Wave and ship motion modeling

Matheus V. Bernat - Ludvig H. Granström

ABSTRACT

This report describes the mathematical modeling of waves as suggested in research literature, as well as a possible way to model ship motion suggested by the authors. Important concepts such as sea state, wave spectrum and lifting force are introduced. The end results of the work are two MATLAB-files; the first file simulates waves given desired wave properties, while the second file places the ship on pre-simulated waves and calculates the motion of the ship on these waves. Videos explaining the demonstration files were also recorded and are published in the YouTube channel of one of the authors.

KEYWORDS

Waves. Ship motion. Mathematical modeling. Sea state.

1 INTRODUCTION

This report describes the knowledge acquired during the six-week summer job done within the LINK-SIC cooperation between Linköping University and UMS Skeldar. In short, the goals of the project were:

- to research the motions of waves and the concept *sea state*;
- to simulate wave heights through time;
- to simulate ship motion generated by wave motion.

The end goal with this research is to study the motion of ships on water in order to facilitate the landing of multi-rotor aerial vehicles on a ship.

2 SEA WAVES

In this section, important concepts about waves and associated equations are described.

Waves are formed due to the influence of winds, astronomical gravitational forces, and floating structures such as ships. They are usually described by their height, length and period [7]. The wave height is defined as the vertical difference between the elevations of a crest and a neighbouring trough. The focus of this report is to describe waves formed by the presence of wind, which is called a *fully developed sea*.

Sea waves can be interpreted as superpositions of many different harmonic waves, each with its own amplitude, frequency and phase, as can be seen from Figure 1 [2].

2.1 Wave models: short vs long-crested sea

There are two main models to describe the type of sea waves: short-crested and long-crested. In a long-crested sea (aka unidirectional), all of the waves are parallel. In a short-crested sea, the waves are multi-directional. Modeling of a short-crested sea is done with a *spreading function*, see page 209 in [3]. The concept of a spreading function is further explained in Section 3. An example of a long and short-crested sea can be seen in Figure 2.

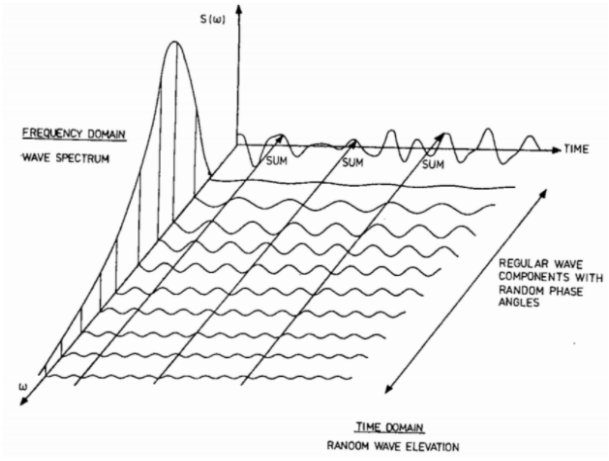

Figure 1: Superposition of waves and spectrum.

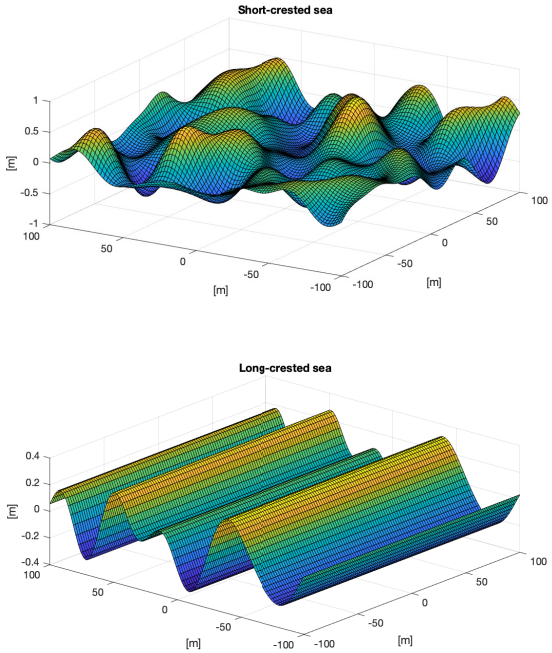

Figure 2: Simulated short-crested and long-crested sea.

2.2 Wave direction

In this report and in the resulting MATLAB-files, the main wave direction is denoted as β. See Figure 3 for details on how the angles are defined relative to the ship.

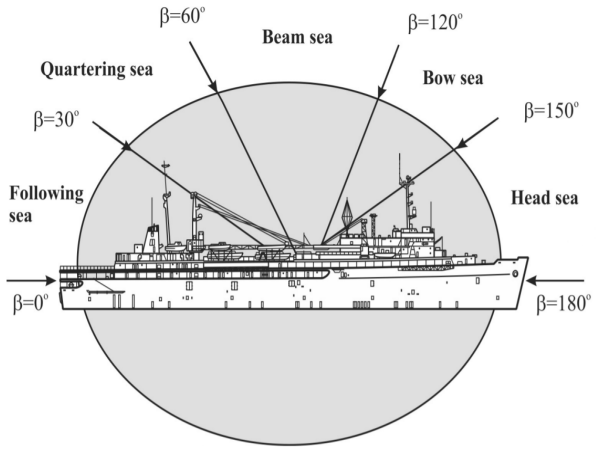

Figure 3: Wave directions in relation to the ship heading.

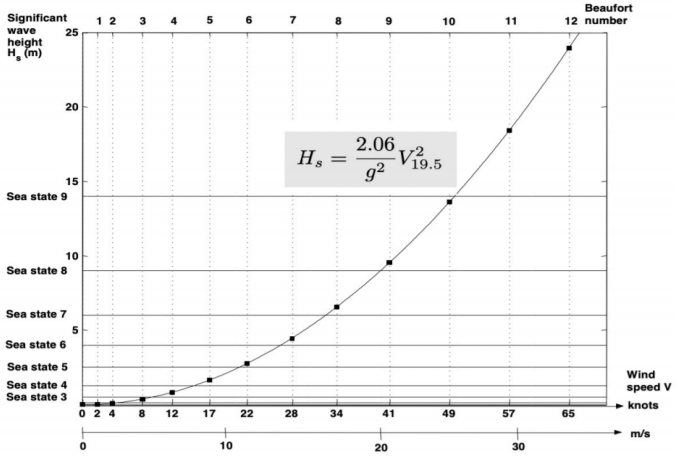

Figure 5: Significant wave heights and sea states.

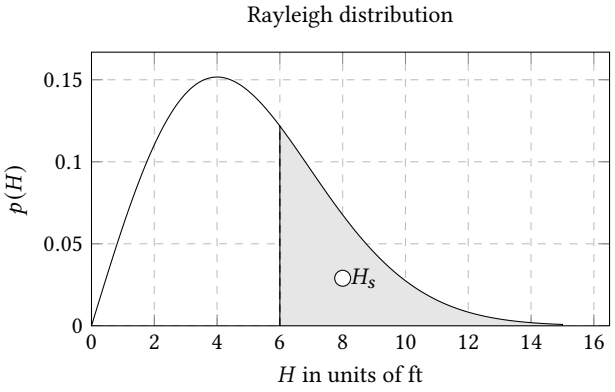

Figure 4: Rayleigh distribution. Shaded area equals 1/3 of the area under the function. The point H_s is the centroid of the shaded area.

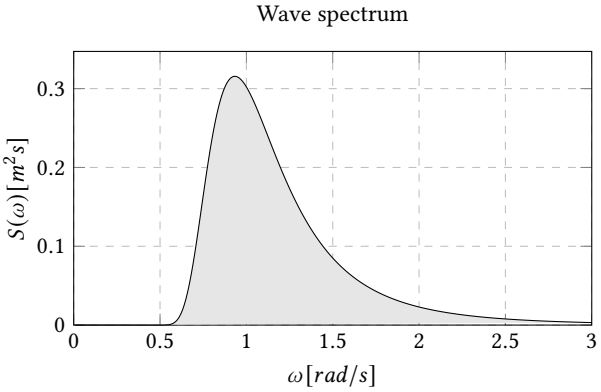

Figure 6: Bretschneider wave spectrum for sea state 3.

