degrees)
theta = 0;

% Calculate the components of the fluid velocity
% Uf(1) = UfX, Uf(2) = UfY, Uf(3) = UfZ
Uf(1) = U*sind(theta);
Uf(2) = U*cosd(theta);
Uf(3) = 0;
end
```

Appendix 8 – newBody.m – Body Class

```
% Body Class
% Keeps track of all properties, forces and potential energies for a body

classdef newBody

    % Define properties constant to all Body objects
    properties (Constant)
        g = 9.81;       % Gravity (m/s^2)
        rho_f = 1020;   % Fluid density (kg/m^3)
    end

    % Define properties specific to each Body
    properties (SetAccess = private)
        M       % Mass of body (kg)
        V       % Volume of body (m^3)
        cB      % Center of buoyancy relative to center of mass in body-
fixed frame (m), column vector [cbx'; cby'; cbz']
        CD      % Coefficients of Drag in the body-fixed frame [CDx';
CDy'; CDz']
        A       % Effective surface areas of the Body in the body-fixed
frame (m^2) [Ax'; Ay'; Az']
        Fa      % Applied forces to the Body in global frame (N) [FaX;
FaY; FaZ]
        Ta      % Applied torques to the Body in global frame (N*m) [TaX;
TaY; TaZ]
        W       % Weight of the Body (N) (acts in negative Z direction)
        FB      % Buoyant force on the Body (N) (acts in positive Z
direction)
        FD      % Constant drag force on the Body (N)
    end

    methods

        % Define the syntax to create a new Body object
        function obj = newBody(M,V,cB,CD,A,Fa,Ta,c_i,angles_i)

            % Initialize the properties
            obj.M = M;
            obj.V = V;
            obj.cB = cB;
            obj.CD = CD;
            obj.A = A;
            obj.Fa = Fa;
            obj.Ta = Ta;
            obj.W = -M*newBody.g;
            obj.FB = newBody.rho_f*V*newBody.g;
            obj.FD = BodyDragForce(CD,A,c_i,angles_i);
        end

        function VW = Vb_weight(obj,z)
            % Calculate the gravitational potential energy of the Body
```

```
            % Given the global z coordinate of the center of mass
            % Ex. VW = Vb_weight(obj,z)

            % VW = -W*z
            VW = -obj.W*z;
        end

        function VB = Vb_buoyancy(obj,c,angles)
            % Calculate the potential energy of the buoyant force
            % Given the global coordinates of the center of mass c =
[x;y;z]
            % and the XYZ Euler angles = [alpha; beta; gamma]
            % Ex. VB = Vb_buoyancy(obj,c,angles)

            % Calc. center of buoyancy in global frame, cBg
            cBg = c + Txyz(angles)*obj.cB;

            % VB = -FB*cBgZ
            VB = -obj.FB*cBg(3);
        end

        function VD = Vb_drag(obj,c)
            % Calculate the potential energy of the drag force on the Body
            % Given the global coordinates of the center of mass c =
[x;y;z]
            % Ex. VD = Vb_drag(obj,c)

            % VD = -FDX*x - FDY*y - FDZ*z
            VD = -obj.FD(1)*c(1) - obj.FD(2)*c(2) - obj.FD(3)*c(3);
        end

        function VFa = Vb_appl_forces(obj,c)
            % Calculate the potential energy of the applied forces
            % Given the global coordinates of the center of mass c =
[x;y;z]
            % Ex. VFa = Vb_appl_forces(obj,c)

            % VFa = -FaX*x - FaY*y - FaZ*z
            VFa = -obj.Fa(1)*c(1) - obj.Fa(2)*c(2) - obj.Fa(3)*c(3);
        end

        function VTa = Vb_appl_torques(obj,angles)
            % Calculate the potential energy of the applied torques
            % Given the XYZ Euler angles = [alpha; beta; gamma]
            % Ex. VTa = Vb_appl_torques(obj,angles)

            % Calc. torques in body-fixed frame, Tab = [Ta_alpha; Ta_beta;
Ta_gamma]
            % Inverse Txyz = Transpose Txyz = Txyz(angles)'
            Tab = Txyz(angles)'*obj.Ta;

            % VTa = -Ta_alpha*alpha - Ta_beta*beta - Ta_gamma*gamma
```

```
            VTa = -Tab(1)*angles(1) - Tab(2)*angles(2) - Tab(3)*angles(3);
        end

        function Vb_total = Vbody_total(obj,c,angles)
            % Calculate the total potential energy of all forces on the
body
            % Given the global coordinates of the center of mass c =
[x;y;z]
            % and the XYZ Euler angles = [alpha; beta; gamma]
            % Ex. Vb_total = Vbody_total(obj,c,angles)

