Ship manoeuvring

Candidate number: 10032

December 8, 2019

Abstract

This technical report presents a modular ship simulator, in which the main components of a ship have been modelled. The purpose of this simulator is to calculate the trajectory of a ship on the basis of its parameters and to implement a controller to adjust it in order to reach the desired heading.

The proposed simulator has been designed according to software engineering good practices in order to maintain a modular architecture, in which the sub-components may be reused in other simulators.

In order to implement the ship simulator, a bottom-up approach has been adopted: the simpler components have been studied and implemented before the overall system architecture. Moreover, the simulator has been subdivided on the basis of a layered architecture, in which the access to the data, the system business logic (that implements its behaviour) and the visualization of the results (or presentation) are autonomous the ones from the others.

The simulator allows studying the behaviour of the ship by taking into account the uncertainty on the input data and studies the absolute and relative errors (with respect to the desired heading) when the controller correction is applied to the system.

The last part of this report presents some proposals for further development and improvements of the system, by implementing new components or by using standardized protocols and architectures to integrate it in a more complex simulator.

Contents

1 Terminology

1.1 Concepts

- **Controller**: instrument capable of controlling the evolution of the control variables in an automatic control problem;
- **Digital twin**: a set of digital assets that mirror a physical asset. It typically refers to a data-rich 3D model that represents, reacts to, and can cause changes in the Physical Twin[1];
- **Dynamic positioning**: the ability of a ship to change its position (and heading) according to the desired value by using a control system;
- **Heading**: the direction in which the bow of the ship points;
- **Laplace domain**: a domain in which signals and systems are represented by functions of s, where s is the variable of a signal or system after having been transformed using Laplace transform;
- **Laplace transform**: transform that transforms a real function (whose variable is usually time) to a function of a variable s (usually the complex frequency), where s is a complex variable;
- **PID controller**: a standard controller whose behaviour is determined by the proportional, integral and derivative gains;
- **Propeller**: a rotating fan-like ship component used to propel the ship by using the power transmitted by the ship's engine;
- **Rudder**: a ship component use for steering and manoeuvring;
- **Rudder angle**: the angle formed by the rudder with respect to a reference system;
- **Shaft**: rotating component connecting the engine and the propeller in order to transmit the engine's power;
- **Surge (motion)**: translational motion along the longitudinal axis of a ship;
- **Sway (motion)**: translational motion along the transverse axis of a ship;
- **Software engineering**: an engineering branch concerning software development, whose goal is to build efficient and reliable products;
- **Thruster**: a propulsor, usually installed near the bow or near the stern, able to provide a transversal propulsion;
- **Yaw (motion)**: the rotation of a ship about its vertical axis;

1.2 Notation and symbols

- $e(t)$: error in a feedback system;
- K_d: derivative gain;
- K_i: integral gain;
- K_p: proportional gain;
- k_p: propeller constant for positive ω;
- k_n: propeller constant for negative ω;
- L: lift force;

[1] `https://bimdictionary.com/en/digital-twin/1` (As of December 2019)

- D: drag force (if it is a scalar) or linear damping matrix (if it is a matrix);
- M: mass matrix of the ship;
- X^{-1}: if X is a matrix, then X^{-1} is its inverse;
- R: rotation matrix;
- S: the system dynamics in a feedback system, without the PID controller;
- T_0: thrust force, generated by the thruster;
- $u(t)$: output in a feedback system;
- y: the same as $u(t)$ in a feedback system;
- y°: the desired control value in a feedback system;
- v: velocity vector in the body reference frame;
- δ: rudder angle;
- η: position and heading in the world reference frame;
- ω: shaft speed;

1.3 Abbreviations

- **DP**: Dynamic Positioning;
- **PID (cotroller)**: proportional–integral–derivative (controller);
- **RPM**: Revolutions per minute;

2 Introduction

Ship manoeuvring involves the study of different parameters, that contribute to determine a complex behaviour. One of modelling's purposes is to propose models that describe reality in an approximate way, still being able to describe a phenomena of interest accurately. The proposed model is a simplified version of a ship, where only its main elements are considered.

The consequence of such a choice is that it is possible to disregard some variables, that are not consider of primary importance, while selecting other variables, whose contribution is considered more relevant.

Being able to determine the trajectory of a ship on the basis of some conditions may result useful in order to predict the outcome of complex manoeuvres by simulating them before any physical input is given to the system. Moreover, an extended version of the proposed simulation may be used to implement the behaviour of a digital twin of one or multiple ships. The beneficial effects related to simulation include safety improvement and efficiency.

In particular, the task to be solved consists in finding the trajectory of a ship on the basis of certain conditions and in using a PID controller to adjust it, in order to point to a specific direction (heading).

The report presents an overview of the methodology adopted to solve the problem (Section 3), the applied theories (Section 4), which describe the theoretical background on the basis of which the simulator has been built, the results (Section 5), which describe how the simulator has been developed on the basis of the theories, and a discussion on how they can be interpreted (Section 6), which also contains the conclusions. Part of the calculations has been made available in Appendix A, while the code is presented in Appendix B.

2.1 Background and motivation

The proposed problem belongs to a wider problem, that is Dynamic Positioning (DP) [Pinkster, 1971], which considers, among other parameters, the trajectory and the heading of a ship.

This project presents a simple PID controller, even though this approach has several limitations. One of the most important ones is that it is not possible, in a real-world scenario, to realize a PID controller able to maintain a given heading in any situation, since the environmental conditions are not predictable a priori. Moreover, the provided models are ideal models, and real components can behave in a slightly different way in a real-world scenario.

It is also important to take into account that a controller able to maintain the desired heading with small deviations requires more energy with respect to a controller that allows the ship to have a greater deviation from the desired trajectory, adjusting it less frequently.

In order to implement the simulator the following software engineering principles have been considered[2]:

- Separation of concerns and modularity: the software has been subdivided according to the sub-models that compose the system;
- Abstraction: the interface the user has access to hides the implementation details;
- Anticipation of change and generality: this has been obtained by using sub-components that may work independently the ones from the others;
- Incremental development: a bottom-up approach has been used in order to develop the software, so that each component has been built independently from the others.

2.2 Objectives

The aim of the project consists in developing a modular simulator able to calculate the trajectory of a ship on the basis of the ship's parameters and of the ship components' movement through time.

Moreover, the simulator has to contain a controller module to adjust the heading so that it reaches the desired value.

Finally, a goal of the project is to verify whether the simulator behaves consistently from a physical perspective. In particular, it is important to consider the uncertainties related to any measurement and the relative and absolute errors related to the heading correction.

2.3 Research strategy and approach

The proposed approach is a bottom-up approach, that consists in the analysis of simple models of the sub-components of the ship. Then, each sub-component's behaviour is modelled independently from the system's overall behaviour. Finally, the sub-components' interaction is taken into account and the ship model can be realized, as described in detail in Section 4.

As for the simulator's architecture, it has been organized in layers[3]:

- Data access;
- Business logic: implementation of the component's behaviour;
- Presentation: visualization of the data on the basis of the calculations

Each layer relies on the previous layer and a layer is not aware of how the next layers use the computed data.

The data access layer usually relies on a database. However, only a limited amount of data has been provided in this project and the database setup, for the final user, would have required additional time. Anyway, it is possible to take advantage of MATLAB Database Toolbox[4] in case of future developments of the project.

[2] `https://www.d.umn.edu/~gshute/softeng/principles.html` (As of December 2019)

[3] In all the situations in which this approach wouldn't have compromised readability.

[4] `https://it.mathworks.com/products/database.html`

2.4 State of the art and literature

There exist several studies that have been carried out in order to perform DP ([Veksler et al., 2016], [Sørensen, 2011], [Cheng et al., 2019]). They consider ship models with different degrees of complexity and propose different methods in order to implement a controller. For instance, [Veksler et al., 2016] presents the model-predictive control algorithm (MPC), that combines positioning control and thrust allocation into a single algorithm, exploring its advantages and disadvantages. This article also introduces the ship geometry and the definition of the different forces that act on the ship. The proposed model is an extended version of the one presented in this technical report. MPC doesn't rely on artificial intelligence, while other models (such as [Tu et al., 2018]) rely on machine learning techniques.

3 Methodology

The proposed approach relies on a MATLAB implementation, even though the same modular approach may be used to implement a similar simulator in other programming languages, such as Python or Java.

The first part of the project has consisted in studying the ship model from a theoretical point of view, by understanding the specifications and by examining the problem of ship positioning thoroughly.

Then, the ship model has been divided into sub-models, in order to implement a modular and reusable structure. Finally, the code for each sub-module has been written and tested on the basis of the given parameters. All the sub-models can be adapted, so that it is possible to reuse them by considering different input parameters and a ship described differently.

An error analysis has been implemented, in order to understand the consequences of an imprecise interpolation of the given data. Different trajectories have been simulated on the basis of different input values. Moreover, the error relative to the displacement with respect to the objective heading has been studied.

In a real-world scenario, this simulator may result useful for a ship company that wants to perform a preliminary analysis on a simplified model of a ship before proceeding to an in-depth analysis. In order to guarantee the interoperability of this simulator with other simulators or with the physical components of a ship[5] it is important to rely on recognized standards. Examples of such standards are HLA[6] and FMU/FMI[7]. The choice of the standard to use depends on the specific system on which it has to be implemented, since there exist many possible standards or proprietary architectures. For the same reason, it is important to use a standardized communication protocol, such as Google Protocols Buffer[8], Data Distribution Services[9] or ZeroMQ[10]. Similarly to the simulator architectures, it is necessary to choose the protocol (or the protocols) according to the system in which the simulator has to work.

4 Applied theories

The ship has been modelled according to the specifications of the project and of the given dynamic model. In particular, the model has been subdivided into several simpler components:

- Thruster: there exist two thusters, which can rotate in two directions. They generate a force T_0 on the x axis on the basis of the following model:

$$T_0 = \begin{cases} k_p\omega^2, & \omega \geq 0 \\ k_n|\omega|\omega, & \omega < 0 \end{cases} \tag{1}$$

[5] For instance, in case it is used for a digital twin

[6] `https://standards.ieee.org/standard/1516-2010.html` (As of December 2019)

[7] `https://fmi-standard.org/` (As of December 2019)

[8] `https://developers.google.com/protocol-buffers` (As of December 2019)

[9] `https://www.dds-foundation.org/` (As of December 2019)

[10] `https://zeromq.org/` (As of December 2019)

where ω is the shaft speed, expressed in Hz, $k_p = 1.47 \times 10^5 Ns^2$ and $k_n = 1.65 \times 10^5 Ns^2$.

- Shaft: the shaft rotates with a speed in the range $[-132, 132]RPM = [-2.2, 2.2]Hz$
- Propeller and rudder: they can generate a lift force L and a drag force D in case ω is positive. Those forces depend on the rudder angle δ, that varies between -45 degrees and 45 degrees.

The ship is therefore subject to the surge force, to the sway force and to the yaw torque [Pedersen, 2012], as shown in Figure 1[11].