2.3 Sea state

In oceanography, sea state is a metric describing the general condition of a body of water on a scale from 0 to 9 where 0 corresponds to a calm, glassy sea and 9 corresponds to a phenomenally agitated sea. The parameters that define a sea state level are the significant wave height H_s and the mean or peak period time of the waves T_s. The significant wave height H_s is defined as the average height of the largest one third of the waves (see page 200 in [3]) and can be derived from the Rayleigh probability density function as shown in Figure 4.

The relation between sea states and different H_s is explained in Figure 5. See page 204 in [3].

2.4 Wave spectrum

The spectrum of a signal is a description of the distribution of the energy in the frequency components that compose the signal. In the case of sea waves, there are predefined spectra, where different spectra are derived based on different assumptions on how the waves are formed and on the surrounding environment, see Section 8.2.2 in [3] and Section 5.4 in [7].

Throughout this report, the *Bretschneider* spectrum will be used, which was developed for the North Atlantic sea, for seas of infinite depth. See page 202 in [3] and page 5.13 in [7]. The spectrum is modeled using

$$S(w) = \frac{0.78}{w^{-5}} e^{\frac{-B}{w^4}}, \tag{1}$$

where $B = \frac{3.11}{H_{1/3}^2}$ and $H_{1/3}$ is a statistical measure of the wave heights depending on the *sea state*, see Table 5.4 and Equation (5.36) in [7].

An example of the spectrum is shown in Figure 6.

3 SPECTRUM TO WAVE ELEVATION

This section describes the process of achieving a wave elevation given a sea state and a spectrum. In the sections below, the formulas for getting the wave elevation in a single point and in a 2D plane are described.

3.1 Wave heights in a point through time

Given a wave spectrum and the parameters required to describe the spectrum, the wave elevation $\xi(t)$ for a short-crested sea is calculated according to (see page 210 in [3])

$$\xi(t) = \sum_{k=1}^{N} \sum_{i=1}^{M} A_k \cos(w_k t + \epsilon_{i,k}). \tag{2}$$

The amplitude A_k is

$$A_k = \sqrt{2S_m(w_k, \mu_i - \beta)\Delta w \Delta \mu}, \tag{3}$$

and

$$S_m(w, \mu) = S(w)M(\mu). \tag{4}$$

The variables in (2), (3) and (4) are:

- $S(w)$ - wave spectrum;
- $M(\mu)$ - spreading function;
- w_k - k^{th} frequency in the discretized wave spectrum;
- β - direction of the *main wave*;
- μ_i - direction of the i^{th} wave relative to the main wave;
- Δw - interval between each discretized frequency in the spectrum;
- $\delta\mu$ - interval between each wave direction;
- $\epsilon_{i,k}$ - random phase in interval $[0, 2\pi]$ for the i^{th} direction of a wave with k^{th} frequency.

The spreading function $M(\mu)$ is defined as:

$$M(\mu) = \begin{cases} \frac{2}{\pi}\cos^2(\mu), & -\frac{\pi}{2} \leq \mu \leq \frac{\pi}{2} \\ 0, & \text{elsewhere.} \end{cases} \tag{5}$$

As can be seen in Figure 7, this is a bell function which decays for values far from the mean, where the *mean* in this case is the main wave direction β. The function has this specific shape due to the assumption that, if the main wave is traveling with direction β, than all other waves are in the interval $[\beta - \frac{\pi}{2}, \beta + \frac{\pi}{2}]$, which is a sensible assumption made in [3]. The constant term $\frac{2}{\pi}$ ensures that $\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} M(\mu)d\mu = 1$, this way the energy is fully described by the spectrum $S(w)$. The spreading function spreads the energy, it neither adds or removes it, see page 209 in [3]. A narrower spreading function would simply disperse more of the energy in the near proximity of the main wave direction and make the sea model more long-crested.

3.2 Wave heights in a 2D plane through time

In a similar way as in (2), one can extend the equation to find the wave height through time for all points (x, y). To start off with a simple example, we will look at how to extend the sinusoidal timedependent function $\cos(\omega t)$ to also cover one dimension in space, the x-axis for instance. This can be done by extending the equation from $\cos(\omega t)$ to $\cos(kx - \omega t)$ [6], where k is the wave number described by

$$k = \frac{\omega}{v} = \frac{2\pi}{\lambda}, \tag{6}$$

where λ is the wave length and v is the wave speed.

The extended Equation (2) for wave height in a 2D plane can be expressed as

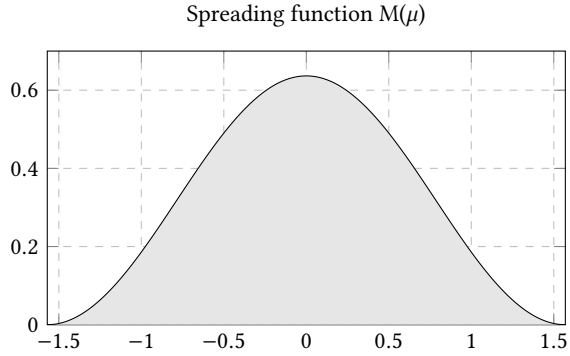

Figure 7: Spreading function M(μ).

$$\xi(x, y, t) = \sum_{k=1}^{N} \sum_{i=1}^{M} A_k \cos(k(x\cos(\mu_i - \beta) + y\sin(\mu_i - \beta)) - w_k t + \epsilon_{i,k}), \tag{7}$$

where the amplitude A_k is

$$A_k = \sqrt{2S_m(w_k, \mu_i - \beta)\Delta w \Delta \mu}. \tag{8}$$

If the wave speed and wave length are unknown, k can be approximated to $k = \frac{\omega^2}{g}$ for a deep sea, $\omega^2 = gk\tanh(kh)$ for intermediate depth, and $k = \frac{\omega}{\sqrt{gh}}$ for a shallow sea [6]. The sea is assumed to be deep in all of the simulations provided. The variable h above stands for the depth of the sea.

3.3 Result: simulated waves

Figure 8 shows the waves achieved by getting the wave heights for all (x, y) coordinates in a 2D grid, for different sea states. This is the end result achieved by the research of how to simulate sea waves in Sections 2 and 3.

4 SHIP MOTION IN 6 DEGREES OF FREEDOM

In this section, it is described how the mathematical modeling of the ship motions was done with six degrees of freedom. A general presentation is given first, and further down, more details are given on how the forces and torques on the ship hull are computed.

4.1 State setup

To analyse the motion of a ship on water, the ship is described in six degrees of freedom. There are three degrees of freedom for translation, and three for rotation, see Figure 9. Two different right-hand coordinate systems are used: an earth-fixed and a body-fixed coordinate system. The origin of the body-fixed coordinate system is located at the ship's center of gravity.

A state space model with twelve variables was created to study the motion of the ship through time. The states are:

$$[x, y, z, v_u, v_v, v_w, \phi, \theta, \psi, \omega_\phi, \omega_\theta, \omega_\psi]^T,$$

where the meaning of the state variables is the following:

- $[x, y, z]$ - position of the ship's center of gravity described in the earth-fixed coordinate system;

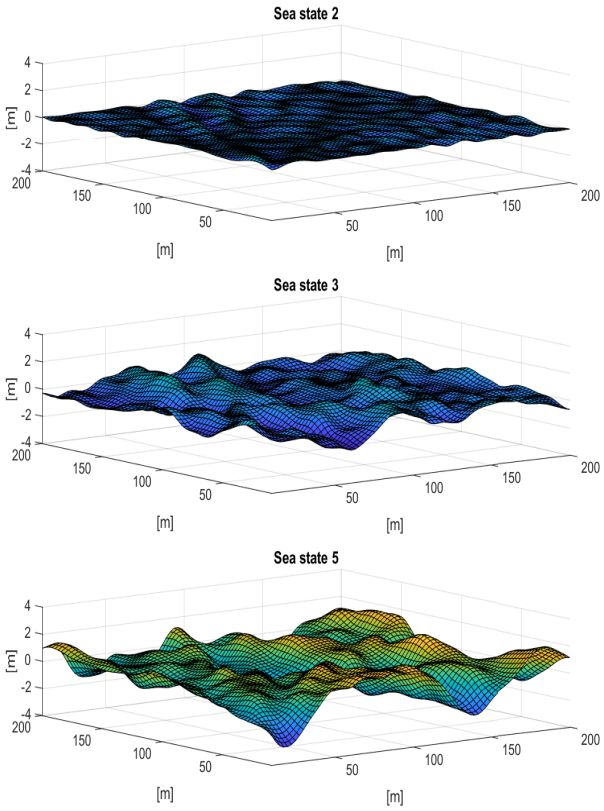

Figure 8: End result: waves of sea states 2, 3 and 5.