            VW = Vb_weight(obj,c(3));            % Gravitational potential
            VB = Vb_buoyancy(obj,c,angles);      % Buoyancy potential
            VD = Vb_drag(obj,c);                 % Drag potential
            VFa = Vb_appl_forces(obj,c);         % Applied force potential
            VTa = Vb_appl_torques(obj,angles);   % Applied torque
potential

            % Total Body potential energy
            Vb_total = VW + VB + VD + VFa + VTa;
        end

        function PlotBody3D(obj,c,angles)
            % Plots a representation of the Body object
            % Given the global coordinates of the center of mass c =
[x;y;z]
            % and the XYZ Euler angles = [alpha; beta; gamma]
            % Ex. PlotBody3D(obj,c,angles)

            % Define the dimensions of the Body, as a 3D box centered with
            % the body-fixed frame. User may also create new properties of
            % the Body class to define values for each Body object.
            w = 28;     % width along x' axis (m)
            h = 8.5;    % height along z' axis (m)
            d = 4;      % depth along y' axis (m)

            % Corner points of the box, to include translation and
rotation
            % of the Body
            p1 = c + Txyz(angles)*[w/2; -d/2; -h/2];
            p2 = c + Txyz(angles)*[w/2; -d/2; h/2];
            p3 = c + Txyz(angles)*[-w/2; -d/2; h/2];
            p4 = c + Txyz(angles)*[-w/2; -d/2; -h/2];
            p5 = c + Txyz(angles)*[w/2; d/2; -h/2];
            p6 = c + Txyz(angles)*[w/2; d/2; h/2];
            p7 = c + Txyz(angles)*[-w/2; d/2; h/2];
            p8 = c + Txyz(angles)*[-w/2; d/2; -h/2];

            % Front surface coords.
            xf = [p1(1); p2(1); p3(1); p4(1)];
            yf = [p1(2); p2(2); p3(2); p4(2)];
            zf = [p1(3); p2(3); p3(3); p4(3)];
```

```
            % Back surface coords.
            xb = [p5(1); p6(1); p7(1); p8(1)];
            yb = [p5(2); p6(2); p7(2); p8(2)];
            zb = [p5(3); p6(3); p7(3); p8(3)];

            % Right surface coords.
            xr = [p5(1); p6(1); p2(1); p1(1)];
            yr = [p5(2); p6(2); p2(2); p1(2)];
            zr = [p5(3); p6(3); p2(3); p1(3)];

            % Left surface coords.
            xl = [p8(1); p7(1); p3(1); p4(1)];
            yl = [p8(2); p7(2); p3(2); p4(2)];
            zl = [p8(3); p7(3); p3(3); p4(3)];

            % Top surface coords.
            xt = [p2(1); p6(1); p7(1); p3(1)];
            yt = [p2(2); p6(2); p7(2); p3(2)];
            zt = [p2(3); p6(3); p7(3); p3(3)];

            % Bottom surface coords.
            xm = [p1(1); p5(1); p8(1); p4(1)];
            ym = [p1(2); p5(2); p8(2); p4(2)];
            zm = [p1(3); p5(3); p8(3); p4(3)];

            % Body polygon X coords.
            X = [xf xb xr xl xt xm];

            % Body polygon Y coords.
            Y = [yf yb yr yl yt ym];

            % Body polygon Y coords.
            Z = [zf zb zr zl zt zm];

            % Plots 3D faces of the Body on the current plot
            fill3(X,Y,Z,'g','FaceAlpha',0.2)
        end
    end
end

function FD = BodyDragForce(CD,A,c,angles)
% Calculates the drag force on the Body, estimated as a constant force for
% the solution process. Given the Body's coefficients of drag, CD, and
% surface areas, A, initial global coordinates of the center of mass
% c = [x;y;z] and the initial XYZ Euler angles = [alpha; beta; gamma].
% Ex. FD = DragForce(CD,A,c,angles)

% Get fluid velocity in global frame at center of mass, uf = [ufX; ufY;
ufZ]
uf = FluidVelocity(c);

% Calc. fluid velocity in body-fixed frame, ufb = [ufx'; ufy'; ufz']
```

```
% Inverse Txyz = Transpose Txyz = Txyz(angles)'
ufb = Txyz(angles)'*uf;

% Calc. drag forces in body-fixed frame
% FDb = 0.5*rho_f*CD*A*uf*|uf|, FDb = [FDx'; FDy'; FDz']
FDb = zeros(3,1);
for i=1:3
    FDb(i) = 0.5*newBody.rho_f*CD(i)*A(i)*ufb(i)*abs(ufb(i));
 end

% Calc. drag forces in global frame, FD = [FDX; FDY; FDZ]
FD = Txyz(angles)*FDb;
end
```

Appendix 9 – newLine.m – Mooring Line Class