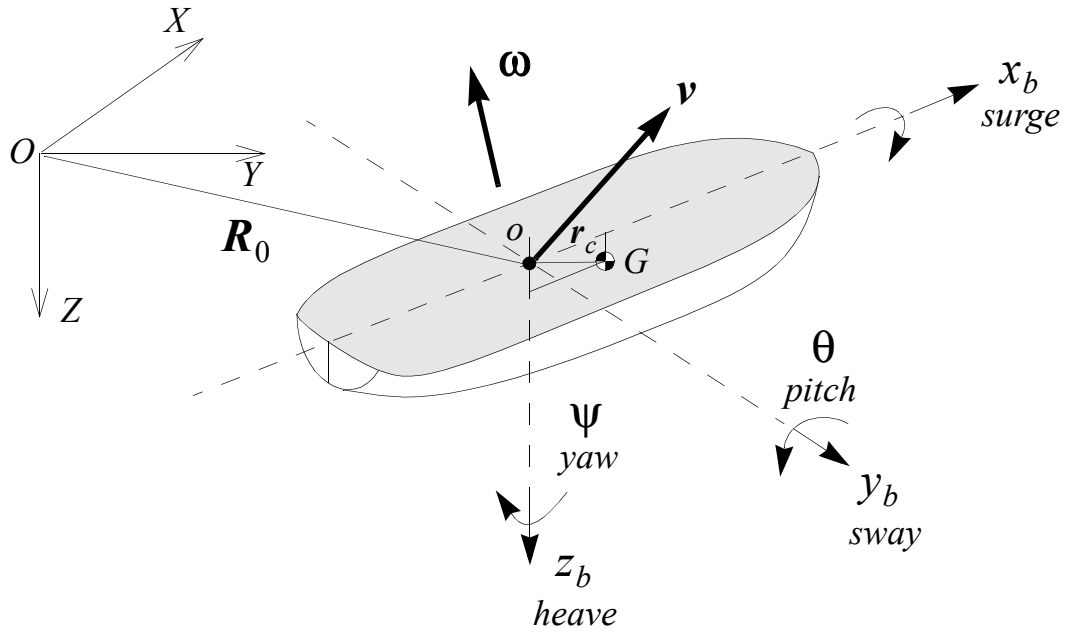

Figure 1: Forces acting on a ship

The ship dynamics model, that considers the described behaviour, is described by the following system of differential equations:

$$\begin{cases} \dot{\eta} = R(\varphi)v \\ M\dot{v} + Dv = F \end{cases} \tag{2}$$

where $\eta = [x\ y\ \varphi]'$ is the position and heading in world frame, $v = [v_{surge}\ v_{sway}\ \dot{\eta}]$ is the velocity vector in the body frame O_B (from the ship perspective) and $R(\varphi)$ is the rotation matrix, defined as:

$$R(\varphi) = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

M is the mass matrix, defined as:

$$M = \begin{bmatrix} 1.1 \times 10^7 & 0 & 0 \\ 0 & 1.1 \times 10^7 & 8.4 \times 10^6 \\ 0 & 8.4 \times 10^6 & 5.8 \times 10^9 \end{bmatrix} \tag{4}$$

D is the linear damping matrix, defined as:

$$D = \begin{bmatrix} 3.0 \times 10^5 & 0 & 0 \\ 0 & 5.5 \times 10^5 & 6.4 \times 10^5 \\ 0 & 6.4 \times 10^5 & 1.2 \times 10^8 \end{bmatrix} \tag{5}$$

[11]This figure presents a more complete model of the ship with respect to the one proposed in this report, since it includes, for instance, vertical forces.

The rotation matrix introduces rotation in the system, the mass matrix takes into account the mass of the ship and the linear damping matrix models the presence of damping forces.

4.1 Data fitting

It has been necessary to introduce an approximation in order to describe the relationship among the rudder angle and the lift and drag forces, since an analytical description of this behaviour is complex and relies on parameters that have not been taken into account in the current project ([Lin et al., 2018]). A simplified model has been obtained after having interpolated the given data using a third degree polynomial.

4.2 Measurement uncertainty

The lift force and the drag force can be calculated on the basis of the rudder angle. Since no explicit relationship has been given, it is necessary to sample the points represented on the given graph and to obtain an interpolating function, as described in the previous section.

Since the resolution of the forces axis is not infinite, it is necessary to consider the uncertainty for the values of the forces. For instance, if a point has a force value between 0.3 and 0.4, it is not possible to determine a value for it that has a precision greater than 0.1.

The proposed approach expresses a point's y-coordinate as $y_P \pm u$ where $2 \cdot u$ is the total uncertainty around the value. Then, the interpolation is performed on the curves obtained by considering the extreme points and their average value.

In particular, three different possibilities have been analyzed:

- The value y_P has been esteemed on the basis of an approximate measurement on the graph;
- The value y_P has been esteemed as the approximate measured value with the addition of a positive uncertainty u ($y_P^{high} = y_P + u$);
- The value y_P has been esteemed as the approximate measured value with the addition of a negative uncertainty $-u$ ($y_P^{low} = y_P - u$).

The results of such choices are presented in the next sections.

4.3 Control

The considered system has a control variable, that is the rudder angle, and several controlled variables[12]. The trajectory of the ship, therefore, depends on how this variable changes through time.

As for the controlled variables (for example the heading of the ship), it is possible to set the desired value and to implement a controller to bring the controlled variable to that value or, at least, closer to that value[13].

An example of controller is the PID controller, defined in [Åström, 1995]. In the ship manoeuvring problem, it is possible, for instance, to introduce a PID controller to bring the ship's heading to 30°, as shown in 5.8.3.

The PID controller has to adjust the heading on the basis of the system output, and it also has to change the system input, therefore it is necessary to implement a feedback system.

A PID controller is defined by:

$$u(t) = K_p \cdot e(t) + K_i \int_0^t e(t)dt + K_d \cdot \frac{de(t)}{dt} \tag{6}$$

[12] As specified in the assumptions section, the other variables are not considered control variables in this model, while they might be control variables in a real-world system.

[13] In a real-world scenario it is not possible to have an error of zero, and in some systems it is not possible to have a zero error even if the behaviour of the system is completely known.

where $u(t)$ is the output of the PID, $e(t)$ is defined as the difference between the desired value y° [14] and the output of the system y, K_p is the proportional gain, K_i is the integral gain and K_d is the derivative gain.

It is also possible to obtain a P controller by setting $K_i = K_d = 0$ or a PI controller by setting $K_d = 0$, for instance.

A generic PID controller in a feedback system is represented in Figure 2. In this representation, S represents the system dynamics without the controller.

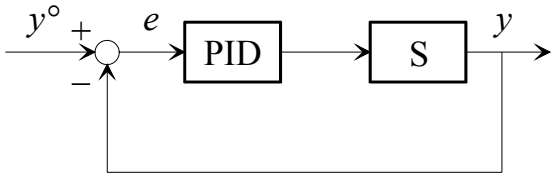

Figure 2: Block diagram of a closed-loop controlled system
Source: http://home.deib.polimi.it/rocco/leonardo/dispensa.pdf, page 183

4.3.1 Controller discretization

Equation 6 refers to the representation of a PID controller if the time is not considered discrete. During the proposed simulation the time is discrete, therefore it is necessary to write it recursively. For a generic iteration i[15]:

$$\begin{gathered} e(i) = y^\circ(i) - y(i) \\ e_{int}(i) = e_{int}(i) + e(i) \\ PID(i) = K_p \cdot e(i) + K_i \cdot e_{int}(i) \cdot dt + K_d \cdot \frac{e(i)-e(i-1)}{dt} \end{gathered} \tag{7}$$

where dt is a finite time step. While in the real world $dt \to 0$, during a simulation dt is a constant small number. The simulation precision increases the closer dt is to 0, but a higher precision requires more calculations and, therefore, more time.

4.4 Analytic form

It is possible to obtain an analytic expression for S and to design a controller on the basis of that expression, for instance by expressing S in the Laplace domain and by writing the equation of the controller accordingly. This approach hasn't been adopted since it presents some problems:

- The interpolation used to obtain the propeller and rudder function introduces an approximation, therefore even finding the analytical form of S would not guarantee an optimal result in a real-world scenario;
- Deriving the analytical function is far more complex than introducing a PID by considering S as a black box;
- It would be necessary to calculate the analytical function in case, for instance, M or D will be changed in the future, while changing the PID controller gains is easier;
- A controller able to adjust the heading perfectly may be more complex than a PID, without significant improvements in the simulator's precision;
- A more complex controller would require the simulator to perform more calculations, slowing it down.

[14] Here, the desired heading.

[15] The equations represent the logic behind discretization. In the real simulator signs are sometimes different to take into account the angle periodicity.

4.5 Computer analysis programs

The development of a ship simulator requires the implementation of a program able to perform matrix operations and to solve ordinary differential equations. Moreover, it is important to present the result of the simulation properly. MATLAB allows to perform those operations, but there exist also other possibilities (for instance, it may be possible to build a web-based application based on Java and JavaScript).

Since the simulator implementation is the central part of the project, a more detailed analysis will be provided in the following sections.

5 Results

This section introduces the description of the code and of the obtained results. A bottom-up approach is proposed, therefore the single components are presented individually, and then the system is presented as a whole.

5.1 Assumptions

The following assumptions have been made:

- It is impossible for the shaft to have a speed ω higher than 132 RPM or lower than -132 RPM[16];
- If the shaft speed is negative, then the lift and drag forces will be equal to zero: this means that only T_0 acts on the ship;
- The functions describing the relationship between the rudder angle and the lift coefficient and between the rudder angle and the drag coefficient can be approximated by using third degree polynomial functions;
- During the trajectory simulation there are no perturbations and the rudder movement can be described by a periodic function;
- There exists only one controller variable, that is the rudder angle. All the other variables are not controllable;
- The ship is symmetrical with respect to its longitudinal axis;
- There are no external forces or disturbances affecting the system.

5.2 Unit conversion and normalization

5.2.1 Frequency

Frequency can be defined in Hertz or in RPM. In order to maintain consistency, all the frequencies are converted in Hertz, since Hertz belongs to the International System of Units (SI)[17].

Although the RPM to Hertz conversion can be performed with only one arithmetic operation (1 Hz = 60 RPM), the function `rpmToHz` has been implemented. The reason is that the program has to be maintained modular, in order to improve code readability and maintainability.

5.2.2 Angle wrapping

Angles can assume any value in $(-\infty, +\infty)$, but it is possible to consider that $\alpha\ [rad] + 2k\pi = \alpha\ [rad]\ \forall k \in \mathbb{Z}$, therefore they can be wrapped in the interval $[0, 2\pi)$. MATLAB's function `wrapTo2Pi` allows to perform this operation.

The reason why it is important to wrap the angles is that, otherwise, there might be the possibility to obtain high angle values, that can slow down the simulation, since it is necessary to allocate more memory to represent them.

5.3 Thruster

The thruster contributes to the dynamics of the ship is modelled by equation 1. In order to model this behaviour, the function `calculateT0` has been implemented.

This function takes k_p, k_n, ω, the minimum allowed shaft speed and the maximum allowed shaft speed as parameters. In particular, the shaft speeds are given in RPM and are converted in Hertz by using the `rpmToHz` function.

