Integrating a high fidelity ship maneuvering simulator with ROS: a path-following case study

Bruno Scaglione, Pedro Marzagão
Mechatronics Engineering dept. (PMR-POLI)
Polytechnic School of the University of São Paulo (POLI-USP)
São Paulo, Brazil
bruno.c.scaglione@usp.br
phmarzagao@usp.br

***Abstract*—Ship maneuvering is a difficult task, especially in restricted areas such as port access canals. The design of these access passages is done by analyzing the empirical distribution of ship trajectories. The samples are taken both from computer run simulations, known as *fast-time*, and human pilot run simulations, known as *real-time*. This article tackles the problems with the current situation of the *fast-time* simulations. The current software that controls the ship is relatively legacy and does not take advantage of the vast capabilities of the state-of-the-art ship maneuvering simulator, named *pydyna*, and developed inside the Numerical Offshore Tank (TPN) of the University of São Paulo (USP). The *Robot Operating System 2* (*ROS2*) is presented as a solution to host these applications. The simulator is integrated within *ROS2* in the form of a *ROS2* package, and a proof-of-concept study case is developed to approximate real-world *fast-time* simulations. The study case is Guidance-Navigation-Control (GNC) architecture that makes a ship follow a desired path defined by *waypoints* containing desired 2D localization coordinates and desired surge velocity. The results show that the ship is able to follow slight zigzag path, at the environmental condition in which it was tuned and with a favorable initial yaw angle. The velocity control works approximately well in this case. A *ROS2* package for the maneuvering simulator and another one for the *path-following* architecture were developed. TPN has a starting point to kickoff *ROS2* efforts regarding maritime projects with large impact on society.**

***Index Terms*—Ship, Simulator, ROS2, GNC, Control**

I. Introduction

Operating a ship is not an easy task, especially in constrained areas with obstacles, such as canals leading to ports. Consequently, incidents occur, and their impact can be massive socially and economically.

According to the European Maritime Safety Agency, in 2016 there were 3145 casualties and incidents and 26 ship lost due to maritime accidents of over 3500 ships. By far the biggest cause of the reported accidents were human erroneous actions, making up 60.5 percent of them. Not only that, the majority of ships involved in these situations are general cargo ships, making up 43 percent of the accidents. Of these, 42 percent of the casualties occurred in port areas, and 28 percent in coastal areas [1].

In addition, recently, in March 2021, the whole world witnessed a single ship stranded in the Suez Canal blocking 12 percent of the world's trade and costing nearly 10 billion dollars a day [2]. An animation of the accident can be seen in [3, Fig. I].

Fig. 1. Animation of Ever Given stuck in Suez Canal [3]

Thus, efforts towards minimizing incidents in these situations are relevant. Considering human error is an important factor, moving towards solutions which depend less on humans could be beneficial.

One path is the development of autonomous ships. This term can be used for ships spanning different levels of autonomy, from simple auto-pilots to complete autonomy. However, the current development and use of this technology is closer to the former than the latter. Nonetheless, Guidance-Navigation-Control (GNC) Systems are performing increasingly harder tasks and already offer tremendous benefits to the operation of ships at the present moment.

Considering the use of completely autonomous ships is a far cry from the current paradigm, solutions taking into account humans are used. Sometimes the line between a situation that is considered an error by the pilot and accounting for human navigation variability (and error-prone nature), can be difficult to draw. With this is mind, much of the safety engineering can drift towards the design of canals.

With the advancement of simulation techniques, design of ports and canals by hand has become outdated. Not only the calculations take time, but different ship dynamics, en-

vironmental influences and pilot navigation variability must be taken into account. The simulations are able to recreate the studied area, with different models of ships. Regions in the canal can be separated through the variability of trajectories they present after a set of simulation runs. High variability regions demand a greater width. It is essential to guarantee a secure gap between the border of the region and the canal coast.

Another pertinent point that must be taken into consideration is how costly building a canal is. For example, in nowadays currency, the Panama Canal would have cost approximately 14.3 billion dollars and its length is of around 82 kilometers, with a depth of 12m and width of 33.5 meters [4]. Just a meter less or more in construction would have had an absolutely great impact on the total cost. This means that saving money is a huge concern also, and needs to be handled together with safety efforts.

Therefore, by using simulators it is possible to analyze, under a myriad of conditions, ship trajectories. Based on the results found, the ones building or adapting the port canal are then able determine the required shape of the course that guarantees that specific types of ships can securely pass trough, while only spending as much as needed in the process.

Optimal port design is one of the main areas of work, having significant expertise, inside Numerical Offshore Tank (TPN), known as "Tanque de Provas Numérico", Laboratory of the University of São Paulo (USP) [5]. TPN is a "[...] computational and experimental hydrodynamic laboratory for the design and analysis of offshore systems, ports, platforms, ships and barges" [6].