```
% Line Class
% Keeps track of the potential energies for all N segments of an entire
% mooring line

classdef newLine

    % Define properties constant to all Line objects
    properties (Constant)
        g = 9.81;       % Gravity (m/s^2)
        rho_f = 1020;   % Fluid density (kg/m^3)

        CDt = 0.3;      % Coefficient of drag tangent to segment
        CDn = 1;        % Coefficient of drag normal to segment
    end

    % Define properties specific to each mooring line
    properties (SetAccess = private)
        % Properties to be defined when creating a new Line object
        rho_l   % Linear mass density of the line (kg/m)
        L       % Unstretched length of the entire mooring line (m)
        N       % Number of segments per mooring line
        d       % Diameter of the line (m)
        K       % Stiffness of mooring line (N/m)
        pB      % Attachment point of mooring line to body in body-fixed
frame (m) [pBx';pBy',pBz']
        pG      % Attachment point of mooring line to ground in global
frame (m) [pGX; pGY; pGZ]

        % Properties calculated from previous properties
        Ls      % Original length of each segment (m)
        Ms      % Mass of each segment (kg)
        Vs      % Volume of each segment (m^3)
        Ast     % Area of each segment in the tangental direction (m^2)
        Asn     % Area of each segment in the normal direction (m^2)
        Ks      % Stiffness of each segment (N/m)
        Qi      % Matrix (3xN) containing the initial coordinates of the
center of each segment as column vectors
        FDs     % Constant drag force on each segment (N) (3xN matrix)
        FDt     % Tangential component of drag (N) (3xN matrix)
        FDn     % Normal component of drag (N) (3xN matrix)
    end

    methods

        % Define the syntax to create a new Body object
        function obj = newLine(rho_l,L,N,d,K,pB,pG,P_i)

            % Initialize the properties
            obj.rho_l = rho_l;
            obj.L = L;
            obj.N = N;
```

```
            obj.d = d;
            obj.K = K;
            obj.pB = pB;
            obj.pG = pG;
            obj.Ls = L/obj.N;
            obj.Ms = rho_l*obj.Ls;
            obj.Vs = pi*(d^2)/4*obj.Ls;
            obj.Ast = pi*d*obj.Ls;
            obj.Asn = d*obj.Ls;
            obj.Ks = obj.N*obj.K;
            obj.Qi = SegCenters(P_i);
            [obj.FDs, obj.FDt, obj.FDn] =
SegDragForces(obj.Ast,obj.Asn,obj.Qi,P_i,obj.N);
        end

        function VW = Vl_weight(obj,Q)
            % Calculates the potential energy of the net weight and
buoyant
            % force for the entire mooring Line, given the 3xN matrix Q of
            % the coordinates of the center of each segment.
            % Ex. VW = Vl_weight(obj,Q)

            % Calculates the total weight+buoyancy potential energy of the
            % line by summing the potential of each segment. The net
weight
            % and buoyant force on each segment is w = (-Ms + rho*Vs)*g.
            % The potential energy of each segment is VW = -w*qZ
            VW = -(-obj.Ms + newLine.rho_f*obj.Vs)*newLine.g*sum(Q(3,:));
        end

        function VD = Vl_drag(obj,Q)
            % Calculates the potential energy of the drag force for the
            % entire mooring Line, given the 3xN matrix Q of the
            % coordinates of the center of each segment.
            % Ex. VD = Vl_drag(obj,Q)

            VD = 0; % Initialize drag potential

            % Calculate the total drag potential energy by summing the
            % potential of each segment, VD = -FDX*qX - FDY*qY - FDZ*qZ
            for i=1:obj.N
                VD = VD - obj.FDs(1,i)*Q(1,i) - obj.FDs(2,i)*Q(2,i) -
obj.FDs(3,i)*Q(3,i);
            end
        end

        function VK = Vl_stiffness(obj,P)
            % Calculates the potential energy due to stiffness for the
            % entire mooring Line, given the 3xN+1 matrix P of the
            % coordinates of N+1 consecutive nodes.
            % Ex. VK = Vl_stiffness(obj,P)

            VK = 0; % Initialize stiffness potential
```

```
            % Calculate the total stiffness potential energy by summing
the
            % potential of each segment, VK = (1/2)Ks*(delta_Ls)^2
            for i=1:obj.N
                VK = VK + 0.5*obj.Ks*(norm(P(:,i+1)-P(:,i)) - obj.Ls)^2;
            end
        end

        function Vl_total = Vline_total(obj,P)
            % Calculate the total potential energy of the mooring Line,
            % given the 3xN+1 matrix P of the coordinates of N+1
            % consecutive nodes.
            % Ex. Vl_total = Vline_total(obj,P)

            Q = SegCenters(P);          % Calc. segment centers
            VW = Vl_weight(obj,Q);      % Weight + buoyant potential
            VD = Vl_drag(obj,Q);        % Drag potential
            VK = Vl_stiffness(obj,P);   % Stiffness potential