[16] In case those limits are not respected, there will be an error.

[17] `https://www.bipm.org/utils/common/pdf/si-brochure/SI-Brochure-9.pdf` (As of December 2019)

The thruster sub-model may be used for a generic thruster described by the same physical model, since the parameters are not hard-coded inside the `calculateT0` function.

The thruster graph is shown in Figure 3.

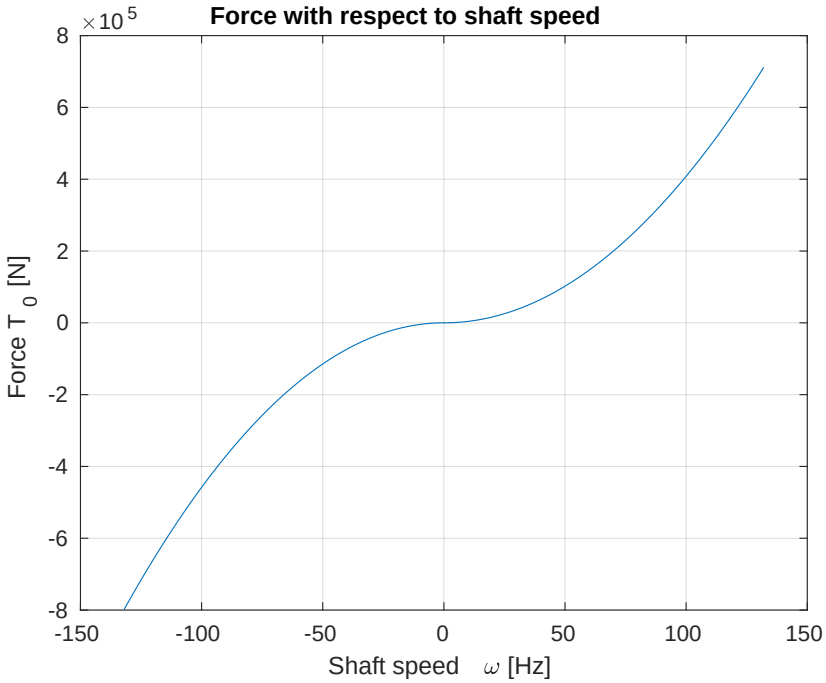

Figure 3: The relationship between T_0 and ω

As expected, the function describing the thruster is monotonically increasing and continuous. The values obtained during the simulation have the expected magnitude, on the basis of Equation 1.

5.4 Propeller and rudder

5.4.1 Interpolation

If $\omega > 0$, the main propeller-rudder pairs are able to generate a lift force L and a drag force D. In particular, the relationship among the rudder angle and the forces, with respect to T_0, is shown in Figure 4.

The provided functions are the result of an approximation because the other coefficients have an extremely low magnitude and are not significant, since their contribution is negligible. MATLAB, however, considers all the coefficients during the simulation.

The reason why those coefficients can be considered negligible is that they are at least 10^{14} smaller than the other coefficients.

Third degree polynomials have been chosen after having tested several options. It is possible to observe that the higher the grade is, the more time the simulator may get, since it has to deal with higher order exponents. Lower degrees, on the other side, did not guarantee a proper fitting for both the functions.

As introduced in section 4.2, it is possible to consider different points, by taking into account the uncertainty related to them. The curves proposed in this section rely on a more precise (but still approximate) measurement of the values presented in Figure 4. The same operation can be proposed for different sets of points, but it is not reported here because it might be redundant.

The system has been analyzed from a physical point of view, and the hypothesis made is that $\delta = 0$ corresponds to the situation in which the drag force is 0, therefore f_D does not refer to the drag force with respect to T_0, but as $1 - \frac{D}{T_0}$, that is the blue curve represented in Figure 4.

The interpolated curves are, therefore:

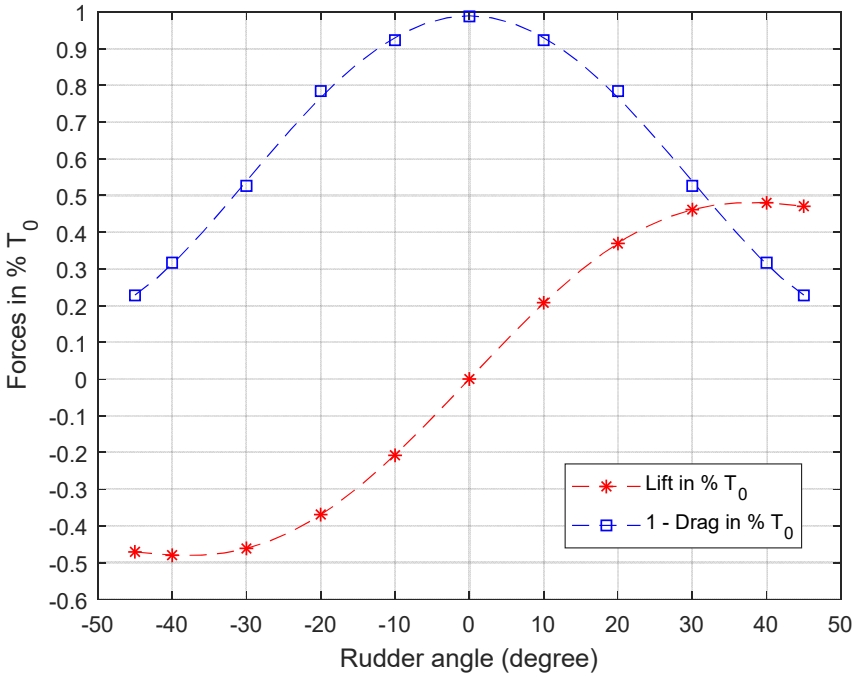

Figure 4: The relationship among the rudder angle δ and the lift and drag forces

$$f_D(\delta) = 1 - \frac{D}{T_0} \approx -3.7376 \cdot 10^{-4}\delta^2 + 0.9324 \tag{8}$$

$$f_L(\delta) = \frac{L}{T_0} \approx -5.0453 \cdot 10^{-6}\delta^3 + 0.0203\delta \tag{9}$$

Since each curve has been interpolated taking into account the accuracy of the given values, three curves can be obtained: one (named *average* in the graph) represents the given values, while the other two curves (*low* for a negative uncertainty and *high* for a positive uncertainty) represent the algebraic sum of the given value and of the uncertainty, that may be positive or negative. Figure 5 represents the interpolation for f_D, while Figure 6 represents the interpolation for f_L.

5.5 Resulting forces

The drag force is along the x axis, while the lift force is along the y axis, therefore it is possible to obtain the resulting forces for both the axis[18]:

$$F_{X_B}(\delta) = F_X(\delta) = 2 \cdot (T_0 - D) = 2 \cdot (T_0 - (1 - f_D(\delta)) \cdot T_0) = 2 \cdot (f_D(\delta) \cdot T_0) \tag{10}$$

$$F_{Y_B}(\delta) = -F_Y(\delta) = 2 \cdot L = 2 \cdot f_L(\delta) \cdot T_0 \tag{11}$$

In particular T_0 and D are parallel, they have an opposite direction and they are applied on the central axis of the ship, that is a rigid body. It is possible, therefore, to calculate their effect on the point O_B as the difference between T_0 and D.

L is not applied on the centre of mass of the ship, but on its stern. Since the two rudders are symmetric with respect to the central axis of the ship and they move synchronously, it is possible to model them as a single rudder aligned with the central axis, therefore the applied force will be L.

The yaw torque is defined as:

$$\tau_{Z_B}(\delta) = \overrightarrow{r} \times \overrightarrow{F}(\delta) = [-41.5, 0, 0] \times [F_X(\delta), -F_Y(\delta), 0] \tag{12}$$

[18]The reason of the 2 factor is that the forces are referred to only one element, while there exist two elements for each type.

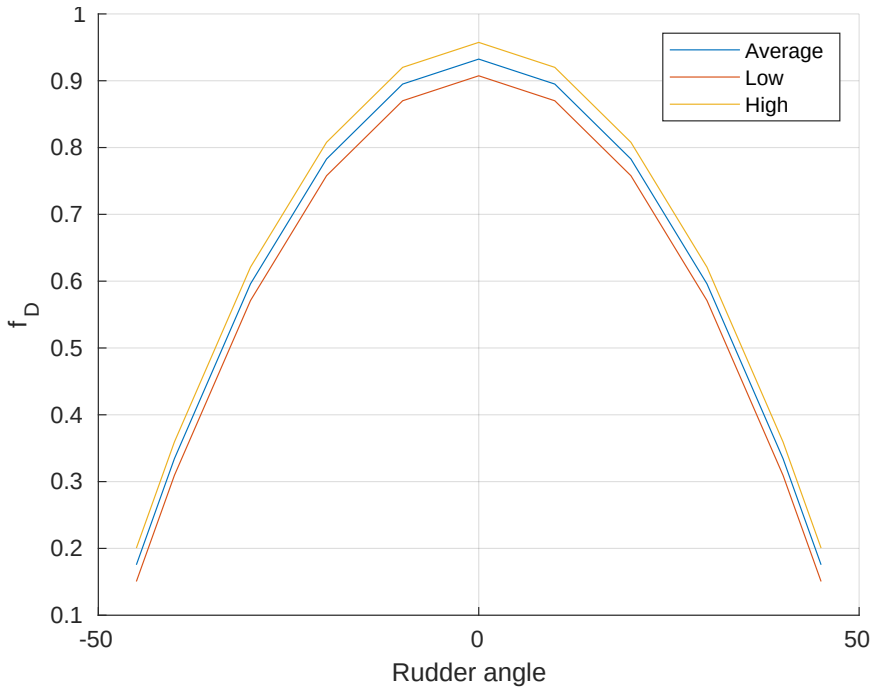

Figure 5: Interpolation of f_D (uncertainty around the average value of 0.05)

where $\times$ represents the cross product and 41.5 is the distance (along x) of the centre of mass from the rudders (that is, the stern of the ship).

The reason why the distance is taken into account is that the lift force's application point is not the centre of mass. Since the ship is symmetric, the application point lies on the same (longitudinal) axis as the centre of mass. Alternatively, it is possible to consider D and L for each component and then using vector algebra to obtain the forces, but the result is the same as the one presented here.

The force, therefore, is $F = [F_{X_B}, F_{Y_B}, \tau_{Z_B}]' = [2 \cdot f_D(\delta) \cdot T_0, 2 \cdot f_L(\delta) \cdot T_0, [-41.5, 0, 0] \times [2 \cdot f_D(\delta) \cdot T_0, 2 \cdot f_L(\delta) \cdot T_0, 0]]'$.

5.6 Code optimization

The project specifications require to have a function `get_force_torque` with parameters ω and δ. In order to calculate the force and the torque, however, it is necessary to calculate the parameters f_L and f_D.

The proposed approach consists in obtaining the polynomial functions 8 and 9. Then, it is possible to evaluate them for different values of δ.

There exist at least three possible approaches to solve the problem:

- Calculate the functions' coefficients and store them using global variables;
- Calculate the functions' coefficient for every simulation step;
- Calculate the functions' coefficients before the differential equations resolution and pass them as parameters in `get_force_torque`.