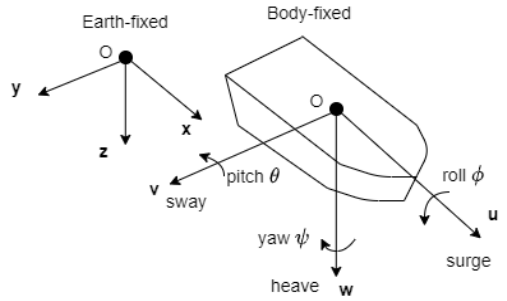

Figure 9: Ship and its six degrees of freedom. The earth-fixed axes are (x, y, z) and the body-fixed axes are (u, v, w).

- $[v_u, v_v, v_w]$ - velocity of the ship's center of gravity described in the body-fixed coordinate system;
- $[\phi, \theta, \psi]$ - Euler angles describing rotation of the body-fixed axes in relation to the earth-fixed axes;
- $[\omega_\phi, \omega_\theta, \omega_\psi]$ - rotational velocity in the body-fixed coordinate system.

4.2 Motion model

In this section, we introduce how the motion model suggested by the authors is related to the motion model suggested by the author of [3].

4.2.1 *States* $[x, y, z]$.

For states $[x, y, z]$, the equations of motion are:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \mathbf{R}^T \begin{pmatrix} v_u \\ v_v \\ v_w \end{pmatrix}. \tag{9}$$

Here, a transformation from the body-fixed velocities $[v_u, v_v, v_w]$ to earth-fixed velocities $[\dot{x}, \dot{y}, \dot{z}]$ is made using the rotation matrix R^T (see Equation (13.7) in [4]), where s represents *sinus* and c represents *cosinus*:

$$\mathbf{R} = \begin{pmatrix} c(\theta)c(\psi) & c(\theta)s(\psi) & -s(\theta) \\ s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & s(\phi)c(\theta) \\ c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) & c(\phi)c(\theta) \end{pmatrix}. \tag{10}$$

The rotation matrix R transforms a vector from earth-fixed coordinates to the ship's body-fixed coordinates by rotating the vector with the given angles $[\phi, \theta, \psi]$.

4.2.2 *States* $[v_u, v_v, v_w]$.

For states $[v_u, v_v, v_w]$, the equations of motion are:

$$\begin{pmatrix} \dot{v}_u \\ \dot{v}_v \\ \dot{v}_w \end{pmatrix} = \begin{pmatrix} -\frac{C_{d,u}A_u\rho}{m}v_u + \frac{1}{m}F_u \\ -\frac{C_{d,v}A_v\rho}{m}v_v + \frac{1}{m}F_v \\ -\frac{C_{d,w}A_w\rho}{m}v_w + \frac{1}{m}F_w \end{pmatrix} - \begin{pmatrix} \omega_\phi \\ \omega_\theta \\ \omega_\psi \end{pmatrix} \times \begin{pmatrix} v_u \\ v_v \\ v_w \end{pmatrix}. \tag{11}$$

The constants in 11 are:

- C_d - damping constants of velocity;
- $[A_u, A_v, A_w]$ - cross-sectional areas along the ship's u, v and w axes $[m^2]$;
- ρ - water density equal to 998$[\frac{kg}{m^3}]$ [8];
- m - mass of the ship, $[kg]$.

See Section 4.2.6 for details on how the viscous damping force $\frac{C_d A_u \rho}{m}v$ is "linearized", and read more about the Coriolis force in Section 4.3.2 for details on the cross product term.

The $F_{\{u,v,w\}}$ terms are the sum of all buoyant forces and weight expressed in ship-fixed coordinates:

$$\begin{pmatrix} F_u \\ F_v \\ F_w \end{pmatrix} = \mathbf{R} \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix}. \tag{12}$$

For more details on how $F_{\{x,y,z\}}$ are computed, see 4.3.

4.2.3 *States* $[\phi, \theta, \psi]$.

The time derivatives of these states are given by

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \mathbf{T} \begin{pmatrix} \omega_\phi \\ \omega_\theta \\ \omega_\psi \end{pmatrix}, \tag{13}$$

where T in (13) denotes the transformation matrix that gives the angle derivatives $[\dot{\phi}, \dot{\theta}, \dot{\psi}]$ given the rotational velocities $[\omega_\phi, \omega_\theta, \omega_\psi]$ (see Equation (13.9) in [4]):

$$\mathbf{T} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{pmatrix}. \tag{14}$$

4.2.4 *States* $[\omega_\phi, \omega_\theta, \omega_\psi]$.

Finally, the motion equations for the rotational velocities are

$$\begin{pmatrix} \dot{\omega}_\phi \\ \dot{\omega}_\theta \\ \dot{\omega}_\psi \end{pmatrix} = \begin{pmatrix} -\frac{B_u}{I_u}\omega_\phi + \frac{1}{I_u}\tau_u \\ -\frac{B_v}{I_v}\omega_\theta + \frac{1}{I_v}\tau_v \\ -\frac{B_w}{I_w}\omega_\psi + \frac{1}{I_w}\tau_w \end{pmatrix}. \tag{15}$$

The term $-\frac{B_u}{I_u}\omega_\phi$ is the viscous damping along the u-axis, see (22).

The variables $\tau_{\{u,v,w\}}$ denote the sum of all torques acting on the ship hull, and are expressed in ship coordinates. For more details on how these are computed, see Section 4.4.

The constants in (16) are:

- B - torsional viscous damping coefficient $[N \cdot s/m]$;
- $[I_u, I_v, I_w]$ - moments of inertia of the ship along the body-fixed axes u, v and w $[kg \cdot m^2]$.

4.2.5 *Summary.*

In summary, the continuous version of the state update equations suggested by the authors is:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v}_u \\ \dot{v}_v \\ \dot{v}_w \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{\omega}_\phi \\ \dot{\omega}_\theta \\ \dot{\omega}_\psi \end{pmatrix} = \begin{pmatrix} \mathbf{R}^T \begin{pmatrix} v_u \\ v_v \\ v_w \end{pmatrix} \\ -\frac{C_{d,u}A_u\rho}{m}v_u + \frac{1}{m}F_u \\ -\frac{C_{d,v}A_v\rho}{m}v_v + \frac{1}{m}F_v \\ -\frac{C_{d,w}A_w\rho}{m}v_w + \frac{1}{m}F_w \\ \mathbf{T}\begin{pmatrix} \omega_\phi \\ \omega_\theta \\ \omega_\psi \end{pmatrix} \\ -\frac{B_u}{I_u}\omega_\phi + \frac{1}{I_u}\tau_u \\ -\frac{B_v}{I_v}\omega_\theta + \frac{1}{I_v}\tau_v \\ -\frac{B_w}{I_w}\omega_\psi + \frac{1}{I_w}\tau_w \end{pmatrix}. \tag{16}$$

4.2.6 *Thor I. Fossen's motion model.*

The motion model proposed by Fossen in [3] is

$$(M_{RB} + M_A)\dot{v} + (C_{RB} + C_A)v + (D_p + D_v)v + G\eta + g_0 = \tau, \tag{17}$$

where:

- v is the body-fixed velocities and angular velocities;
- M_{RB} is the mass and inertia of the ship
- M_A is the added mass from the movement in water;
- C_{RB} is the rigid-body Coriolis and centripetal matrix due to the rotation;
- C_A is the added mass Coriolis and centripetal matrix due to the rotation;
- D_p is the linear potential damping;
- D_v is the linear viscous damping;
- G is the restoring forces from the sea;
- η is the earth-fixed coordinates and Euler angles;
- g_0 is a term regarding trimming of the hull;
- τ is the forces and torque from the propulsion of the ship and waves.