TPN has a state-of-the-art ship maneuvering simulator developed on-premise, by the Ship Maneuvering Simulation Center, and using proprietary knowledge and technology. There are two types of maneuvering simulations conducted: *real-time* and *fast-time*. *Real-time* simulations are the ones that aim to mimic real-world operation. The pilot controls the simulator, as if it were a real ship, to perform the trajectory (e.g. navigating through a port access canal). The *Full Mission Simulator 1*, one of TPN's real-time simulators, can be seen in [7, Fig. I]. Differently, *fast-time* simulations are done by computer controlled maneuvers, named path-following autopilot, that follows desired *waypoints*. These *waypoints* contain desired locations and may present desired surge velocities as well. An example of a *fast-time* simulation, of a ship entering a port, can be seen in [8, Fig. I].

Although *real-time* simulations are more realistic in terms of real-world operations, *fast-time* simulations can have much more runs in the same period of time, demand less resources and allows standardization of practices.

Currently TPN makes use of both types of simulations for analysis and design of port access canals. With a small amount of *real-time* (approximately 30) runs and a vast amount *fast-time* (aproximately 500) runs.

The *fast-time* software in TPN is composed of two parts: the core state-of-the-art maneuvering simulator implemented in *C++* with a *Python* Interface, distributed as a private *Python* library and named *pydyna*; and the surrounding path-following architecture implemented in *MatLab*.

Fig. 2. *Real-time* simulator: *Full Mission Simulator 1* [7]

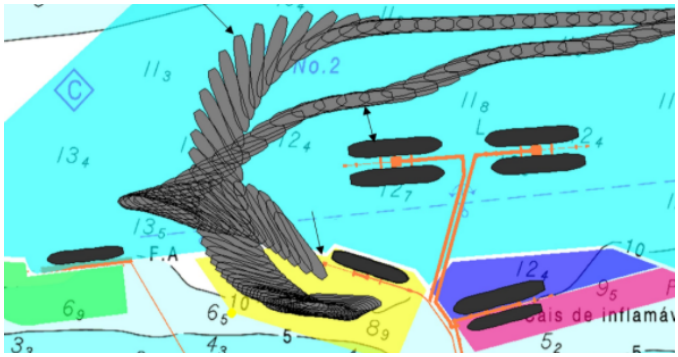

Fig. 3. Example of a *fast-time* simulation [8]

However, this structure does not take advantage of the vast capabilities provided by the state-of-the-art maneuvering simulator. The control software presents some limitations, such as: lack of native integration with the maneuvering simulator; lack of support for modularity; does not enable easy aggregation of state-of-the-art robotics packages; does not enable rapid prototyping and experimenting; requires experts to develop and maintain; does not have hardware in the loop; cannot scale easily to a complex robotic system.

The Robot Operating System 2 (ROS2) is a low-level framework, that operates on top of an Operating System (OS), and provides solutions to many of these problems; in addition to having a large community behind it. *ROS2* "[...] provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more [...]" [9].

A great advantage also resides in the fact that the framework operates mainly in *Python* and *C++*. The *Python* language provides a wide range of useful open-source libraries and applications to be used right away. The *C++* library also provides a lot of libraries, but aggregates the advantage of code efficiency. With these two, quality software can be aggregated into the *ROS2* environment, without having any relation to *ROS2*.

This project develops the *pydyna_simple* package that en-

capsulates the maneuvering simulator. There it can be combined with multiple other packages natively and in modular fashion.

The package proposition is done in the context of a proof-of-concept case study, where the aim is to provide demonstration of some capabilities that come along with the *ROS2* environment. It is a GNC path-following architecture that makes the ship follow a desired path. This study case is representative of actual *fast-time* simulations. A package that encapsulates the whole path-following architecture, called *path_following*, is also developed.

II. Task Description

The path-following task is the is described by a GNC system that receives *waypoints* from an operator and tries to follow the path described by the lines connecting the *waypoints*. In addition, the ship also tries to reach the desired surge velocities of the *waypoints* at their locations. Achieving the velocities is not part of the task, rather a complementary desired behavior. Therefore, the path is described by a variable number of tuples $(x_{des},\ y_{des},\ u_{des})$ where $(x_{des},\ y_{des})$ is the desired location and u_{des} the desired surge velocity at the location.

The concept of reaching a *waypoint* is generalized to reaching a radius of acceptance R_w around the *waypoint*. In this way the ship can adapt itself earlier to the next path line, and develop a resulting path that is an interpolation between two consecutive path lines. This behavior generates smooth path roughly designed by the initial path lines.