            % Total mooring Line potential energy
            Vl_total = VW + VD + VK;
        end
    end
end

function [FDs, FDt, FDn] = SegDragForces(Ast,Asn,Q,P,N)
% Calculates the drag force on each line segment, estimated as a constant
% force for the solution process based on the initial locations of segment
% centers. Returns the 3xN matrices FDs, FDt, FDn of the net drag force,
% tangential drag force, and normal drag force respectively as column
% vectors on each segment, given the segment areas in the tangential and
% normal directions respectively (Ast, Asn), the 3xN matrix Q of the
% coordinates of the center of each segment, the 3xN+1 matrix P of the
% coordinates of consecutive nodes, and the number of segments N.
% Ex. [FDs, FDt, FDn] = SegDragForces(Ast,Asn,Q,P,N)

rho = newLine.rho_f;    % Fluid density
CDt = newLine.CDt;      % Tangential drag coeff.
CDn = newLine.CDn;      % Normal drag coeff.

% Initialize matrices
Uf = zeros(3,N);    % Fluid velocities
t = zeros(3,N);     % Tangent unit vectors
a = zeros(1,N);     % Tangential magnitude of fluid velocity
b = zeros(1,N);     % Normal magnitude of fluid velocity
n = zeros(3,N);     % Normal unit vectors
FDs = zeros(3,N);   % Segments net drag forces
FDt = FDs;          % Segments tangential drag force
FDn = FDs;          % Segments normal drag force

% Calculate the drag force for each segment
for i=1:N
```

```
    % Get fluid velocity at segment center
    Uf(:,i) = FluidVelocity(Q(:,i));
    % Calc. tangent unit vector, t = (P2-P1)/|P2-P1|
    t(:,i) = (P(:,i+1)-P(:,i))/norm(P(:,i+1)-P(:,i));
    % Tangential magnitude, a = UfX*tX + UfY*tY + UfZ*tZ
    a(i) = Uf(1,i)*t(1,i) + Uf(2,i)*t(2,i) + Uf(3,i)*t(3,i);
    % Normal magnitude, b = sqrt(|Uf|^2 - a^2);
    b(i) = sqrt(Uf(1,i)^2 + Uf(2,i)^2 + Uf(3,i)^2 - a(i)^2);
    % Check for non-zero normal magnitude
    if b(i) ~= 0
        % Normal unit vector, n = (1/b)(Uf - a*t)
        n(:,i) = (1/b(i))*(Uf(:,i) - a(i)*t(:,i));
    end
    % Calc. tangential drag force, FDt = (1/2)rho*CDt*At*a*|a|
    FDt(:,i) = 0.5*rho*CDt*Ast*a(i)*abs(a(i))*t(:,i);
    % Calc. normal drag froce, FDn = (1/2)rho*CDn*An*b^2
    FDn(:,i) = 0.5*rho*CDn*Asn*(b(i)^2)*n(:,i);
    % Calc. net drag force, FDs = FDt + Fdn
    FDs(:,i) = FDt(:,i) + FDn(:,i);

end
end
```

Appendix 10 – SegCenters.m – Calculate Center of Line Segments

```
function Q = SegCenters(P)
% Returns a m x N matrix Q of the coordinates of the center of N line
% segments as column vectors, given the coordinates of N+1 consecutive
% nodes as column vectors in a m x N+1 matrix P.
% Ex. Q = SegCenters(P)

[m,n] = size(P);
N = n-1;      % Number of line segments

Q = zeros(m,N);     % Initialize Q matrix

% Calculate the center for N segments
for i=1:N
    Q(:,i) = 0.5*(P(:,i) + P(:,i+1));
end
end
```

Appendix 11 – EquilSolver.m – Equilibrium Solver, fmincon Parameters and Constraints Setup

```
function qf = EquilSolver(Body,Lines,q_i)
% Sets up all parameters and constraint equations needed to pass to the
% solving function used to minimize the potential energy of the system and
% return the final (equilibrium) values q of the displacements. Given the
% Body object (Body), the array of the Line objects (Lines) and the array
% of the initial displacements q_i. q_i must contain the Body C.O.M.
% coords., followed by the XYZ Euler angles, then the coords of
consecutive
% nodes for each mooring line. For n mooring lines each of N segments:
% q_i = [x;y;z;a;b;g;p10x;p10y;p10z;...;p1Nx;p1Ny;p1Nz;p20x;p20y;p20z;...;
% pijx;pijy;pijz;...;pnNx;pnNy;pnNz]
% where j goes from 1 to N, and then i from 1 to n.
% Ex. q = EquilSolver(Body,Lines,q_i)

N = Lines(1).N;     % Number of line segments per mooring line

nl = length(Lines); % Number of mooring lines

% Bound Constraints
% For the ground constraint, set the lower bound of the Z coord. of every
% point to 0. Ex. z >= 0. All other displacements unbounded.
lb = -Inf(size(q_i));
lb(3) = 0;  % Z coord. of Body
% Set lower bound of Z coord. of every line node to 0
for i=9:3:length(lb)
    lb(i) = 0;
end

% Linear Equality Constraints
% Constraints of the form Aeq*x = beq, where Aeq is an m by n matrix, and
% beq is an m by 1 vector. m is the # of constraints and n # of variables.
% Sets up the ground attachment constraints where the last node of each
% mooring line has the same coords. as its ground attachment point.