The added mass terms M_A and C_A, linear potential damping D_p, and trimming constant g_0 will not be considered in the simulations. The restoring forces G are moved into τ, resulting in the model

$$M_{RB}\dot{v} + (C_{RB} + D_v)v = \tau. \tag{18}$$

The approximated motion model of the ship, described in (18) is a spring-damper mass system, with the exception of the Coriolis term and the linear spring force being replaced with a non-linear spring force generated between the hydro-static pressure of the sea and the hull of the ship. Equation (19) shows a 1D example of the motion model in body-fixed z-coordinates

$$m\ddot{z} + c\dot{z} = -F_{nl} + g. \tag{19}$$

In reality the damping of the ship is realised by the radiated waves, more about radiated waves in [5], and viscous damping given by

$$F_d = \frac{1}{2}C_d\rho A_s v^2, \tag{20}$$

where C_d is the drag coefficient dependent on the geometry, and A_s is the cross-sectional area. To keep the model as linear as possible this drag force will be approximated to

$$F_d = cv, c = C_d\rho A_s. \tag{21}$$

Note that "linearizing" this way makes the coefficients of the expression obsolete and they can no longer be related to the physical properties of the ship. Hence, the coefficients can be replaced by a single coefficient c that is chosen freely to achieve the desired dynamics of the ship (see Equation (3.30) in [11]).

Similarly to (19), rotation in 1D can be modeled in the same way

$$I\ddot{\theta} + B\dot{\theta} = \tau. \tag{22}$$

4.3 Forces

In this section, the forces taken into consideration are first introduced, and then an explanation of how they are calculated in the given context is given.

4.3.1 *Hydrostatic lifting force.*

The *hydrostatic lifting force* (also called *buoyant force*) is generated by the hydrostatic pressure acting on the wet part of a submersed object. It is defined as (Equation (4) in [7])

$$\overline{F}_b = \oiint_S \rho g h \overline{n} dS, \tag{23}$$

where ρ is the water density, g is the gravitational acceleration ($g = 9.81\ \frac{m}{s^2}$), h is the distance between the submersed point and the water surface, and $\overline{n}$ is the normal vector to the surface dS, see Section 2.2 in [7].

4.3.2 *Coriolis force.*

The Coriolis force is an inertial force that acts on objects that are in a motion in a rotating reference frame. In this case, the rotating reference frame is the ship, and the objects are the points in the ship hull for which the forces are calculated [1] [9]. The Coriolis force in a point with velocity v in the reference frame,

where the reference frame itself rotates with rotational velocity ω is: $F_{coriolis} = -m \cdot \omega \times v$.

4.3.3 Calculations.
The forces taken into consideration are the forces from the hydrostatic pressure of the sea (described in Section 4.3.1), the weight of the ship and the Coriolis force. Therefore, forces such as the dynamic pressure from the sea, added mass, wind forces and lift force from forward translation are ignored. See page 82 in [3] for a list of the forces that are often taken into consideration when modeling ship motion and see [5] for more information about how to derive the hydrodynamic pressure and added mass.

The 3D model of the ship, in this case in the form of an *stl* file, is made of several triangles (called *faces* in the code) whose normal vectors and areas can be computed. To compute the sum of all forces acting on the ship, the hydrostatic forces applied by the water orthogonally to each one of the triangles are summed up

$$\begin{pmatrix} F_u \\ F_v \\ -F_w \end{pmatrix} = \sum_i \rho g A_i h_i \bar{n}_i + \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix} + \begin{pmatrix} F_{control} \\ 0 \\ 0 \end{pmatrix} \tag{24}$$

where A_i is the area, h_i is the relative height of the water located above the triangle's midpoint and $\overline{n}_i$ is the normal direction of the i^{th} triangle. See Figure 10 for a sketch of how the triangles and normal vectors are set up. $F_{control}$ is an added force to control the forward speed of the ship in the simulations.

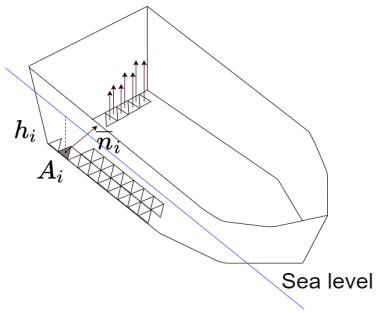

Figure 10: Triangles that represent ship hull and its normal vectors.

4.4 Torques

Here, it is described how the torques $\tau_{\{u,v,w\}}$ named in Section 4.1 are calculated.

Each of the forces acting on the ship hull generates torque relative to the center of gravity. So, given a force $\overline{F}_i$ and a lever $\overline{d}_i$, both expressed in body-fixed coordinates, the three dimensional torque vector is computed as

$$\overline{\tau}_i = (\mathbf{R}\overline{d}_i) \times (\mathbf{R}\overline{F}_i). \tag{25}$$

Therefore, the sum of all torques becomes

$$\overline{\tau}_{net} = \begin{pmatrix} \tau_u \\ \tau_v \\ \tau_w \end{pmatrix} = \sum_i \overline{\tau}_i + \begin{pmatrix} 0 \\ 0 \\ \tau_{control} \end{pmatrix} \tag{26}$$

where $\tau_{control}$ is a moment added to the simulation to control and maintain the reference yaw angle of the ship.

4.5 NED coordinate system in MATLAB

The standard coordinate system for marine vessels engineering is NED (north-east-down), while MATLAB defines its coordinate system as NWU (north-west-up) with both NED and NWU being right-hand coordinate systems. To go from NWU to NED, it is enough to roll 180 degrees along the x-axis, which corresponds to inverting the signs of the y-axis and the z-axis. Finally, to adapt to MATLAB's coordinate system, the sign of the values along the z-axis are inverted, so the plot is in the coordinate system NEU (north-east-up).

4.6 Result: Simulated ship

See Figures 11 and 12 for a snapshot of a ship when on sea state 3, and the inherent plot of the 12 states through time.

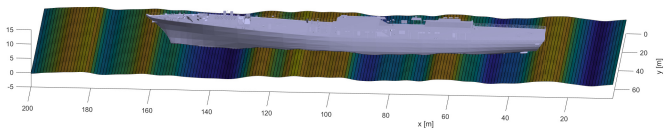

Figure 11: Snapshot of ship on waves of sea state 3.

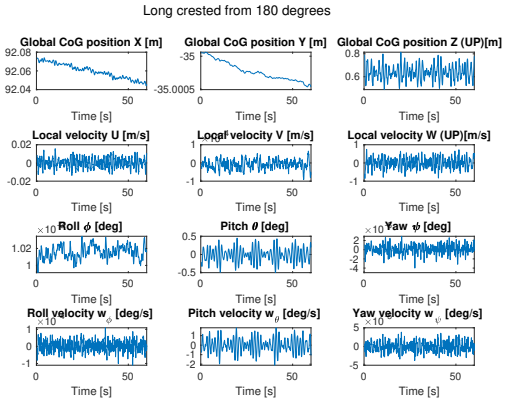

Figure 12: Plot of the states through time.

As can be seen in Figure 12, the global x and y positions of the center of gravity change minimally while the global z position oscillates with greater amplitude, which is expected when the height of the waves varies and the waves are meeting the ship from the front. It is also clear that the roll and that the yaw angles should remain around 0 and that the pitch angle should vary with greater amplitude as the waves meet the ship from the front.