It is important to describe how the evaluation of the path-following task occurs. The following conditions must be satisfied in order to judge the path-following algorithm as successful:

$$ate \ \leq \ R_{wFinal} \tag{1}$$

where R_{wFinal} is the radius of acceptance of the final *waypoint*, which can be different than the radius of acceptance of all the other *waypoints* R_w.

$$cte_m \ < \ \frac{beam}{2} \tag{2}$$

Where cte_m is the mean *cross-track* error and $beam = 32.2\ m$ is the width of the vessel used in this project, at it's widest point. Also illustrated in Fig. The *cross-track*, or cte, is the smallest distance from the ship's center of mass to any path line that composes the path.

The value chosen for R_w provides a trade-off between reaching the *waypoints* precisely and having a smooth path.

The ship has a single thruster, which generates the thrust force for this task. However, the input to the system is actually the thruster rotation, which can be approximately found given the required thrust. Thus, the modeling is done considering thrust force as the input and then converts to rotation.

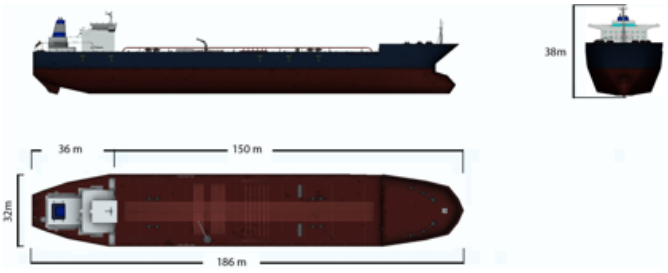

Fig. 4. Different views and parametrization of the *Tanker55000DWT* [10]

III. Architecture

The project's main architecture is composed of five major groups: Dynamical Model, Control, Guidance, Navigation and Visualization; as illustrated in Fig. III. In this environment, the controllers and filters are subject to a final tuning, guided by the error metrics, to get a better path-following behavior.

Fig. III portrays the main interactions[1] between the blocks that compose the system. Where: w_{ps} are the *waypoints*; ψ_{ref} is the reference yaw angle; n is the propeller angular velocity; δ is the rudder angle; X is the state; X_s is the simulated state, that is, considering *hardware-in-the-loop* dynamics of realistic sensors; $\bar{X}_s$ is the filtered simulated state; initial conditions are position η and velocity ν.

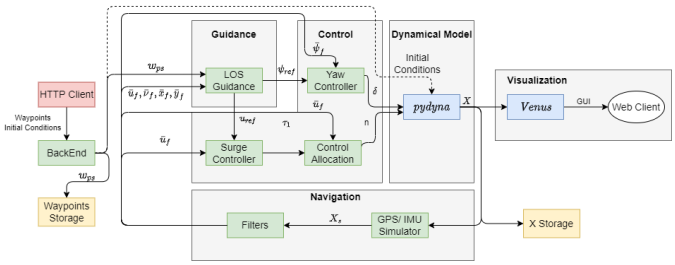

Fig. 5. Architecture of the production environment, of the path-following system. Where: nodes in green are implemented with the use of public libraries; node in blue is implemented with the use of private libraries and public libraries; node in red is a third-party tool; nodes in yellow are files; node with no color is a web browser.

IV. Modeling

A. Maneuvering Simulator Model

The mathematical model behind the simulator developed by [10] in *python*, called *pydyna*, is presented by [11] in three degrees of freedom (3DOF) in [11, eq. (3)]. The 3DOF configuration space is illustrated in [12, Fig. IV-A]

[1]There are other data exchanged between blocks in some cases, but they are not fundamental to the description of the system and would pollute the schematic.

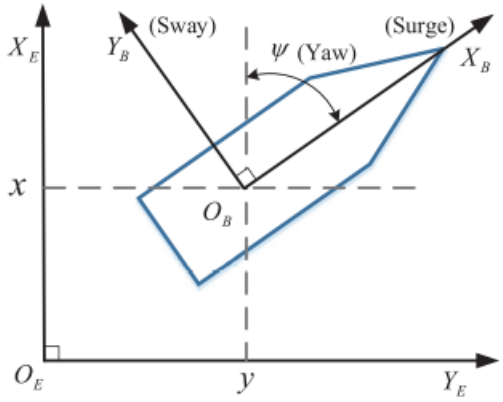

Fig. 6. Schematic diagram of the 3-DOF configuration space of a marine ship [12].

$$\begin{bmatrix} m - X_{\dot{u}} & 0 & 0 \\ 0 & m - Y_{\dot{v}} & mx_G - Y_{\dot{r}} \\ 0 & mx_G - Y_{\dot{r}} & I_z - N_{\dot{r}} \end{bmatrix} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 & -mr & -mx_G r + Y_{\dot{v}} v + Y_{\dot{r}} r \\ mr & 0 & -X_{\dot{u}} u \\ mx_G r + Y_{\dot{v}} v + Y_{\dot{r}} r & X_{\dot{u}} u & 0 \end{bmatrix} \times \begin{bmatrix} u \\ v \\ r \end{bmatrix} + (X_{\dot{u}} - Y_{\dot{v}}) \begin{bmatrix} 0 & r & 0 \\ r & 0 & 0 \\ -v & u_c - u & 0 \end{bmatrix} \begin{bmatrix} u_c \\ v_c \\ 0 \end{bmatrix} = F_{rudder} + F_{prop} + F_{tugs} + F_{curr} + F_{wind} + F_{wave} \tag{3}$$