% Sets up beq vector containing the coords. of the ground attachment
points
beq = [];
% Sets up the Aeq matrix, placing a 1 corresponding to the coord. of the
% last node for each mooring.
Aeq = zeros(3*nl,length(q_i));
for i=1:nl;
    % Adds ground attachment coords. to beg vector
    beq = [beq;Lines(i).pG];
    Aeq(3*i - 2, 4 + 3*(N+1)*i) = 1;    % X coord.
    Aeq(3*i - 1, 5 + 3*(N+1)*i) = 1;    % Y coord.
    Aeq(3*i, 6 + 3*(N+1)*i) = 1;        % Z coord.
end

% Non-linear Constraints
```

```
function [Cineq, Ceq] = NonLinCon(q)
% This function returns the non-linear inequality and equality
constraints.

% Non-linear Inequality Constraints
% Sets up the slack constraint with the change in length of each segment
is
% greater than or equal to zero. Constraints must be of the form
% Cineq(x) <= 0.
% Ex. -|(pi,j+1) - (pi,j)| + Ls <= 0

% Sorts the coords. of the line nodes into column vectors
P = reshape(q(7:length(q)),3,(nl*(N+1)));

Cineq = zeros(nl*N,1);
for k=1:nl
    for j=1:N
        Cineq(j+(N*k-N)) = -norm(P(:,j+(N+1)*(k-1)+1)-P(:,j+(N+1)*(k-1)))
+ Lines(k).Ls;
    end
end

% Non-linear Equality Constraints
% Sets up the Body attachment constraints where the first node of each
% mooring line has the same coords. as its Body attachment point.
% Constraints must be of the form Ceq(x) = 0.
% Ex. pi,0 - c - Txyz*piB = 0.

c = [q(1);q(2);q(3)];       % Body C.O.M. coords
angles = [q(4);q(5);q(6)];  % Euler angles

Ceq = [];
for k=1:nl
    % Coords. of Lines first node
    pi0 = [q(7+3*(N+1)*(k-1));q(8+3*(N+1)*(k-1));q(9+3*(N+1)*(k-1))];

    Ceq = [Ceq; pi0 - c - Txyz(angles)*Lines(k).pB];
end
end

% Objective function
% This function returns the value to be minimized by the solver
function Vt = Vtotal(q)
% This function returns the total potential energy of the system
c = [q(1);q(2);q(3)];       % Body C.O.M. coords
angles = [q(4);q(5);q(6)];  % Euler angles
% Sorts the coords. of the line nodes into column vectors
P = reshape(q(7:length(q)),3,(nl*(N+1)));

Vt = 0;     % Total potential energy
Vt = Vt + Vbody_total(Body,c,angles);   % Potential of Body
for k=1:nl  % Total potential of each mooring line
    Vt = Vt + Vline_total(Lines(k),P(:,1+(N+1)*(k-1):(N+1)*k));
```

```
end
end

% Constrained Nonlinear Optimization solver
% Function used to minimize the potential energy and return the final
% values of the displacements
% Sets up the options for the solver
options = optimset('Algorithm','active-set','Display','iter','TolFun',1e-
6,'MaxFunEvals',500*length(q_i));

[qf,Vf,exitflag,output,lambda,grad,hessian] =
fmincon(@Vtotal,q_i,[],[],Aeq,beq,lb,[],@NonLinCon,options);

end
```

Appendix 12 – PlotSolu.m – Makes a 3D Plot of the Body and Mooring Lines

```
function PlotSolu(q,Body,Lines)
% Makes a 3D plot of the mooring lines and body system, given the vector
of
% displacements q, the Body object, and array of Line objects.
% Ex. PlotSolu(q,Body,Lines)

N = Lines(1).N;     % Number of line segments per mooring line
nl = length(Lines); % Number of mooring lines

c = [q(1);q(2);q(3)];       % Body C.O.M. coords
angles = [q(4);q(5);q(6)];  % Euler angles

% Sorts the coords. of the line nodes into column vectors
P = reshape(q(7:length(q)),3,(nl*(N+1)));