The three approaches present some issues:

- If global variables are declared, the simulator becomes less modular, since `get_force_torque` has to rely on variables that are neither parameters of the function nor values calculated inside the function;
- If coefficients are calculated for every simulation step, then exactly the same coefficients are calculated many times, increasing the algorithm's complexity and slowing down the simulator;

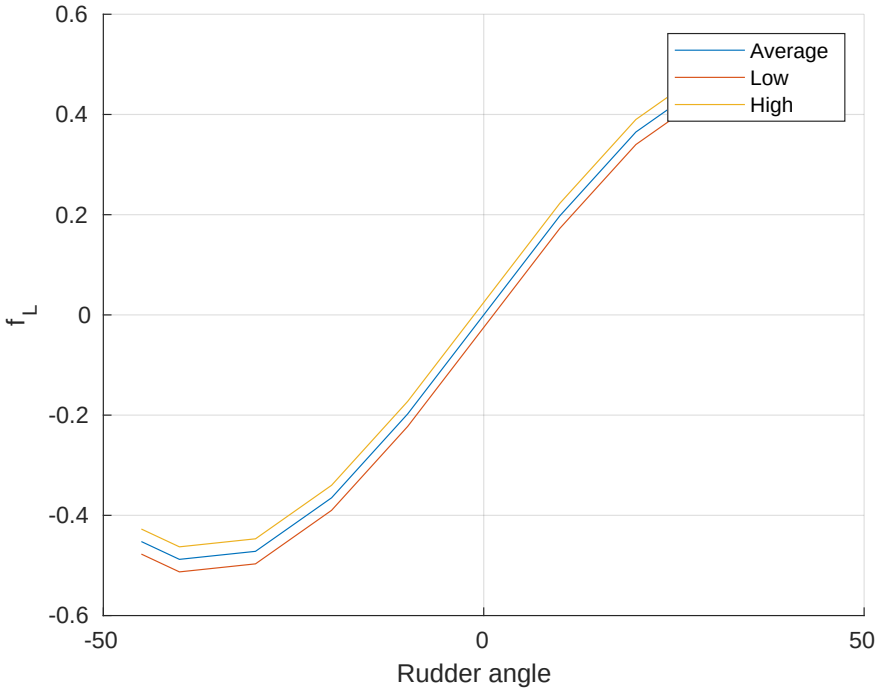

Figure 6: Interpolation of f_L (uncertainty around the average value of 0.05)

- If the coefficients are passed as parameters, then the specification is not completely respected.

The adopted approach consists in maintaining the modularity and calculating the coefficients for every step. Moreover, an alternative proposal has been presented and it relies on passing the coefficients as parameters. The alternative proposal allows to have a reduction in the total simulation time of about 24.5%[19]. The reason is that in the non-optimized version `polyfit` is called 9348 times, while in the optimized version it is called only 6 times.

5.7 Dynamic model in matrix form

It is possible to rewrite the dynamic model of the ship using a set of ODE by considering the equations introduced in Section 4[20]:

$$\begin{bmatrix}\dot{\eta}\\ \dot{v}\end{bmatrix} = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\varphi}\\ \dot{v}_{surge}\\ \dot{v}_{sway}\\ \ddot{\varphi}\end{bmatrix} = \begin{bmatrix}0 & 0 & 0 & & \\ 0 & 0 & 0 & & \boldsymbol{R} \\ 0 & 0 & 0 & & \\ 0 & 0 & 0 & & \\ 0 & 0 & 0 & & \boldsymbol{-M^{-1}D} \\ 0 & 0 & 0 & & \end{bmatrix} \begin{bmatrix}x\\ y\\ \varphi\\ v_{surge}\\ v_{sway}\\ \dot{\varphi}\end{bmatrix} + \begin{bmatrix}0\\ 0\\ 0\\ \\ \boldsymbol{M^{-1}F}\\ \\ \end{bmatrix} \tag{13}$$

In this set of differential equations, v represents the velocity with respect to the ship reference frame, x, y and *phi* represent the position and the heading of the ship with respect to the world reference frame.

The obtained matrix suggests that $\dot{\eta} = [\dot{x}, \dot{y}, \dot{\varphi}]$ depends only on $v = [v_{surge}, v_{sway}, \dot{\varphi}]$. It is important to notice that in matrix R the element in position[21] $(3, 3)$ has to be 1, so that $\dot{\varphi} = \dot{\varphi}$. Moreover, $= [\dot{v_{surge}}, \dot{v_{sway}}, \ddot{\varphi}]$ also depends on $v = [v_{surge}, v_{sway}, \dot{\varphi}]$. It also depends on $M^{-1}F$.

[19]This measurement is only an approximation: its purpose is to show that the optimized version is actually optimized with respect to the other version.

[20]The detailed procedure has been reported in Section A

[21]Indexes start from one.

5.8 Trajectory of the ship

In this section, the trajectory simulation is presented. It is the result of the system of differential equations resolution, therefore the first section analyses the resolution algorithm. Then, the trajectory without any adjustment is presented, while the last subsection presents the result of the correction.

Since the trajectory of the ship depends on the assumptions made when obtaining the data from Figure 4, different results are presented.

5.8.1 Differential equations resolution

MATLAB provides several possibilities to compute the result of a system of Ordinary Differential Equations (ODEs), such as `ode45`. Those methods make use of existing numeric tools to find the approximate solutions of ODEs.

It is also possible to implement those methods (such as Runge-Kutta or Euler) explicitly, without relying on MATLAB's `ode45`.

The proposed approach consists in using `ode45` to find the solutions in case there is no controller, and to implement 4^{th} order Runge-Kutta method to solve the system when the controller is inserted. The reason why Runge-Kutta has been adopted is to maintain the code more readable, without excessively complicate the `systemODE` function. Moreover, in this way the PID implementation is kept separate from the ODE definition, as shown in `computeExample`'s code.

5.8.2 Without correction

The ship movement is simulated for a shaft speed of $\omega = 100RPM$ and a rudder movement, assumed to be periodic, $\delta(t) = 30\sin(0.06t)$. During this first simulation no perturbations nor corrections are considered.

The ship movement can be studied in the world reference system, described by the axis x_W and y_W. In order to maintain the specification orientation of the axis, the x_W axis points upwards and the y_W axis points to the right.

Given the considered periodic signal, the ship moves within the limits of the third quadrant, defined by $\{x_W \leq 0,\ y_W \leq 0\}$ for the majority of the time. During the first part of the simulation, during the transient, the ship turns, going in the other quadrants.

It is possible to observe that the ship trajectory is not a straight line, since the rudder movement is oscillatory. In order to understand this behaviour, both the trajectory graph and the heading are presented.

It is possible to observe that the ship goes from the coordinates $(x_W, y_W) = (0, 0)$ to the coordinates $(-1227, -1013)$[22]. The ship, therefore, covers $\sqrt{1227^2 + 1013^2} = \sqrt{2080898} \simeq 1442\,m$ in 1000 s, that is physically consistent with respect to the modelled object.

Oscillations range from -1.7 to less than -3.3 rad, after the transient. This means that the ship's heading ranges from about -98 degrees to about -190 degrees in about 50 seconds[23].

Fast rotations should be avoided, since some negative consequences may happen. For instance:

- Dizziness for people aboard the ship or using the ship simulator [Bos et al., 2006];
- Waste of time: the ship would take less time to go from a point to another following a straight line;
- Fuel consumption: since the ship takes a longer time and follows a longer path, then more fuel is used.

For this reason, a correction has been introduced, by using a PID controller.

[22] The final coordinates are approximate

[23] Values are not accurate, since the purpose of this analysis is understanding if the provided model is physically consistent. Moreover, these values change slightly during the simulation.

5.8.3 With heading correction

The proposed correction relies on a PID controller, described in Section 2. The goal is to set a heading of 30° and to reduce the trajectory oscillations. Moreover, it is necessary to avoid abrupt changes in the ship trajectory, since they can bring to the negative consequences described in the previous section.

The proposed simulation allows setting a correction able to bring the ship's heading to 30° with respect to the third quadrant (the angle, with respect to the world reference system, is of $180° + 30° = 210°$) because it has been considered a more realistic and common situation. The hypothesis that has been made is that in a typical situation, the PID controller may be activated to oppose, for instance, disturbances, not to completely change the direction of the ship. However, it is possible to change the PID parameters and the desired angle in the function `acquireData`.

One possibility for the PID is to set $K_p = 10$, $K_i = K_d = 0.0001$. Figures 7 and 8 present the possible headings and trajectories of the ship for all the possibilities proposed in 4.2 when the desired angle of 210°. The relative error after 1000 s is always less than 0.1% for an uncertainty of 0.001 on f_D and f_L. The absolute error is always less than 0.004 rad.

Figures 9 and 10 present the situation in which the desired angle is 30°. In this situation, still having an uncertainty of 0.001, the absolute error is always less than 0.003 rad and the relative error is always less than 0.5%[24].

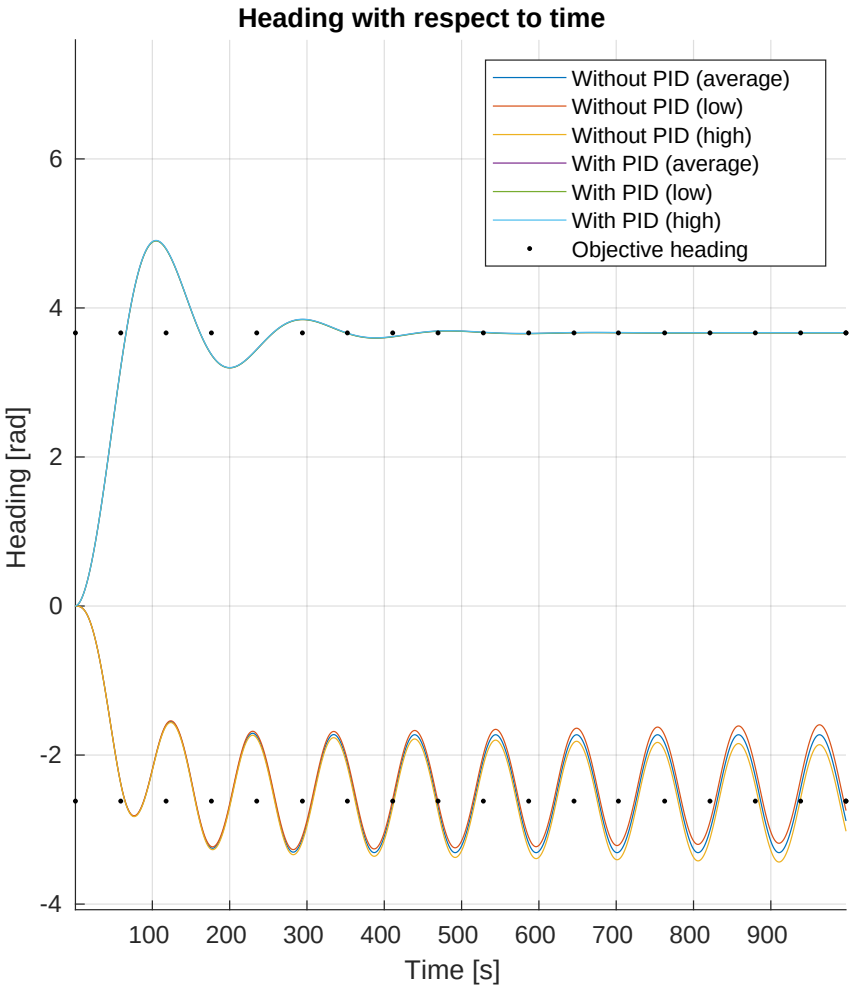

Figure 7: Ship heading with respect to time (desired angle of 210°)

It is important to notice that, since angles are periodic, the angle may be shifted of $2k\pi$, $k \in \mathbb{Z}$

[24]The reason why the absolute error is lower than in the previous case and the relative error is greater is that the relative error is measured on the basis of the desired angle, that here is lower.