Figure 13 shows the states through time for the helipad instead of the ship's center of gravity. This is achieved by calculating the

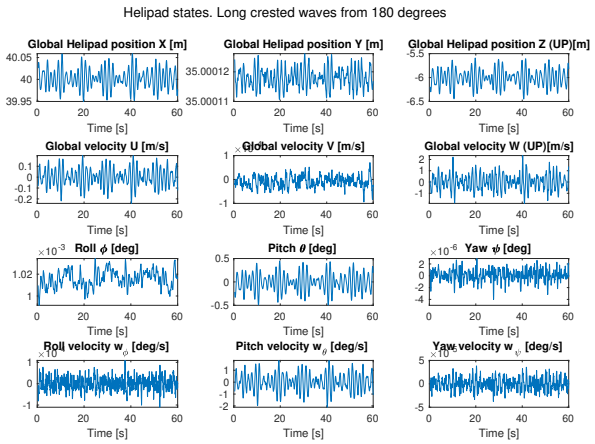

Figure 13: Plot of the helipad states through time.

vector from the helipad to the center of gravity ($\overline{PQ}$) and rotating it with the rotation matrix $\mathbf{R}^T$ to get the new position, and the helipad velocity is calculated by $\overline{v}_Q = \overline{v}_P + \overline{\omega} \times \overline{PQ}$.

5 DISCUSSION

The motion model developed by the authors has been shown to be a simpler version of the model presented by Fossen in [3], as some forces and resulting torques are neglected. Judging by the visual results of the motion of the ship on the simulated waves, the motion model seems to be realistic in the authors' opinion. See the appendix for more details and links to simulation videos.

Some possible improvements of the current model are for instance to:

- implement the effect of added mass (see page 3.9 in [7] and [10]);
- research the effects of the change of the center of buoyancy when the ship pitches or rolls (see Chapter 2 in [7] and Equation (1.28) in [5]) and coupling of states (see Section 1.1 in [5]);
- specify that the force that pushes the ship forward comes from the propellers, which will generate a torque, and limit the force generated by the propellers to a realistic limit;
- implement dynamic pressure from water (see [5]).

REFERENCES

[1] [n.d.]. *Coriolis force.* Retrieved July 28, 2021 from https://en.wikipedia.org/wiki/Coriolis_force

[2] O. Faltinsen. 1990. *Sea Loads on Ships and Offshore.*

[3] Thor I. Fossen. 2011. *Handbook of Marine Craft Hydrodynamics and Motion.* Wiley.

[4] Fredrik Gustafsson. 2018. *Statistical Sensor Fusion.* Studentlitteratur.

[5] Nikolai Kornev. [n.d.]. *Ship dynamics in waves.* Retrieved July 27, 2021 from https://www.lemos.uni-rostock.de/storages/uni-rostock/Alle_MSF/Lemos/Lehre/Sommersemester/Dynamik_ST_II/STII_Ship_dynamics_in_waves.pdf

[6] MIT. [n.d.]. *Water waves.* Retrieved July 2, 2021 from http://web.mit.edu/13.021/demos/lectures/lecture19.pdf

[7] Anders Rosén. 2021. *Introduction to Ship Stability and Seakeeping.* KTH.

[8] U.S. Geological survey. [n.d.]. *Water density.* Retrieved June 23, 2021 from https://www.usgs.gov/special-topic/water-science-school/science/water-density?qt-science_center_objects=0#qt-science_center_objects

[9] Linköpings universitet. [n.d.]. Formelblad, Dynamik. ([n. d.]).

[10] Tran Canh Vinh. [n.d.]. Determination of Added Mass and Inertia Moment of Marine Ships Moving in 6 Degrees of Freedom. ([n. d.]).

[11] Gunnar Steinn Ásgeirsson. 2013. *Hydrodynamic Investigation of Wave Power Buoys.* KTH.

APPENDIX - CODE TUTORIAL

In this appendix, a tutorial of the code produced to materialize the research is given. In short, the code has two components: wave simulation and ship simulation.

Quick overview of results

For a quick overview of the resulting simulations, see the YouTube videos at the following links: **roll test**, **pitch test** and **corridor test**.

Get started

Start by cloning the repository from **GitHub** into your computer. The demo-files depend only on MATLAB's official toolboxes and files in the repository. When running the demo-files, remember to first run the section that adds paths to the folders */help-files*, */boat-files* and */wave-files* first as shown in Listing 1.

```
% Add paths to wave-files, help-files
wavesPath = [pwd, '\wave-files'];
helpFilesPath = [pwd, '\help-files'];
boatPath = [pwd, '\boat-files'];
addpath(wavesPath);
addpath(helpFilesPath);
addpath(boatPath);
```

Listing 1: Adding paths.

Wave simulation

The demo-file for wave simulation is *demoCreateWaves.m*. See in Listing 2 an example of how to simulate a wave of sea state 6 over a grid with sides of 100 and 300 meters, over a time of 500 seconds. Properties of the wished waves are set through lines 9 to 14, then the waves are simulated in line 16 in the file *simulateWaves.m*, and finally the simulated waves are saved in a *mat* file in line 31. This *mat* file will be used to simulate the motion of a ship when traveling on these waves.

```
%% Wave #1: Sea state 6 wave coming from bow (front)
% ------- Use wave with properties below
% Sea state:        6
% Wave type (beta): long-crested (unidirectional)
% Wave angle:       180 degrees
% Grid:             200x100
% Time:             0:0.2:500
% Relative speed    0 m/s

wavesStruct.seaState = 6;
wavesStruct.beta     = pi;
wavesStruct.xVec     = linspace(0, 299, 300);
wavesStruct.yVec     = linspace(0, 99, 100);
wavesStruct.Ts       = 0.2;
wavesStruct.tVec     = 0:wavesStruct.Ts:500;
wavesStruct.U        = 0;

wavesStruct.waves = simulateWaves(wavesStruct.seaState, ...
                                  wavesStruct.xVec, wavesStruct.yVec, ...
                                  wavesStruct.beta, wavesStruct.tVec,wavesStruct.U);
wavesStruct.waveType = 'long';

wavesStruct.displayName = [
   'Waves with properties: ', ...
   '\n   -Sea state: ', num2str(wavesStruct.seaState), ...
   '\n   -Significant wave height: ', num2str(getSignificantWaveHeight(wavesStruct.seaState)), ' m' ...
   '\n   -', wavesStruct.waveType, ' crested' ...
   '\n   -Main wave direction (beta): ', num2str(wavesStruct.beta), ' rad', ...
   '\n   -xVec: ', num2str(wavesStruct.xVec(1)), ':', num2str(1), ':', num2str(wavesStruct.xVec(end)), ' m', ...
   '\n   -yVec: ', num2str(wavesStruct.yVec(1)), ':', num2str(1), ':', num2str(wavesStruct.yVec(end)), ' m', ...
   '\n   -tVec: ', num2str(1), ':', num2str(wavesStruct.Ts), ':', num2str(wavesStruct.tVec(end)), ' s', ...
   '\n   -U: ', num2str(wavesStruct.U), ' m/s', ...
   '\n'];
saveWavesFile(wavesStruct);
```

Listing 2: Simulating wave of sea state 6. From *demoSimulateWaves.m*.

A snapshot of the resulting simulation of this file is seen in Figure 14. For visualization of the waves through all the time, run the sections below in Listing 3.

```
% To visualize waves, run:
waveFile = 'waves_seaState_6_long_beta_3.14_grid_300x100_time_0_0.2_500_U_0.mat';
w = load(waveFile).wavesStruct;
sea = surf(w.xVec, w.yVec, w.waves(:, :, 1));
axis([w.xVec(1) w.xVec(end) w.yVec(1) w.yVec(end) -5 5]);
pause(0.1);
for t=2:length(w.tVec)-1
    sea.ZData = w.waves(:, :, t);
    pause(0.02);
    drawnow;
end
```

Listing 3: Code to visualize waves through time. From *demoSimulateWaves.m*.

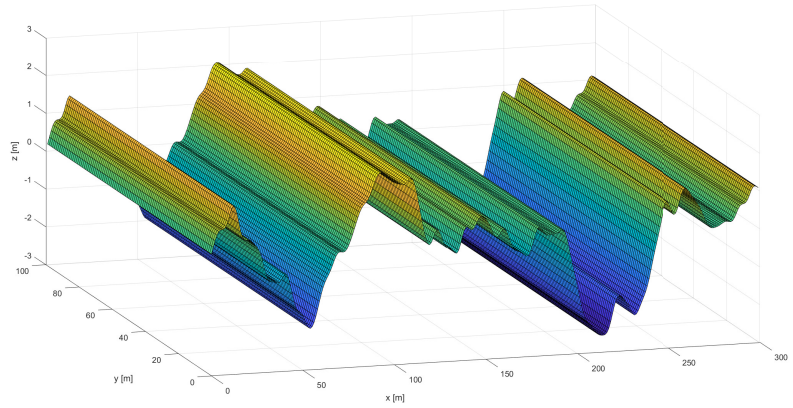

Figure 14: Snapshot of waves of sea state 6.