% Sorts the node coords. into matrices, with each row corresponding to a
% mooring line and each column a node.
X = zeros(nl,N+1);  % X node coords.
Y = zeros(nl,N+1);  % Y node coords.
Z = zeros(nl,N+1);  % Z node coords.

hold on
xlabel('X (m)')
ylabel('Y (m)')
zlabel('Z (m)')
axis equal
view(3)
%grid on

for i = 1:nl
    X(i,:) = P(1,1+(N+1)*(i-1):(N+1)*i);
    Y(i,:) = P(2,1+(N+1)*(i-1):(N+1)*i);
    Z(i,:) = P(3,1+(N+1)*(i-1):(N+1)*i);

    plot3(X(i,:),Y(i,:),Z(i,:),'-
o','LineWidth',2,'MarkerFaceColor','none')
end

% Plot Body object
PlotBody3D(Body,c,angles)
```

Appendix 13 – PlotDrag3D.m – Makes a 3D Plot of the Body and Mooring Line Drag Forces

```
function PlotDrag3D(q,Body,Lines)
% Plots the 3D drag force vectors on the Body and mooring line segments,
% given the vector of displacements q, the Body object and array of Line
% objects.
% Ex. PlotDrag3D(q,Body,Lines)

N = Lines(1).N;     % Number of line segments per mooring line
nl = length(Lines); % Number of mooring lines

% Sorts the coords. of the line nodes into column vectors
P = reshape(q(7:length(q)),3,(nl*(N+1)));

% Plot the Body drag force, scaled down by factor of 10^4
quiver3(q(1),q(2),q(3),Body.FD(1),Body.FD(2),Body.FD(3),10^-4,'r')

% Plot the Line drag forces
for i = 1:nl
    Q = SegCenters(P(:,1+(N+1)*(i-1):(N+1)*i));
    % plot tangential drag forces

quiver3(Q(1,:),Q(2,:),Q(3,:),Lines(i).FDt(1,:),Lines(i).FDt(2,:),Lines(i).
FDt(3,:),'r')
    % plot normal drag forces

quiver3(Q(1,:),Q(2,:),Q(3,:),Lines(i).FDn(1,:),Lines(i).FDn(2,:),Lines(i).
FDn(3,:),'r')
end
```

Appendix 14 – SysUpdate.m – Updates System's Drag Forces

```
function [BodyNew, LinesNew] = SysUpdate(Body,Lines,q)
% Returns new Body and Line objects with an updated estimate of the drag
% forces based on the displacements from a previous solution, given the
% Body object, Lines array and the displacements q from a previous
solution.
% Ex. [BodyNew, LinesNew] = SysUpdate(Body,Lines,q)

% Create new Body object using the new displacement and Euler angles
BodyNew =
newBody(Body.M,Body.V,Body.cB,Body.CD,Body.A,Body.Fa,Body.Ta,q(1:3),q(4:6)
);

N = Lines(1).N;      % Number of line segments per mooring line

nl = length(Lines); % Number of mooring lines

% Sorts the previous coords. of the line nodes into column vectors
P = reshape(q(7:length(q)),3,(nl*(N+1)));

% Creates new Line objects using the new displacements of the nodes
for i=1:nl

    LinesNew(i) = newLine(Lines(i).rho_l,Lines(i).L,N,Lines(i).d,...
        Lines(i).K,Lines(i).pB,Lines(i).pG,P(:,1+(N+1)*(i-1):(N+1)*i));
end
end
```

Appendix 15 – SysRefine.m – Updates System's Drag Forces and Doubles the Number of Mooring Line Segments

```
function [BodyNew, LinesNew, qnew] = SysRefine(Body,Lines,q)
% Returns new Body and Line objects with an updated estimate of the drag
% forces based on the displacements from a previous solution and doubles
% the number of segments for each mooring line, and returns the updated
% displacement vector for the new number of segments. Given the Body
object,
% Lines array and the displacements q from a previous solution.
% Ex. [BodyNew, LinesNew, qnew] = SysRefine(Body,Lines,q)

% Create new Body object using the new displacement and Euler angles
BodyNew =
newBody(Body.M,Body.V,Body.cB,Body.CD,Body.A,Body.Fa,Body.Ta,q(1:3),q(4:6)
);

N = Lines(1).N;     % Number of line segments per mooring line

nl = length(Lines); % Number of mooring lines

% Sorts the previous coords. of the line nodes into column vectors
P = reshape(q(7:length(q)),3,(nl*(N+1)));

% Initialize matrix for new node vectors
Pnew = zeros(3,nl*(2*N+1));

% Doubles the number of segments for each line and creates new Line
objects
for i=1:nl

    % Segment centers for each mooring line
    Q = SegCenters(P(:,1+(N+1)*(i-1):(N+1)*i));

    % First node
    Pnew(:,1+(2*N+1)*(i-1)) = P(:,1+(N+1)*(i-1));