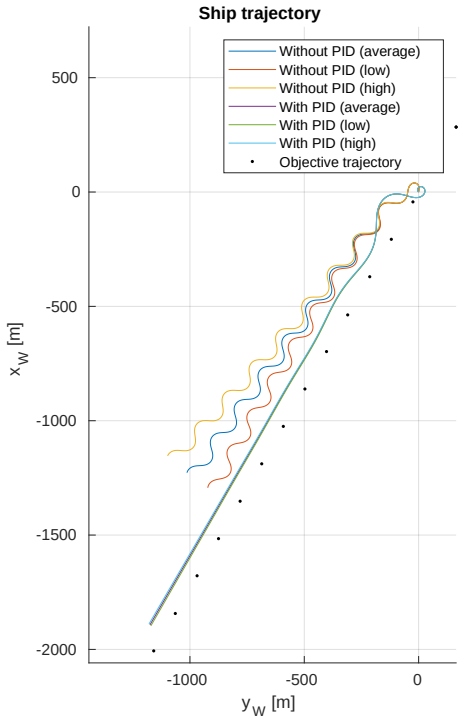

Figure 8: Possible ship trajectories in the world reference frame (desired angle of 210°)

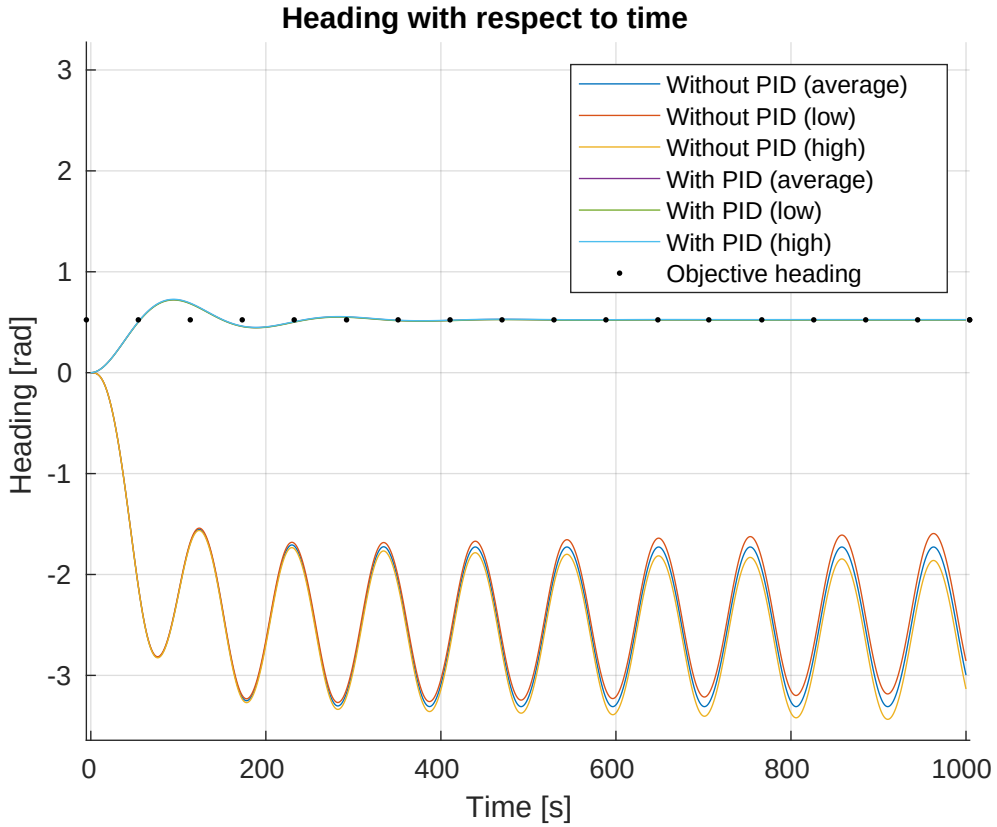

Figure 9: Ship heading with respect to time (desired angle of 30°)

5.9 Overall system

The presented components interact the ones with the others in order to create a more complex system. The code has been structured in such a way that it resembles the organization of the system from a

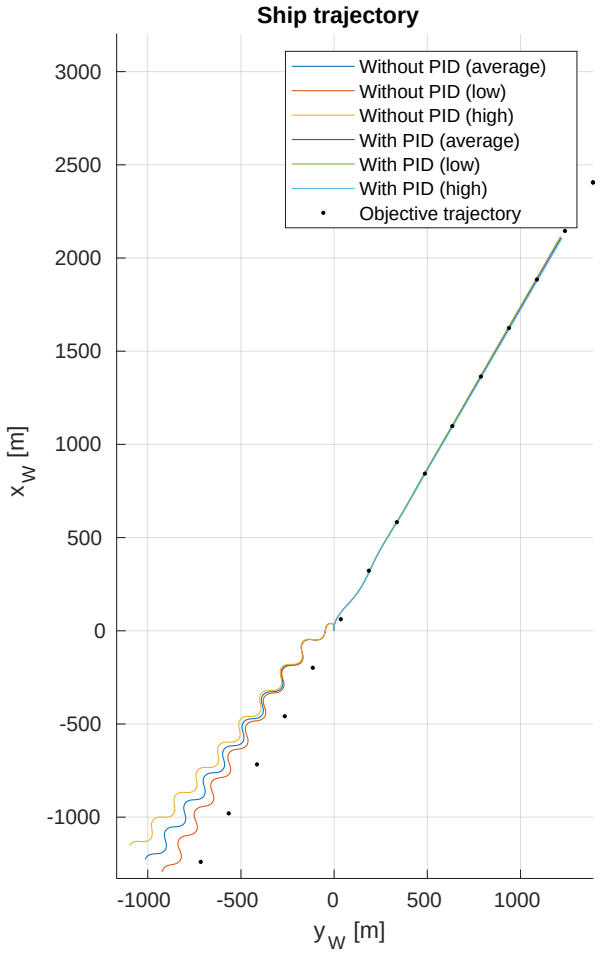

Figure 10: Possible ship trajectories in the world reference frame (desired angle of 30°)

physical point of view.

In a future possible extension of this project, it may be possible to implement it using the object oriented paradigm, so that the ship can be defined as an object, and so the sub-components. In this way the code structure would be also clearer and it would be a one to one mapping with respect to the physical system.

6 Discussion

The project aimed at building a simulator for a simplified ship, constituted by simpler sub-simulators, related to different parts of the ship. Moreover, the purpose of this work was to design a PID controller to regulate the trajectory of the ship on the basis of the desired heading.

The behaviour of the rudder is a source of error, since the model has to be obtained on the basis of real data, while the other proposed models had been provided in an analytical form. For this reason, the simulator takes into account uncertainty on the measured values and interpolates different curves consequently. This report has shown the differences between the curves on the basis of the uncertainty. This source of error could be avoided by having an analytical and physically accurate model or by having a more accurate sampling on the point.

Another possible source of error is the simulation process itself, since it is necessary to use temporal discretization (for instance by using the Runge-Kutta method). In order to have more accurate results, it is possible to reduce the timestep. On the other hand, such a reduction would increase the time necessary to perform the simulation.

The proposed simulator presents some limitations related to the fact that the ship has been simplified. For instance, it may be possible to study the behaviour of water on the basis of the ship movement, or it may be possible to study the vertical movement of the ship.

Notwithstanding the sources of error and the limitations, the simulated trajectory is physically consistent and it behaves as expected in a real-world system with the proposed input. The controller is able to adjust the heading of the ship (at least for the given parameters) without abrupt oscillations and it is able to reach the objective trajectory with a relative error that is always less than 0.5% and an absolute error that is always lower than 0.004 rad (0.229°).

This model may not behave as expected in case of low accuracy of the real data considered as input, since the uncertainty related to them affects the uncertainty related to the heading and to the trajectory.

The results suggest that, in order to design the PID controller properly, the input data should have a low uncertainty. In a real-world scenario, therefore, it would be necessary to measure them by means of an accurate instrument.

The simulator may be further developed by adding other components to the ship, in order to simulate more complex behaviour (related, for instance, to fluidodynamics). A more complex simulator may be used in order to realize a digital twin ([Coraddu et al., 2019], [Danielsen-Haces, 2018], [El Saddik, 2018]).

References

[Bos et al., 2006] Bos, J., Mackinnon, S., and Patterson, A. (2006). Motion sickness symptoms in a ship motion simulator: Effects of inside, outside and no view. *Aviation, space, and environmental medicine*, 76:1111–8.

[Cheng et al., 2019] Cheng, X., Li, G., Skulstad, R., Chen, S., Hildre, H. P., and Zhang, H. (2019). Modeling and analysis of motion data from dynamically positioned vessels for sea state estimation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6644–6650. IEEE.

[Coraddu et al., 2019] Coraddu, A., Oneto, L., Baldi, F., Cipollini, F., Atlar, M., and Savio, S. (2019). Data-driven ship digital twin for estimating the speed loss caused by the marine fouling. *Ocean Engineering*, 186:106063.

[Danielsen-Haces, 2018] Danielsen-Haces, A. (2018). Digital twin development - condition monitoring and simulation comparison for the revolt autonomous model ship.

[El Saddik, 2018] El Saddik, A. (2018). Digital twins: The convergence of multimedia technologies. *IEEE MultiMedia*, 25(2):87–92.

[Lin et al., 2018] Lin, S., Ma, Y., Zheng, W., Zhang, S., Lei, X., and He, Y. (2018). Investigation on rudder hydrodynamics for 470 class yacht. *Proceedings*, 2:308.

[Pedersen, 2012] Pedersen, E. (2012). Bond graph modeling of marine vehicle dynamics. *IFAC Proceedings Volumes*, 45(2):415 – 420. 7th Vienna International Conference on Mathematical Modelling.

[Pinkster, 1971] Pinkster, J. (1971). *Dynamic Positioning of Vessels at Sea: Course held at the Department of Experimental Methods in Mechanics October 1971*, volume 105 of *International Centre for Mechanical Sciences, Courses and Lectures,*. Springer Vienna, Vienna.

[Sørensen, 2011] Sørensen, A. J. (2011). A survey of dynamic positioning control systems. *Annual Reviews in Control*, 35(1):123 – 136.