The simulation of waves achieved with the theoretical results of sections 2 and 3 of the report is implemented in the file *simulateWaves.m*. See the full file in Listing 5.

Ship simulation

Here, it is presented how to simulate the states of a ship through time given some predefined waves. The demo-file for ship simulation is *demoSimulateShip.m*. See in Listing 4 an example of how to simulate a specific ship when on the waves of sea state 6 created previously.

```

%% ------- Create ship with properties below
% Ship:                       HMS Norfolk (hull only)
% Mass:                       2.5e6
% Dimensions:                 137x15x16
% Vertices initial position:  [30 35 5.55]
% Reference velocity along u: 0 [m/s] (should always be 0, the ship speed is set in the wave file)
% Reference yaw:              0 degrees
% CoG offset:                 [-4.77 0.022 -2]
% Position of heliPad         [40 35 6]
shipStruct.file        = 'filteredNorfolkNew.stl';
shipStruct.M           = 2.5e6;
shipStruct.len         = 137;
shipStruct.width       = 15;
shipStruct.height      = 16;
shipStruct.verticesPos = [30 35 5.55];
shipStruct.refSpeedU   = 0;
shipStruct.refYaw      = 0;
shipStruct.cogOffset   = [-4.77 0.022 -2];
shipStruct.helipadPos  = [40 35 6];
%              [v_u v_v v_w phi th psi w_phi w_th w_psi]'
```

```
22 shipStruct.x0 = [0    0   0  0   0  0    0    0    0]; % Initial state values
23
24 %% --------------- Demo #1: Sea state 6 pitch test (waves from the north)
25 % ------- Use wave with properties below
26 % Sea state:        6
27 % Wave type (beta): long-crested (unidirectional)
28 % Wave angle:       180 degrees (from the front)
29 % Grid:             300x100
30 % Time:             0:0.2:500
31 % Ship speed        0
32 isPlot = true;
33 isVisual = true;
34 waveFile = 'waves_seaState_6_long_beta_3.14_grid_300x100_time_0_0.2_500_U_0.mat';
35 [states, face, vert, cogVec] = simulateShip(waveFile, shipStruct, isPlot, isVisual);
```

Listing 4: Simulating ship on waves of sea state 6. From *demoSimulateShip.m*.

A snapshot of the resulting simulation of waves and ship is seen in Figure 15. See also the resulting time plot of the 12 states of the ship in 16. To visualize the full simulation, just set the variable *isVisual* to *true* in line 33 of Listing 4.

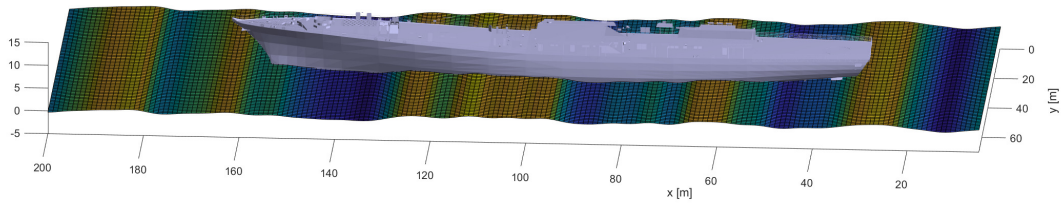

Figure 15: Snapshot of ship on waves of sea state 3.

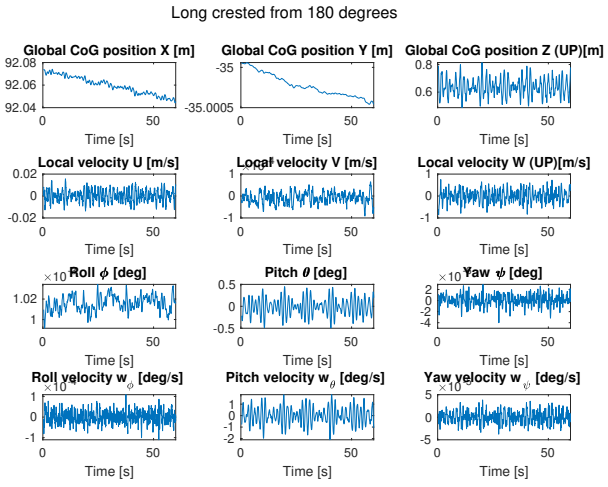

Figure 16: Snapshot of ship on waves of sea state 3.

The simulation of the ship states through time achieved with the state space and motion model in Section 4 is implemented in the file *simulateShip.m*. See the full file in Listing 6.

Full simulation files

See first in Listing 5 the full simulation file *simulateWaves.m* that simulates waves.

```
function waves=simulateWaves(seaState, xVec, yVec, beta, tVec,U , lambda, muVec, dmu)
% SIMULATEWAVES Takes in sea state and plots a wave height. Uses the
% Bretschneider spectrum.
%
% Inputs:
%   - seaState: integer in interval [1, 9].
%   - xVec:     1xN vector. E.g. xVec = linspace(-50, 50, 100).
%   - yVec:     1xN vector. E.g. yVec = linspace(-50, 50, 100).
%   - beta:     direction of main wave in rad.
%   - tVec:     time vector. E.g.: tVec = 0:0.1:50.
%   - U:        speed of the ship [m/s]
%   - muVec:    angle directions vector. E.g.: -pi/2:dmu:pi/2. If muVec=[],
%               then the waves generated will be unidirectional (long-
%               crested).
%   - dmu:      direction interval taken in muVec. Used if muVec ~= [].
%   - lambda:   waveLength. If a sea with infinite depth is assumed, set
%               lambda to [].
%
% Ouput:
%   - waves:    if ~is3d, then size(waves) = (1, length(tVec)), where waves
%               is equal to the wave height in some point in the sea. If
%               is3d, then size(waves) = (length(xVec), length(yVec),
%               length(tVec)), where waves will then represent the wave
%               height of all points (x, y) over a 2D grid defined by xVec
%               and yVec over an interval of time tVec.
tic;
disp('Creating waves...');
g = 9.81;

% Get significant wave height
Hs = getSignificantWaveHeight(seaState);

% Create Bretschneider spectrum given Hs and plot spectrum
dw = 0.1;
wVec = (dw/2:dw:3)';

A = 8.1 * 1e-3 * g^2; % constant, eq 8.54
B = 3.11 / (Hs^2);    % eq 8.55
specType = 1; % Bretschneider (@ Fossen pg 203), from kravspec
S = wavespec(specType, [A, B], wVec, 0);
S(1) = 0; % the first element is NaN for some reason

waves = zeros(length(yVec), length(xVec), length(tVec));

% Get the set of frequencies, directions and phases that will be used to
% generate waves for all points in the grid:
waveFrequencies = zeros(1, length(wVec));
for k=1:length(waveFrequencies)
    waveFrequencies(k) = wVec(k) - dw/2 + dw * rand;
end

if nargin > 7 % Short-crested wave
    waveDirections = zeros(1, length(muVec));
    for i=1:length(waveDirections)
        waveDirections(i) = muVec(i) - dmu/2 + dmu * rand;
    end
    sizeWaveDirections = length(waveDirections);
else
    sizeWaveDirections = 1;
end

wavePhases = zeros(sizeWaveDirections, length(waveFrequencies));
for k=1:length(waveFrequencies)
    for i=1:sizeWaveDirections
        wavePhases(k, i) = 2 * pi * rand;
    end
```

```
end

% Get the wave heights for all coordinates (x,y) for all times in tVec
for yIdx = 1:length(yVec)
    y = yVec(yIdx);
    for xIdx=1:length(xVec)
        x = xVec(xIdx);

        % For each coordinate (x,y), sum the contribution of all the waves
        % to get the final wave amplitude at that point = sumOfWaves.
        sumOfWaves = 0;

        for k=1:length(waveFrequencies)
            w_k = waveFrequencies(k);

            % Long-crested
            if nargin < 8
                if nargin > 6
                    % Lambda is passed as an argument
                    coeff = 2 * pi / lambda;
                else
                    % Infinite depth sea assumed
                    coeff = w_k ^ 2 / g;
                end
                e_k = wavePhases(k);
                amp = sqrt(2 * S(k) * dw);
                wave = amp * cos(coeff * ((x+U*tVec) * cos(-beta) ...
                                + y * sin(-beta))  ...
                                        - w_k * tVec + e_k);
                sumOfWaves = sumOfWaves + wave;

            % Short-crested
            else
                for i=1:length(waveDirections)
                    e_ik = wavePhases(1, k);
                    mu_i = waveDirections(i);

                    amp = sqrt(2 * S(k) * spread(mu_i) * dw * dmu);
                    wave = amp * cos(w_k^2/g * ((x+U*tVec) * cos(mu_i - beta) ...
                                    + y * sin(mu_i - beta))  ...
                                            - w_k * tVec + e_ik);
                    sumOfWaves = sumOfWaves + wave;
                end
            end
        end
        waves(yIdx, xIdx, :) = sumOfWaves;
    end
end
disp('Done creating waves!');
toc;


% ----------------------- Help functions ---------------------------------

function spreadFunction=spread(mu)
    if (mu >= -pi/2 && mu <= pi/2)
        spreadFunction = (2 / pi) * cos(mu)^2;
    else
        spreadFunction = 0;
    end
end
end
```