    % Place the coords. of the segment centers in-between nodes to Pnew
    for j=1:N

        % Segment centers
        Pnew(:,2*j+(2*N+1)*(i-1)) = Q(:,j);

        % Previous nodes
        Pnew(:,2*j+1+(2*N+1)*(i-1)) = P(:,j+1+(N+1)*(i-1));

    end

    % Create new Line object with new nodes
    LinesNew(i) = newLine(Lines(i).rho_l,Lines(i).L,2*N,Lines(i).d,...
```

```
        Lines(i).K,Lines(i).pB,Lines(i).pG,Pnew(:,1+(2*N+1)*(i-
1):(2*N+1)*i));

end

% Create new displacement vector
qnew = [q(1:6);reshape(Pnew,nl*(2*N+1)*3,1)];

end
```

Appendix 16 – EquilSolver2D.m – 2D Equilibrium Solver, fmincon Parameters and Constraints Setup

```
function [qf,hessian] = EquilSolver2D(Body,Lines,q_i)
% 2D Solver, X-Y plane
% Sets up all parameters and constraint equations needed to pass to the
% solving function used to minimize the potential energy of the system and
% return the final (equilibrium) values q of the displacements. Given the
% Body object (Body), the array of the Line objects (Lines) and the array
% of the initial displacements q_i. q_i must contain the Body C.O.M.
% coords., followed by the Z rotation angle, then the coords of
consecutive
% nodes for each mooring line. For n mooring lines each of N segments:
% q_i = [x;y;gamma;p10x;p10y;...;p1Nx;p1Ny;p20x;p20y;...;pijx;pijy;...;
% pnNx;pnNy;], where j goes from 1 to N, and then i from 1 to n.
% Ex. [q,hessian] = EquilSolver2D(Body,Lines,q_i)

N = Lines(1).N;     % Number of line segments per mooring line

nl = length(Lines); % Number of mooring lines

% Linear Equality Constraints
% Constraints of the form Aeq*x = beq, where Aeq is an m by n matrix, and
% beq is an m by 1 vector. m is the # of constraints and n # of variables.
% Sets up the ground attachment constraints where the last node of each
% mooring line has the same coords. as its ground attachment point.

% Sets up beq vector containing the coords. of the ground attachment
points
beq = [];
% Sets up the Aeq matrix, placing a 1 corresponding to the coord. of the
% last node for each mooring.
Aeq = zeros(2*nl,length(q_i));
for i=1:nl;
    beq = [beq;Lines(i).pG(1:2)];
    Aeq(2*i - 1, 2 + 2*(N+1)*i) = 1;    % x coord.
    Aeq(2*i, 3 + 2*(N+1)*i) = 1;        % y coord.
end

% Non-linear Constraints
function [Cineq, Ceq] = NonLinCon(q)
% This function returns the non-linear inequality and equality
constraints.

% Non-linear Inequality Constraints
% Sets up the slack constraint with the change in length of each segment
is
% greater than or equal to zero. Constraints must be of the form Cineq(x)
<= 0.
% Ex. -|(pi,j+1) - (pi,j)| + Ls <= 0

% Sorts the coords. of the line nodes into column vectors
```

```
P = reshape(q(4:length(q)),2,(nl*(N+1)));

% Cineq = []; % No slack constraint

Cineq = zeros(nl*N,1);
for k=1:nl
    for j=1:N
        Cineq(j+(N*k-N)) = -norm(P(:,j+(N+1)*(k-1)+1)-P(:,j+(N+1)*(k-1)))
+ Lines(k).Ls;
    end
end

% Non-linear Equality Constraints
% Sets up the Body attachment constraints where the first node of each
% mooring line has the same coords. as its Body attachment point.
% Constraints must be of the form Ceq(x) = 0.
% Ex. pi,0 - c - Txyz*piB = 0.

c = [q(1);q(2);0];      % Body C.O.M. coords
angles = [0;0;q(3)];    % Euler angles

Ceq = [];
for k=1:nl
    % Coords. of Lines first node
    pi0 = [q(4+2*(N+1)*(k-1));q(5+2*(N+1)*(k-1));0];

    Ceq = [Ceq; pi0 - c - Txyz(angles)*Lines(k).pB];
end
end

% Objective function
% This function returns the value to be minimized by the solver
function Vt = Vtotal(q)
% This function returns the total potential energy of the system
c = [q(1);q(2);0];      % Body C.O.M. coords
angles = [0;0;q(3)];    % Euler angles
% Sorts the coords. of the line nodes into column vectors
P = reshape(q(4:length(q)),2,(nl*(N+1)));   % 2D
Zcoords = zeros(1,nl*(N+1));
P = [P;Zcoords];    % Augment P with 0 for Z coords.