[Tu et al., 2018] Tu, F., Ge, S. S., Choo, Y. S., and Hang, C. C. (2018). Sea state identification based on vessel motion response learning via multi-layer classifiers. *Ocean Engineering*, 147:318 – 332.

[Veksler et al., 2016] Veksler, A., Johansen, T. A., Borrelli, F., and Realfsen, B. (2016). Dynamic positioning with model predictive control. *IEEE Transactions on Control Systems Technology*, 24(4):1340–1353.

[Åström, 1995] Åström, K. J. K. J. (1995). Pid controllers.

Appendices

A Ship dynamics transformation

Given the ship dynamics (Equation 2), the rotation matrix (Equation 3), the mass matrix (Equation 4) and the linear damping matrix (Equation 5), it is possible to derive an equation in the form $\dot{X}_{6\times1} = A_{6\times6}X_{6\times1} + B_{6\times1}$:

$$\begin{cases} \dot{\eta} = R(\varphi)v \\ \dot{v} + M^{-1}Dv = M^{-1}F \end{cases} \implies \begin{cases} \dot{\eta} = R(\varphi)v \\ \dot{v} = -M^{-1}Dv + M^{-1}F \end{cases} \tag{14}$$

The obtained ODEs show that there is no relationship between $\dot{\eta}$ and η, therefore the submatrix of A formed by the elements $(i,j) : 0 \leq i \leq 2,\ 0 \leq j \leq 2$ will be a null matrix.

It is possible to observe that the same holds for $\dot{v}$ and η. This means that the submatrix of A formed by the elements $(i,j) : 3 \leq i \leq 5,\ 0 \leq j \leq 2$ will also be a null matrix.

It is also possible to observe that the submatrix of A formed by the elements $(i,j) : 0 \leq i \leq 2,\ 3 \leq j \leq 5$ will be $R(\varphi)$ and that the submatrix of A formed by the elements $(i,j) : 3 \leq i \leq 5,\ 3 \leq j \leq 5$ will be $-M^{-1}D$.

Since $M^{-1}F$ is independent of the system variables, it is a constant term, therefore it is part of vector B. In particular, since $\dot{\eta}$ does not depend on any constant term:

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \\ \boldsymbol{M^{-1}F} \\ \\ \end{bmatrix} \tag{15}$$

B Program Code

B.1 Main script (computeExample)

B.1.1 Non optimized version

```
% Script to run the simulation with predefined parameters

% Data acquisition
[control_function, rudderArray, liftArray, dragArray, accuracy, invM, timespan, dt, M
    , D, X0, omega, Kp, Ki, Kd, desiredAngle] = acquireData();

% Business logic (and visualization of the interpolation)
[time, solution, timeL, solutionL, timeH, solutionH, solutionPID, solutionPIDhigh,
    solutionPIDlow] = computeResults(rudderArray, liftArray, dragArray, accuracy,
    invM, D, omega, control_function, timespan, dt, X0, Kp,Ki, Kd, desiredAngle);

% Presentation
visualizeResults(solutionL, solutionH, solution, solutionPIDlow, solutionPIDhigh,
    solutionPID, desiredAngle, timeL, timeH, time);
```

B.1.2 Optimized version

```
% Script to run the simulation with predefined parameters (optimized
% version)

% Data acquisition
[control_function, rudderArray, liftArray, dragArray, accuracy, invM, timespan, dt, M
    , D, X0, omega, Kp, Ki, Kd, desiredAngle] = acquireData();

% Business logic
[fL, fD, fLl, fDl, fLh, fDh, time, solution, timeL, solutionL, timeH, solutionH,
    solutionPID, solutionPIDhigh, solutionPIDlow] = computeResults_OPT(rudderArray,
    liftArray, dragArray, accuracy, invM, D, omega, control_function, timespan, dt,
    X0, Kp,Ki, Kd, desiredAngle);

% Presentation
visualizeResults_OPT(rudderArray, fLl, fLh, fL, fDl, fDh, fD, solutionL, solutionH,
    solution, solutionPIDlow, solutionPIDhigh, solutionPID, desiredAngle, timeL,
    timeH, time);
```

B.2 acquireData

```
function [control_function, rudderArray, liftArray, dragArray, accuracy, invM,
    timespan, dt, M, D, X0, omega, Kp, Ki, Kd, desiredAngle, kp, kn] = acquireData()
```

```
%acquireData returns some data that can be used as the properties of the
%system or as the state at the beginning of the simulation.
%  In a real system, data are acquired by using sensors, for instance, or
%  they are stored in a database. Since this simulator does not access
%  real data, this function replaces the data acquisition part in a real
%  world scenario. In case the system is implemented in a real system,
%  the content of this function should be replaced with some functions
%  that are able to get the real data from the sensors.

omega = rpmToHz(100); % Initial omega = 100 rpm (= 100/60 Hz)

X0 = [0 0 0 0 0 0]'; % Initial state

M = [11 0 0; 0 11 8.4; 0 8.4 5800] * 10^6; % mass matrix
D = [3 0 0; 0 5.5 6.4; 0 6.4 1200] * 10^5; % linear damping matrix

dt = 0.1; % timestep
timespan = 0:dt:1000; % simulation interval [s]

invM = inv(M); % inverse of the linear damping matrix (calculated before to reduce
    computational complexity later)

syms control_signal(t)
control_signal = 30*sin(0.06*t); % rudder movement

control_function = matlabFunction(control_signal);

[rudderArray, liftArray, dragArray, accuracy] = getCoefficientData();

% PID parameters
Kp = 10;
Ki = 0.0001;
Kd = 0.0001;

desiredAngle = pi + pi/6; % desired heading angle; 30 degrees in the world reference
    frame --> pi/6 rad

end
```

B.3 getCoefficientData

```
function [rudderArray,liftArray, dragArray, accuracy] = getCoefficientData()
%getCoefficientData This function outputs the arrays that define the
%acquired data regarding the lift force and the drag force. Moreover, it
%outputs the accuracy of the measurement.
%  Note: the accuracy refers to the uncertainty around each value,
%  therefore half of the uncertainty will be positive with respect to the
%  measured value and half will be negative.
rudderArray = [-45, -40, -30, -20, -10, 0, 10, 20, 30, 40, 45];
```

```
liftArray = [-0.46, -0.48, -0.46, -0.38, -0.2, 0, 0.2, 0.38, 0.46, 0.48, 0.46];
dragArray = [0.22, 0.32, 0.52, 0.78, 0.92, 0.98, 0.92, 0.78, 0.52, 0.32, 0.22];
accuracy = 0.001;
end
```

B.4 calculateAbsoluteAndRelativeError

```
function [abs_err,rel_err_perc] = calculateAbsoluteAndRelativeError(solutionPID,
    desiredAngle)
%calculateAbsoluteAndRelativeError This function calculates the absolute
%and relative error with respect to the desired value of a variable.

%  This function calculates the absolute and relative error with respect
%  to the desired value of a variable. For instance, it can be used to
%  calculate the absolute and relative errors for the heading of a ship
%  after having inserted a PID control.

abs_err = abs(wrapToPi(wrapTo2Pi(solutionPID) - wrapTo2Pi(desiredAngle)));
rel_err_perc = 100*abs_err/desiredAngle;

end
```

B.5 computeResults

B.5.1 Non optimized version

```
function [time, solution, timeL, solutionL, timeH, solutionH, solutionPID,
    solutionPIDhigh, solutionPIDlow] = computeResults(rudderArray, liftArray,
    dragArray, accuracy, invM, D, omega, control_function, timespan, dt, X0, Kp,Ki,
    Kd, desiredAngle)
%computeResults The business logic of the simulator
%  This function corresponds to the core of the simulator and implements
%  its business logic. It takes the acquired data as an input and outputs
%  the data that can be used to visualize the results.

[fL, fD] = findCoefficient(rudderArray, liftArray, dragArray, 0);
[fLl, fDl] = findCoefficient(rudderArray, liftArray, dragArray, accuracy/2);
[fLh, fDh] = findCoefficient(rudderArray, liftArray, dragArray, -accuracy/2);

[time, solution] = ode45(@(t, X) systemODE(t, X, invM, D, omega, control_function, 1,
     true, 1), timespan, X0);
solution = solution';

[timeL, solutionL] = ode45(@(t, X) systemODE(t, X, invM, D, omega, control_function,
    1, true, 0), timespan, X0);
```

```
solutionL = solutionL';

[timeH, solutionH] = ode45(@(t, X) systemODE(t, X, invM, D, omega, control_function,
    1, true, 2), timespan, X0);
solutionH = solutionH';

visualizeInterpolation(rudderArray, fLl, fLh, fL, 2, 'f_L');
visualizeInterpolation(rudderArray, fDl, fDh, fD, 3, 'f_D');

solutionPID = RungeKuttaPID(Kp,Ki, Kd, desiredAngle, timespan, X0, invM, D, omega,
    control_function, fL, fD, dt);
solutionPIDhigh = RungeKuttaPID(Kp,Ki, Kd, desiredAngle, timespan, X0, invM, D, omega
    , control_function, fLh, fDh, dt);
solutionPIDlow = RungeKuttaPID(Kp,Ki, Kd, desiredAngle, timespan, X0, invM, D, omega,
     control_function, fLl, fDl, dt);

end
```

B.5.2 Optimized version

```
function [fL, fD, fLl, fDl, fLh, fDh, time, solution, timeL, solutionL, timeH,
    solutionH, solutionPID, solutionPIDhigh, solutionPIDlow] = computeResults_OPT(
    rudderArray, liftArray, dragArray, accuracy, invM, D, omega, control_function,
    timespan, dt, X0, Kp,Ki, Kd, desiredAngle)
%computeResults_OPT The business logic of the simulator (optimized version)
%  This function corresponds to the core of the simulator and implements
%  its business logic. It takes the acquired data as an input and outputs
%  the data that can be used to visualize the results.

[fL, fD] = findCoefficient(rudderArray, liftArray, dragArray, 0);
[fLl, fDl] = findCoefficient(rudderArray, liftArray, dragArray, accuracy/2);
[fLh, fDh] = findCoefficient(rudderArray, liftArray, dragArray, -accuracy/2);

[time, solution] = ode45(@(t, X) systemODE_OPT(t, X, invM, D, omega, control_function
    , 1, true, fL, fD), timespan, X0);
solution = solution';

[timeL, solutionL] = ode45(@(t, X) systemODE_OPT(t, X, invM, D, omega,
    control_function, 1, true, fLl, fDl), timespan, X0);
solutionL = solutionL';

[timeH, solutionH] = ode45(@(t, X) systemODE_OPT(t, X, invM, D, omega,
    control_function, 1, true, fLh, fDh), timespan, X0);
solutionH = solutionH';

solutionPID = RungeKuttaPID(Kp,Ki, Kd, desiredAngle, timespan, X0, invM, D, omega,
    control_function, fL, fD, dt);
solutionPIDhigh = RungeKuttaPID(Kp,Ki, Kd, desiredAngle, timespan, X0, invM, D, omega
    , control_function, fLh, fDh, dt);
```

```
22 solutionPIDlow = RungeKuttaPID(Kp,Ki, Kd, desiredAngle, timespan, X0, invM, D, omega,
       control_function, fLl, fDl, dt);
23 end
```