Listing 5: Full file that simulates waves: *simulateWaves.m*.

And now, the full simulation file *simulateShip.m.* in Listing 6.

```
function [states, faces, vertices, cogVec] = simulateShip(wavesFile, shipStruct, isPlot, isVisual)
% SIMULATESHIP Ship on sea simulation. Given a waves file and a ship,
% simulates 12 states of the ship through time. The states are:
```

```
%   - [x,     y,    z]:    position of the ship's center of gravity (cog)
%                          described in the earth-fixed coordinate system;
%   - [v_u,   v_v,  v_w]:  velocity of the ship's cog described in the body
%                          -fixed coordinate system;
%   - [phi,   th,   psi]:  Euler angles describing rotation of the body-
%                          fixed coordinate system in relation to the earth
%                          -fixed coordinate system;
%   - [w_phi, w_th, w_psi]: rotational velocity in the body-fixed
%                           coordinate system.
% The forces considered are only the hydrostatic (buoyancy) force by the
% water and the ship's weight and the coriolis effect. Ignores: added mass,
% propels force. To calculate the inertia, the ship was assumed to be a
% solid cuboid.
%
% Inputs:
%   - wavesFile:  file containing waves and its properties;
%   - shipStruct: struct containing STL file and other ship properties;
%   - isPlot:     boolean, if true: plots states through time;
%   - isVisual:   boolean, if true: show visualization of simulation in 3D.
% Outputs:
%   - states:    all 12 states simulated through the time vector defined in
%                the wave file;
%   - faces:     ship's faces, only returned to be able to show
%                visualization outside function;
%   - vertices:  ship's vertices, only returned to be able to show
%                visualization outside function;
%   - cogVec:    center of gravity vector, only returned to be able to show
%                visualization outside function.
%
% Constants:
%   - Ax: cross-sectional area of ship along ship's x-axis [m^2];
%   - Ay: cross-sectional area of ship along ship's y-axis [m^2];
%   - Az: cross-sectional area of ship along ship's z-axis [m^2];
%   - Ix: ship's momentum of inertia along x axis;
%   - Iy: ship's momentum of inertia along y axis;
%   - Iz: ship's momentum of inertia along z axis;
%   - Ki: integrational constant for the PI-regulator;
%   - Kp: proportional constant for the PI-regulator;
%   - B:  damping constant for ship's rotational velocity;
%   - Cd: damping constant for ship's velocity;
%   - M:  ship's mass [kg];
%   - ro: water density [kg/m^3];
%   - g:  gravitational acceleration [m/s^2];
%   - l:  length of ship (along x-axis) [m];
%   - w:  width of ship (along y-axis) [m];
%   - h:  height of ship (along z-axis) [m].

tic;

fprintf('\nStarted shipOnSea simulation!\n');

% ------------------------------ Define constants -------------------------
% ----- Non-tunable constants

% Ship dimensions inheritant to HMS Norfolk
len    = shipStruct.len;
width  = shipStruct.width;
height = shipStruct.height;
M      = shipStruct.M;
%Cross sectional area of the submerged hull
Au     = width*0.9 * height*0.33;
Av     = height*0.33 * len;
Aw     = len * width*0.9;

% Offset to ship's center of gravity, set by trial and error
cogOffset = shipStruct.cogOffset;

% Ship's moment of inertia
I  = M/12 * diag([width^2 + height^2, len^2 + height^2, len^2 + width^2]);
```

```
Iu = I(1, 1);
Iv = I(2, 2);
Iw = I(3, 3);

% Physical constants
ro = 997;
g  = 9.81;

% ----- Tunable constants
Kp_force  = 9.5;
Ki_force  = 0.82;
Kp_torque = 3.5;
Ki_torque = 0.5;
K_addedMass = 10;

%Cd        = 5;
%B         = 1e10;

C_du = 5;
C_dv = 5;
C_dw = 5;

B_phi = 1e10/20;
B_th  = 1e10;
B_psi = 1e10;


% ------------------------------ Load waves ------------------------------
disp('1) Loading waves...');
waves = load(wavesFile).wavesStruct.waves;
beta  = load(wavesFile).wavesStruct.beta;
xVec  = load(wavesFile).wavesStruct.xVec;
yVec  = load(wavesFile).wavesStruct.yVec;
tVec  = load(wavesFile).wavesStruct.tVec;
Ts    = load(wavesFile).wavesStruct.Ts;
displayName = load(wavesFile).wavesStruct.displayName;
fprintf(['Waves loaded: \n', displayName]);

% ------------------------------ Load ship's STL file --------------------
disp('2) Loading ship and computing normal vectors to triangles...');
[faces, vertices, ~] = stlreadOwn(shipStruct.file);

vertices = vertices + shipStruct.verticesPos;
% Compute center of gravity (cog) from vertices (geometric center)
cog = [(max(vertices(:, 1)) + min(vertices(:, 1))) / 2, ...
       (max(vertices(:, 2)) + min(vertices(:, 2))) / 2, ...
       (max(vertices(:, 3)) + min(vertices(:, 3))) / 2];
cog = cog + cogOffset;

[facePoints, faceAreas, normals] = calculatePointsAreasNormals(vertices);

disp('Boat loaded!');

% ------------------------------ Define time update model and simulate ---
disp('3) Simulating states through time...');

% ----- Define update matrices A and B (C is arbitrary) and discretize.
%    x y z      v_u       v_v      v_w       phi th psi w_phi   w_th w_psi
A = [0 0 0       1         0        0         0   0   0    0       0    0;
     0 0 0       0         1        0         0   0   0    0       0    0;
     0 0 0       0         0        1         0   0   0    0       0    0;
     0 0 0 -0.5*C_du*ro*Au/M 0        0         0   0   0    0       0    0;
     0 0 0       0 -0.5*C_dv*ro*Av/M 0         0   0   0    0       0    0;
     0 0 0       0         0 -0.5*C_dw*ro*Aw/M 0   0   0    0       0    0;
     0 0 0       0         0        0         0   0   0    1       0    0;
     0 0 0       0         0        0         0   0   0    0       1    0;
     0 0 0       0         0        0         0   0   0    0       0    1;
     0 0 0       0         0        0         0   0   0 -B_phi/(Iu)    0    0;
     0 0 0       0         0        0         0   0   0    0     -B_th/Iv  0;
```

```
     0 0 0       0         0       0         0  0   0    0       0 -B_psi/Iw];

%    F_x/M   F_y/M   F_z/M   Tau_x/Ix   Tau_y/Iy   Tau_z/Iz
B = [  0       0       0        0          0          0;  % x dot
       0       0       0        0          0          0;  % y dot
       0       0       0        0          0          0;  % z dot
       1       0       0        0          0          0;  % v_u dot
       0       1       0        0          0          0;  % v_v dot
       0       0       1        0          0          0;  % v_w fot