Vt = 0;     % Total potential energy
Vt = Vt + Vbody_total(Body,c,angles);   % Potential of Body
for k=1:nl  % Total potential of each mooring line
    Vt = Vt + Vline_total(Lines(k),P(:,1+(N+1)*(k-1):(N+1)*k));
end
end

% Constrained Nonlinear Optimization solver
% Function used to minimize the potential energy and return the final
% values of the displacements
% Sets up the options for the solver
```

```
options = optimset('Algorithm','active-set','Display','iter','TolFun',1e-
6,'MaxFunEvals',400*length(q_i));

[qf,Vf,exitflag,output,lambda,grad,hessian] =
fmincon(@Vtotal,q_i,[],[],Aeq,beq,[],[],@NonLinCon,options);

end
```

Appendix 17 – PlotSolu2D.m – Makes a 2D Plot of the Mooring Lines

```
function PlotSolu2D(q,Body,Lines)
% Makes a 2D plot of the mooring lines and body system, given the vector
of
% 2D displacements q, the Body object, and array of Line objects.
% Ex. PlotSolu(q,Body,Lines)

N = Lines(1).N;      % Number of line segments per mooring line
nl = length(Lines); % Number of mooring lines

c = [q(1);q(2);0];       % Body C.O.M. coords
angles = [0;0;q(6)];     % Euler angles

% Sorts the coords. of the line nodes into column vectors
P = reshape(q(4:length(q)),2,(nl*(N+1)));

% Sorts the node coords. into matrices, with each row corresponding to a
% mooring line and each column a node.
X = zeros(nl,N+1);  % X node coords.
Y = zeros(nl,N+1);  % Y node coords.
Z = zeros(nl,N+1);  % Z node coords.

hold on
xlabel('X (m)')
ylabel('Y (m)')
axis equal
%grid on

for i =1:nl
    X(i,:) = P(1,1+(N+1)*(i-1):(N+1)*i);
    Y(i,:) = P(2,1+(N+1)*(i-1):(N+1)*i);

    plot(X(i,:),Y(i,:),'-o','LineWidth',2,'MarkerFaceColor','none')
end
```

Appendix 18 – SegTensions.m – Calculates Tension Forces in Mooring Line Segments

```
function [FTs_mag, FTs_vec] = SegTensions(Line, P)
% Calculates the tension forces in the mooring line segments, given the
% Line object and the coordinates of N+1 consecutive nodes as column
% vectors in a m x N+1 matrix P. Returns a 1 x N matrix FTs_mag of the
% magnitude of the tension force in each segment, and the m x N matrix
% FTs_vec of the tension force vectors pointing in the same direction as
% the segments unit vector.
% Ex. [FTs_mag, FTs_vec] = SegTensions(Line, P)

[m,n] = size(P);

% # of line segments in the mooring line, N must equal n - 1
N = Line.N;

% Initialize the matrices
FTs_mag = zeros(1,N);    % Tension magnitudes
FTs_vec = zeros(m,N);    % Tension vectors
t = zeros(m,N);          % Segment tangent unit vectors

% Calculate tension for each segment
for i = 1:N
    % FTs_mag = Ks*delta_Ls
    FTs_mag(i) = Line.Ks*(norm(P(:,i+1)-P(:,i)) - Line.Ls);
    % Calc. tangent unit vector, t = (P2-P1)/|P2-P1|
    % For this case, the unit vector and therefor the tension vector only
    % points from the lower ordered node to the next higher ordered node
    t(:,i) = (P(:,i+1)-P(:,i))/norm(P(:,i+1)-P(:,i));
    % FTs_vec = FTs_mag*t
    FTs_vec(:,i) = FTs_mag(i)*t(:,i);
end
```

Appendix 19 – Txyz.m – Calculates the Euler XYZ 3D Rotation Transformation Matrix

```
function T = Txyz(angles)
% Computes the 3D rotation transformation matrix (T) using the Euler XYZ
% (or 1-2-3) convention, given the three rotation angles in a vector,
% angles = [alpha; beta; gamma] (radians)
% Ex. T = Txyz(angles)

a = angles(1);  % alpha
b = angles(2);  % beta
g = angles(3);  % gamma

T = zeros(3);
T(1,1) = cos(b)*cos(g);
T(1,2) = -cos(b)*sin(g);
T(1,3) = sin(b);
T(2,1) = sin(a)*sin(b)*cos(g) + cos(a)*sin(g);
T(2,2) = -sin(a)*sin(b)*sin(g) + cos(a)*cos(g);
T(2,3) = -sin(a)*cos(b);
T(3,1) = -cos(a)*sin(b)*cos(g) + sin(a)*sin(g);
T(3,2) = cos(a)*sin(b)*sin(g) + sin(a)*cos(g);
T(3,3) = cos(a)*cos(b);
end
```