B.6 plotForceAgainstShaftSpeed

```
1  function plotForceAgainstShaftSpeed(minShaftSpeed,maxShaftSpeed,shaftStep, kp, kn,
       fig_num)
2  %plotForceAgainstShaftSpeed This function calculates and plots the force T0
3  %with respect to the value of omega
4  %  This function calculates and plots the force T0
5  %with respect to the value of omega for all the possible values of omega.
6  %In particular, the initial omega (minShaftSpeed) and the final omega
7  %(maxShaftSpeed) have to be specified. Moreover, shaftStep considers the
8  %distance between a function evaluation and the next one. kp and kn are
9  %defined according to their meaning in calculateT0.
10
11 omega = minShaftSpeed:shaftStep:maxShaftSpeed;
12 T0 = zeros(1, length(omega));
13
14 for i = 1:length(omega)
15    T0(i) = calculateT0(kp, kn, rpmToHz(omega(i)), minShaftSpeed, maxShaftSpeed);
16 end
17
18 figure(fig_num);
19 plot(omega,T0)
20 title('Force with respect to shaft speed')
21 xlabel('Shaft speed \omega [Hz]')
22 ylabel('Force T_0 [N]')
23 grid on
24
25 end
```

B.7 calculateT0

```
1  function T0 = calculateT0(kp,kn,omega, minShaftSpeed, maxShaftSpeed)
2  %  calculateT0 - This function computes the force T0
3  %  T0 force is the force generated by the thrusters (propellers) when they
4  %  rotate. It can be either positive (omega >= 0) or negative (omega < 0).
5  %  The shaft speed is expressed in RPM (1 RPM = 1/60 Hz). kp and kn are
6  %  coefficients. (Suggested value for kp = 1.47 * 10^5 Ns^2, suggested value
7  %  for kn = 1.65 * 10^5 Ns^2, suggested values for minShaftSpeed and
8  %  maxShaftSpeed: -132 and 132). omega should be expressed in Hz.
9
10 if or(omega < rpmToHz(minShaftSpeed), omega > rpmToHz(maxShaftSpeed))
```

```
11      error("Please define a valid omega.");
12  end
13
14  if (omega >= 0)
15      T0 = kp*omega^2;
16  elseif (omega < 0)
17      T0 = kn*abs(omega)*omega;
18  else
19      error("Error in calculating T0");
20  end
21  end
```

B.8 rpmToHz

```
1   function Hz = rpmToHz(rpm)
2   %rpmToHz Converts an input in rpm in Hertz
3   %   The conversion considers that 1 Hz = 1/60 rpm
4   Hz = rpm/60;
5   end
```

B.9 systemODE

B.9.1 Non optimized version

```
1   function dXdt = systemODE(t, X, invM, D, omega, control_signal, pid, bypassPID,
        resultAccuracy)
2   %systemODE Function to compute the result of the system that describes the
3   %dynamic of the system.
4   %   This function may be used in order to calculate the result of the
5   %   matrix calculations describing the dynamics of the system. In
6   %   particular, this may be used both in ode45, when no PID has been set (bypassPID =
        true) and in
7   %   Runge-Kutta, when a PID has been inserted (bypassPID = false).
8   %   t is the current time (from the beginning of the simulation)
9   %   X is the current status (at time t)
10  %   invM is the inverse of matrix M
11  %   D is the damping matrix
12  %   omega, expressed in Hz, is the shaft angular speed
13  %   control_signal is the input given to the system (the rudder angle)
14  %   pid is the input to the ship system and it takes into account the
15  %   control signal, the backpropagation and the desired angle.
16
17  phi = X(3);
18
19  if (bypassPID==false)
```

```
20      delta = pid; % the input of the system is the output of the PID
21  else
22      delta = control_signal(t); % there is no PID, so there is only the control signal
23  end
24
25  R = calculateR(phi);
26
27  [Fm, Fl, Fh] = get_force_torque(omega, delta);
28
29  switch(resultAccuracy)
30      case 0
31          F = Fl';
32      case 1
33          F = Fm';
34      case 2
35          F = Fh';
36  end
37
38
39  [A, B] = systemMatrix(R, invM, D, F);
40
41  [rA, cA] = size(A);
42  [rB, cB] = size(B);
43
44  if rA~=rB
45      error("A and B should have the same number of rows");
46  end
47
48  dXdt = zeros(rA, 1); % fill dXdt with 0s
49
50  % compute dX/dt = AX + B
51  for i=1:rA
52      for j=1:cA
53          dXdt(i) = dXdt(i) + A(i, j)*X(j);
54      end
55      for k=1:cB
56          dXdt(i) = dXdt(i) + B(i, k);
57      end
58  end
```

B.9.2 Optimized version

```
1   function dXdt = systemODE_OPT(t, X, invM, D, omega, control_signal, pid, bypassPID,
        fL, fD)
2   %systemODE_OPT Function to compute the result of the system that describes the
3   %dynamic of the system.
4   %   This function may be used in order to calculate the result of the
5   %   matrix calculations describing the dynamics of the system. In
6   %   particular, this may be used both in ode45, when no PID has been set (bypassPID =
```

```
    true) and in
%   Runge-Kutta, when a PID has been inserted (bypassPID = false).
%   t is the current time (from the beginning of the simulation)
%   X is the current status (at time t)
%   invM is the inverse of matrix M
%   D is the damping matrix
%   omega, expressed in Hz, is the shaft angular speed
%   control_signal is the input given to the system (the rudder angle)
%   pid is the input to the ship system and it takes into account the
%   contrl signal, the backpropagation and the desired angle.

phi = wrapTo2Pi(X(3));

if (bypassPID==false)
    delta = pid; % the input of the system is the output of the PID
else
    delta = control_signal(t); % there is no PID, so there is only the control signal
end

R = calculateR(phi);

F = get_force_torque_OPT(omega, delta, fL, fD);
F = F';

[A, B] = systemMatrix(R, invM, D, F);

[rA, cA] = size(A);
[rB, cB] = size(B);

if rA~=rB
    error("A and B should have the same number of rows");
end

dXdt = zeros(rA, 1); % fill dXdt with 0s

% compute dX/dt = AX + B
for i=1:rA
    for j=1:cA
        dXdt(i) = dXdt(i) + A(i, j)*X(j);
    end
    for k=1:cB
        dXdt(i) = dXdt(i) + B(i, k);
    end
end
```

B.10 calculateR

```
function R = calculateR(phi)
%calculateR calculate the rotation matrix on the basis of the heading
```

```
%  The rotation matrix is defined as R = [cos(phi) -sin(phi) 0; sin(phi)
%  cos(phi) 0; 0 0 1], where phi is the heading.
R = [cos(phi) -sin(phi) 0; sin(phi) cos(phi) 0; 0 0 1];

end
```

B.11 get_force_torque

B.11.1 Non optimized version

```
function [F, Fl, Fh] = get_force_torque(omega,delta)
%get_force_torque This function gives the force and the torque as a result.
%  The first variable being returned corresponds to the force and the
%  torque in case the values of the drag and lift array are considered
%  without any uncertainty. Fl and Fh refer, respectively, to the
%  situations in which the uncertainty is taken into account with the
%  negative sign and with the positive sign.

[rudderArray, liftArray, dragArray, accuracy] = getCoefficientData(); % gets the
    arrays from the input data

% Calculates the coefficients fL and fD for the different uncertainties
[fL, fD] = findCoefficient(rudderArray, liftArray, dragArray, 0);
[fLl, fDl] = findCoefficient(rudderArray, liftArray, dragArray, -accuracy/2);
[fLh, fDh] = findCoefficient(rudderArray, liftArray, dragArray, accuracy/2);

% Calls the optimized version to calculate the forces and the torques
F = get_force_torque_OPT(omega,delta, fL, fD);
Fl = get_force_torque_OPT(omega,delta, fLl, fDl);
Fh = get_force_torque_OPT(omega,delta, fLh, fDh);

end
```

B.11.2 Optimized version

```
function F = get_force_torque_OPT(omega,delta, fL, fD)
%get_force_torque_OPT This (optimized) function gives the force and the torque as a
    result.
%  Differently from the non-optimized version, forces and torques are
%  calculated on the basis of the given fL and fD. There is no need to
%  output the low, average and high forces and torques, that correspond to
%  different uncertainties, since it is possible to call this function
%  using different fL and fD to obtain them.

[kp, kn] = getKpKn();
```

```
[minShaftSpeed, maxShaftSpeed] = getShaftSpeedLimits();
distance = getDistanceCenterOfMass();

T0 = calculateT0(kp, kn, omega, minShaftSpeed, maxShaftSpeed);

if(omega<0) % The shaft has a negative rotation, therefore it is not possible to
    calculate the drag and the lift because of how the problam has been specified
    Fx = [T0 0 0];
    Fy = [0 0 0];
else

    Fx = [2*T0*polyval(fD, delta) 0 0]; % with respect to the world
    Fy = [0 -2*T0*polyval(fL, delta) 0]; % with respect to the world

    FxB = Fx; % with respect to the body
    FyB = -Fy; % with respect to the body
end

TauzB = cross([distance 0 0], FyB+FxB); % the x distance is caused by the reference
    system change; the y distance has not been considered since the propellers are
    symmetrical
F = FxB + FyB + TauzB; % with respect to the body

% Note: the distance refers to the distance from the center of mass to the
% x coordinate of the point on which the lift force is applied.

end
```

B.12 findCoefficient

```
function [fL, fD] = findCoefficient(rudderArray,liftArray, dragArray, err)
%findCoefficient This function is used in order to find a coefficient (fL
%or fD).
%  The coefficient is calculated on the basis of a given dataset,
%  consisting in a rudder angle array rudderArray and in a force array
%  forceArray. liftArray and dragArray contains the values of the considered
%  forces divided by T0, that is the force in the x direction.

% fL = L/T0
% fD = 1-D/T0



if not(and(length(liftArray) == length(dragArray), length(dragArray) == length(
    rudderArray)))
    error("Please use arrays with the same size");
end

for j=1:length(rudderArray)
```

```
19      dragArray(j) = dragArray(j)+err;
20      liftArray(j) = liftArray(j)+err;
21  end
22
23  fL = polyfit(rudderArray, liftArray, 3); % interpolation with a grade 3 polynomial
        function
24  fD = polyfit(rudderArray, dragArray, 3); % interpolation with a grade 3 polynomial
        function
25
26  end
```

B.13 systemMatrix

```
1   function [A,B] = systemMatrix(R,invM,D,F)
2   %systemMatrix This function calculate the ship system matrices A and B
3   %   A is defined as [zeros(3) R; zeros(3) invM*D*(-1)] and B is defined as
4   %   the following column vector: [0 0 0 invM*F]'
5
6   A = [zeros(3) R; zeros(3) invM*D*(-1)];
7   referenceVariation = [0 0 0];
8   B = vertcat(referenceVariation', invM*F);
9
10  end
```