       0       0       0        0          0          0;  % phi dot
       0       0       0        0          0          0;  % th dot
       0       0       0        0          0          0;  % psi dot
       0       0       0        1          0          0;  % w_phi dot
       0       0       0        0          1          0;  % w_th dot
       0       0       0        0          0          1]; % w_psi dot
C = [1 0 0 0 0 0 0 0 0 0 0 0;
     0 1 0 0 0 0 0 0 0 0 0 0;
     0 0 1 0 0 0 0 0 0 0 0 0];
sys = c2d(ss(A, B, C, []), Ts);

% ----- Define reference states and initialize other variables
refSpeedU      = getVelMetPerTs(shipStruct.refSpeedU, Ts); % m/Ts
refYaw         = shipStruct.refYaw;
extraUForce    = 0;
extraYawTorque = 0;
cogVec         = zeros(length(tVec), 3);

% ----- Initialize all states and set 1st state
states = zeros(12, length(tVec));
%                    [x       y       z     ],[v_u v_v v_w phi th psi w_phi w_th w_psi]'
states(:, 1) = cat(2,[cog(1)  -cog(2)  -cog(3)],shipStruct.x0)';

% ----- Rotate facePoints & normals according to initial states
phi = states(7, 1); th = states(8, 1); psi = states(9, 1);
facePoints = (R(phi, -th, -psi)' * (facePoints - cog)')' + cog;
normals    = (R(phi, -th, -psi)' * normals')';

% ----- State update for all time steps
for tIdx=1:length(tVec)-1
    cogVec(tIdx, :) = cog;
    % ------- Compute sum of all forces F_net & sum of all torques Tau_net
    F_net = zeros(3, 1);
    Tau_net = zeros(3, 1);
    for nIdx=1:length(normals)
        % Get index of waves that is closest to the evaluated normal
        xIdx = round(facePoints(nIdx, 1));
        yIdx = round(facePoints(nIdx, 2));
        facePointHeight = facePoints(nIdx, 3);
        % Add netto component if wave is above the considered normal
        if waves(yIdx, xIdx, tIdx) > facePointHeight
            % Force
            h = waves(yIdx, xIdx, tIdx) - facePointHeight;
            volume = h * faceAreas(nIdx);
            F_buoy = (ro * g * volume) * normals(nIdx, :)';
            F_net = F_net + F_buoy;
            % Torque
            lever = (facePoints(nIdx, :) - cog)';
            phi = states(7, tIdx); th = states(8, tIdx); psi = states(9, tIdx);
            Tau = cross(R(phi, -th, -psi) * lever, R(phi, -th, -psi) * F_buoy);
            Tau_net = Tau_net + Tau;
        end
    end

    % ------- Set up input
    phi = states(7, tIdx); th = states(8, tIdx); psi = states(9, tIdx);
    v_u = states(4, tIdx);
    extraUForce = Ki_force * extraUForce + pRegulator(Kp_force, refSpeedU, v_u);
    addedMass = v_u * K_addedMass;
    forcesInLocalCoord = R(phi, -th, -psi) * [F_net(1)/(M + addedMass); -F_net(2)/M; -F_net(3)/M + g] ...
```

```
                        + [extraUForce; 0; 0];

    extraYawTorque = Ki_torque * extraYawTorque + pRegulator(Kp_torque, refYaw, psi);
    torqueInLocalCoord = [Tau_net(1)/Iu; -Tau_net(2)/Iv; -Tau_net(3)/Iw] ...
                        + [0; 0; extraYawTorque];

    inputs = [forcesInLocalCoord' torqueInLocalCoord']';

    % ------- Time-update
    v_u= states(4, tIdx); v_v= states(5, tIdx); v_w= states(6, tIdx);
    w_phi = states(10, tIdx); w_th = states(11, tIdx); w_psi = states(12, tIdx);
    velInGlobalCoord = R(phi, th, psi)' * [v_u; v_v; v_w];
    rotVelDerivative = T(phi, th) * [w_phi; w_th; w_psi];

    coriolisV = cross([w_phi, w_th, w_psi]',[v_u, v_v, v_w]');

    % Make time-update as below due to non-linearities in the A matrix
    states(:, tIdx+1) = [states(1,tIdx) + sys.A(1,4) * velInGlobalCoord(1);
                        states(2,tIdx) + sys.A(2,5) * velInGlobalCoord(2);
                        states(3,tIdx) + sys.A(3,6) * velInGlobalCoord(3);
                        sys.A(4,4) * states(4,tIdx) - Ts * coriolisV(1);
                        sys.A(5,5) * states(5,tIdx) - Ts * coriolisV(2);
                        sys.A(6,6) * states(6,tIdx) - Ts * coriolisV(3);
                        states(7,tIdx) + sys.A(7,10) * rotVelDerivative(1);
                        states(8,tIdx) + sys.A(8,11) * rotVelDerivative(2);
                        states(9,tIdx) + sys.A(9,12) * rotVelDerivative(3);
                        sys.A(10,10) * states(10,tIdx);
                        sys.A(11,11) * states(11,tIdx);
                        sys.A(12,12) * states(12,tIdx)] + sys.B * inputs;

    % ------- Update ship hull
    deltaX = states(1, tIdx+1) - states(1, tIdx);
    deltaY = states(2, tIdx+1) - states(2, tIdx);
    deltaZ = states(3, tIdx+1) - states(3, tIdx);

    deltaPhi = states(7, tIdx+1) - states(7, tIdx);
    deltaTh = states(8, tIdx+1) - states(8, tIdx);
    deltaPsi = states(9, tIdx+1) - states(9, tIdx);

    % Rotate facepoints and normals with R'
    facePoints = (R(deltaPhi, -deltaTh, -deltaPsi)' * (facePoints - cog)')' ...
                  + cog + [deltaX -deltaY -deltaZ];
    normals = (R(deltaPhi, -deltaTh, -deltaPsi)' * normals')';
    cog = cog + [deltaX -deltaY -deltaZ];
end
disp('Simulation done!');
toc;

% ------------------------------ Plot states through time ----------------
if isPlot
    plotShipStates(states, tVec, Ts, beta);
    plotHelipadStates(states, tVec, Ts, beta, shipStruct.helipadPos);
end
% ------------------------------ Visualize simulation in 3D --------------
if isVisual
    visualizeSimulation(states, waves, xVec, yVec, tVec, faces, vertices, cogVec);
end

%% Help functions
function [facePoints, faceAreas, normals] = calculatePointsAreasNormals(V)
    j = 1;
    for i=1:3:length(V)-2
        facePoints(j, :) = [mean(V(i:i+2,1)), mean(V(i:i+2, 2)), mean(V(i:i+2, 3))];
        faceAreas(j) = areaOfFace(V(i:i+2,1), V(i:i+2, 2), V(i:i+2, 3));
        p0 = V(i, :)';
        p1 = V(i+1, :)';
        p2 = V(i+2, :)';
        n = cross(p0-p1, p0-p2)' ./ norm(cross(p0-p1, p0-p2)');
        normals(j, :) = -n; % The normal of the face
```

```
        j = j + 1;
    end
    facePoints(isnan(facePoints)) = 0;
    normals(isnan(normals)) = 0;
end
function T = T(phi, th)
% Gustafsson, "Statistical Sensor Fusion" 3rd edition
% Eq. (13.9), p. 349
    T = [1 sin(phi)*tan(th) cos(phi)*tan(th);
         0       cos(phi)         -sin(phi);
         0 sin(phi)/cos(th) cos(phi)/cos(th)];
end
function regulation = pRegulator(Kp, refValue, currentValue)
% Simple P-control of speed.
    regulation = Kp * (refValue - currentValue);
end
function velMeterPerTs = getVelMetPerTs(vel, Ts)
% Gets a velocity in m/s and returns a velocity in m/Ts
    velMeterPerTs = vel * Ts;
end
```

Listing 6: Full file that simulates ship on waves: *simulateShip.m*.