B.14 RungeKuttaPID

```
1   function solutionPID = RungeKuttaPID(Kp,Ki, Kd, desiredAngle, timespan, X0, invM, D,
        omega, control_function, fL, fD, dt)
2   %RungeKuttaPID This function returns the solutions of the feedback system
3   %in case the PID controller is inserted.
4   %   The simulation has been performed by using the 4th order Runge-Kutta
5   %   method.
6
7   solutionPID = zeros(6, length(timespan)); % the initial solution matrix (dimension
        timespan x 6) is filled with 0s.
8   solutionPID(:,1) = X0; % the initial state is X0 ([0 0 0 0 0 0]')
9   PID = zeros(size(timespan)); % the PID effect is also inizialized to zero for the
        first iteration (there is no backpropagation yet)
10
11  e_previous = 0;
12  e_int = 0;
13
14  %%% 4th order Runge-Kutta method
15  for i=1:length(timespan)-1
16
```

```
17      e = solutionPID(3,i)-desiredAngle; % error (desiredAngle - OUTPUT)
18
19      e_int = e_int + e; % discrete integration
20
21      PID(i) = Kp*e + Ki*e_int*dt + Kd*(e - e_previous)/dt; % PID function (discrete)
22      e_previous = e;
23
24      % Setting of RK parameters
25      K1 = systemODE_OPT(timespan(i), solutionPID(:,i), invM, D, omega, control_function
            , PID(i), false, fL, fD);
26      K2 = systemODE_OPT(timespan(i)+ 0.5*dt, solutionPID(:,i)+0.5*dt*K1, invM, D, omega
            , control_function, PID(i), false, fL, fD);
27      K3 = systemODE_OPT(timespan(i)+ 0.5*dt, solutionPID(:,i)+0.5*dt*K2, invM, D, omega
            , control_function, PID(i), false, fL, fD);
28      K4 = systemODE_OPT(timespan(i)+ dt, solutionPID(:,i)+ dt*K3, invM, D, omega,
            control_function, PID(i), false, fL, fD);
29
30      solutionPID(:,i+1) = solutionPID(:,i) + dt*(K1/6 + K2/3 + K3/3 + K4/6); % Discrete
             step
31  end
32
33  end
```

B.15 showErrorsMessage

```
1   function [] = showErrorsMessage(absolute,relative, measureType)
2   %showErrorsMessage This function allows to visualize the absolute and
3   %relative errors by using a message box.
4   %  This function is part of the visualization (or presentation) part.
5
6   message = strcat('The absolute error for the heading is ', {' '}, num2str(absolute),
        {' '}, 'rad, while the relative error is ', {' '}, num2str(relative),{' '}, '%');
7   msgbox(message, strcat('Absolute and relative error ', {' '}, '(', measureType, ')'))
        ;
8
9   end
```

B.16 getDistanceCenterOfMass

```
1   function distance = getDistanceCenterOfMass()
2   %getDistanceCenterOfMass This function only provides a distance from the
3   %ship center of mass to the point of application of the lift force (the
4   %distance is considered for the x coordinate).
5   %  This is a geometric parameter of the ship.
6   distance = -41.5;
```

```
end
```

B.17 getKpKn

```
function [kp,kn] = getKpKn()
%getKpKn This function outputs the coefficients kp and kn related to the
%propeller nominal thrust.
%  Those values are related to the propeller model and they can change if
%  this model changes.

kp = 1.47*10^5;
kn = 1.65*10^5;

end
```

B.18 getShaftSpeedLimits

```
function [minShaft,maxShaft] = getShaftSpeedLimits()
%getShaftSpeedLimits This function outputs the maximum speed of the
%shaft, expressed in RPM.
%  The provided values depend on the given model and they can change in
%  case the shaft changes.
minShaft = -132;
maxShaft = 132;
end
```

B.19 visualizeInterpolation

```
function [] = visualizeInterpolation(rudderArray, low, high, medium, fig_num, name)
%visualizeInterpolation This function allows to show the result of the
%interpolation of the coefficients fD and fL.
%  This function is part of the visualization (or presentation) part.

% Interpolation
figure(fig_num);
hold on
title(strcat('Interpolation of ',{' '}, name))
xlabel('Rudder angle')
ylabel(name)
grid on
```

```

hold on
plot(rudderArray, polyval(medium, rudderArray));

grid on


hold on
plot(rudderArray, polyval(high, rudderArray));

hold on
plot(rudderArray, polyval(low, rudderArray));

legend( 'Average', 'Low', 'High')

end
```

B.20 visualizeResults

B.20.1 Non optimized version

```
function [] = visualizeResults(solutionL, solutionH, solution, solutionPIDlow,
    solutionPIDhigh, solutionPID, desiredAngle, timeL, timeH, time)
%visualizeResults This function allows to show the result of the
%simulation (headings and trajectories) with and without the uncertainties
%and with and without the PID controller.
%  This function is part of the visualization (or presentation) part.

[minShaftSpeed, maxShaftSpeed] = getShaftSpeedLimits();
[kp, kn] = getKpKn();

% Plot TO
plotForceAgainstShaftSpeed(minShaftSpeed, maxShaftSpeed, 0.1, kp, kn, 1);

%Heading
figure(4)

title('Heading with respect to time')
xlabel('Time [s]')
ylabel('Heading [rad]')


hold on
plot(time, (solution(3,:)));


hold on
plot(time, (solutionH(3,:)));

hold on
```

```
29 plot(time, (solutionL(3,:)));
30
31
32 grid on
33 hold on
34 plot(time, solutionPID(3,:));
35
36 hold on
37 plot(timeL, solutionPIDlow(3,:));
38
39 hold on
40 plot(timeH, solutionPIDhigh(3,:));
41
42 hold on
43 fplot(desiredAngle, 'k.')
44 fplot(desiredAngle-2*pi, 'k.')
45
46
47 legend('Without PID (average)', 'Without PID (low)', 'Without PID (high)', 'With PID
       (average)','With PID (low)','With PID (high)','Objective heading')
48
49 hold off
50
51
52 % Trajectory
53 figure(5);
54 axis equal tight
55 hold on
56 plot(solution(2,:), solution(1,:));
57 title('Ship trajectory')
58 xlabel('y_W [m]')
59 ylabel('x_W [m]')
60 grid on
61
62 hold on
63 plot(solutionL(2,:), solutionL(1,:));
64
65 grid on
66
67 hold on
68 plot(solutionH(2,:), solutionH(1,:));
69
70 hold on
71 plot(solutionPID(2,:), solutionPID(1,:));
72
73 hold on
74 plot(solutionPIDlow(2,:), solutionPIDlow(1,:));
75
76 hold on
77 plot(solutionPIDhigh(2,:), solutionPIDhigh(1,:));
78
79 hold on
80 fplot(@(t) tan(pi/2 - desiredAngle)*t, 'k.')
```

```
legend( 'Without PID (average)', 'Without PID (low)', 'Without PID (high)', 'With PID (average)', 'With PID (low)', 'With PID (high)', 'Objective trajectory')

hold off

[abs_err, rel_err_perc] = calculateAbsoluteAndRelativeError(solutionPID(3,end), desiredAngle);
showErrorsMessage(abs_err, rel_err_perc, "average");

[abs_err_low, rel_err_perc_low] = calculateAbsoluteAndRelativeError(solutionPIDlow(3, end), desiredAngle);
showErrorsMessage(abs_err_low, rel_err_perc_low, "low");

[abs_err_high, rel_err_perc_high] = calculateAbsoluteAndRelativeError(solutionPIDhigh(3, end), desiredAngle);
showErrorsMessage(abs_err_high, rel_err_perc_high, "high");

end
```

B.20.2 Optimized version

```
function [] = visualizeResults_OPT(rudderArray, fLl, fLh, fL, fDl, fDh, fD, solutionL, solutionH, solution, solutionPIDlow, solutionPIDhigh, solutionPID, desiredAngle, timeL, timeH, time)
%visualizeResults_OPT This (optimized) function allows to show the result of the
%simulation (headings and trajectories) with and without the uncertainties
%and with and without the PID controller.
%  This function is part of the visualization (or presentation) part.

[minShaftSpeed, maxShaftSpeed] = getShaftSpeedLimits();
[kp, kn] = getKpKn();

% Plot T0
plotForceAgainstShaftSpeed(minShaftSpeed, maxShaftSpeed, 0.1, kp, kn, 1);

visualizeInterpolation(rudderArray, fLl, fLh, fL, 2, 'f_L');
visualizeInterpolation(rudderArray, fDl, fDh, fD, 3, 'f_D');

%Heading
figure(4)

title('Heading with respect to time')
xlabel('Time [s]')
ylabel('Heading [rad]')


hold on
plot(time, (solution(3,:)));

```

```
27
28  hold on
29  plot(time, (solutionH(3,:)));
30
31  hold on
32  plot(time, (solutionL(3,:)));
33
34
35  grid on
36  hold on
37  plot(time, (solutionPID(3,:)));
38
39  hold on
40  plot(timeL, (solutionPIDlow(3,:)));
41
42  hold on
43  plot(timeH, (solutionPIDhigh(3,:)));
44
45  hold on
46  fplot(desiredAngle, 'k.')
47  fplot(desiredAngle -2*pi, 'k.')
48
49  legend('Without PID (average)', 'Without PID (low)', 'Without PID (high)', 'With PID
        (average)','With PID (low)','With PID (high)','Objective heading')
50
51  hold off
52
53
54  % Trajectory
55  figure(5);
56  axis equal tight
57  hold on
58  plot(solution(2,:), solution(1,:));
59  title('Ship trajectory')
60  xlabel('y_W [m]')
61  ylabel('x_W [m]')
62  grid on
63
64  hold on
65  plot(solutionL(2,:), solutionL(1,:));
66
67  grid on
68
69  hold on
70  plot(solutionH(2,:), solutionH(1,:));
71
72  hold on
73  plot(solutionPID(2,:), solutionPID(1,:));
74
75  hold on
76  plot(solutionPIDlow(2,:), solutionPIDlow(1,:));
77
78  hold on
```

```
79  plot(solutionPIDhigh(2,:), solutionPIDhigh(1,:));
80
81  hold on
82  fplot(@(t) tan(pi/2 - desiredAngle)*t, 'k.')
83  legend( 'Without PID (average)', 'Without PID (low)', 'Without PID (high)', 'With PID
        (average)', 'With PID (low)', 'With PID (high)', 'Objective trajectory')
84
85  hold off
86
87  [abs_err, rel_err_perc] = calculateAbsoluteAndRelativeError(solutionPID(3,end),
        desiredAngle);
88  showErrorsMessage(abs_err, rel_err_perc, "average");
89
90  [abs_err_low, rel_err_perc_low] = calculateAbsoluteAndRelativeError(solutionPIDlow(3,
        end), desiredAngle);
91  showErrorsMessage(abs_err_low, rel_err_perc_low, "low");
92
93  [abs_err_high, rel_err_perc_high] = calculateAbsoluteAndRelativeError(solutionPIDhigh
        (3, end), desiredAngle);
94  showErrorsMessage(abs_err_high, rel_err_perc_high, "high");
95
96  end
```