Respectively, the vectors $\nu = [u \;\; v \;\; r]^T$, $\nu_c = [u_c \;\; v_c \;\; 0]^T$ and $\dot{\nu} = [\dot{u} \;\; \dot{v} \;\; \dot{r}]^T$ are the velocity of the vessel, velocity of the ocean current and acceleration of the vessel; all described in the vessel-fixed reference frame.

As for the right side of the equation: F_{rudder}, F_{prop}, F_{tugs}, F_{curr}, F_{wind}, F_{wave} are, respectively, the forces of the rudder, propeller, tug boats, current, wind and waves. The force F_{tugs} is not considered in this project, since only single ship simulations are performed. The variable m is the mass of the ship. The are variables are coefficients.

However, the controllers are designed using simplified models. The simplifications adopted are listed below:

- restoring (hydrostatic) forces are negligible;
- centripetal and coriolis forces linear around surge velocity u;
- ocean current is negligible;
- port-starboard (left-right sides of the ship) symmetry of the ship;
- sway velocity v and yaw velocity r are small;
- ship operates at surge velocity $|u| \geq 2\ m/s$.

B. Surge model

The surge velocity model used is described by eq. (4).

$$(m - X_{\dot{u}})\dot{u} - X_{|u|u} u|u| = \tau_1 + w_{\tau_1} \tag{4}$$

Where: $m = 40415\ t$ is the ship's mass; $X_{\dot{u}} = -3375\ t$ and $X_{|u|u} = -86\ \frac{kN}{(m/s)}$ are the additional mass and damping coefficients, respectively; u is the surge velocity; τ_1 is the thrust force (cotrol action); and w_{τ_1} is the process noise due to environmental disturbances (waves, wind e current) and unmodelled dynamics.

The parameters m, $X_{\dot{u}}$ and $X_{|u|u}$ are extracted directly from the ship's configuration file.

C. Yaw model

The yaw angle model derived from eq. (5). This is the First Order Nomoto model.

$$\frac{r}{\delta}(s) = \frac{K}{1 + Ts} \tag{5}$$

The model is represented as a transfer function. Where: $\delta\ [rad]$ is the rudder angle; $r\ [rad/s]$ is the yaw velocity; $K = 1.6e-05\ [dimensionless]$ is the gain; $T = 133\ s$ is the time constant.

The parameters K and T are identified by analyzing the step response. Using ψ and transferring to the time domain. The final yaw model is described by eq. (6).

$$T\ddot{\psi} + \dot{\psi} = K\delta \tag{6}$$

D. Actuator Model

The actuator model relates τ_1 to the actual input of the real system: the propeller rotation n_p. Once obtained the necessary force, this model is used to do control allocation. The model is described by eq. (7) below:

$$\tau_1 = \rho\, D_p^4\, K_T\,(J)\, n_p\, |n_p| \tag{7}$$

where: $\rho = 1.0245\ t/m^3$ is the specific mass of the water; $D_p = 7\ m$ is the diameter of the propeller; $K_T\,(J)$ is a coefficient dependent on u where $J = \frac{u}{n_p D}$; and $n_p\ [Hz]$ is the propeller rotation. The maximum rotation of the propeller is $n_{p_{max}} = 1.75\ Hz$, and therefore, the maximum thrust is $\tau_1 = 2445.28\ kN$ (for $J = 0$).

The values of $K_T\,(J)$ are found by a linear fit using three known[2] data points: $(0;\ 0.32)$, $(0.7;\ 0.12)$ and $(1;\ 0)$. The plot of $K_T\,(J)$ is shown in Fig. IV-D and the equation is shown in eq. (8).

$$K_T = 0.3246 - 0.3139 \cdot J \tag{8}$$

Substituting eq. (8) in eq. (7) and solving for n_p:

$$n_p = \begin{cases} c_1\sqrt{(c_2 u^2 + \tau_1)} + c3u & \tau \geq 0 \\ -c_1\sqrt{(c_2 u^2 - \tau_1)} - c_3 u & \tau < 0 \end{cases} \tag{9}$$

where $c_1 = 0.036$, $c_2 = 3.53$, $c_3 = 0.067$, τ_1 is the thrust force, and u is the controlled velocity.

[2]Experimental results found by TPN-USP.

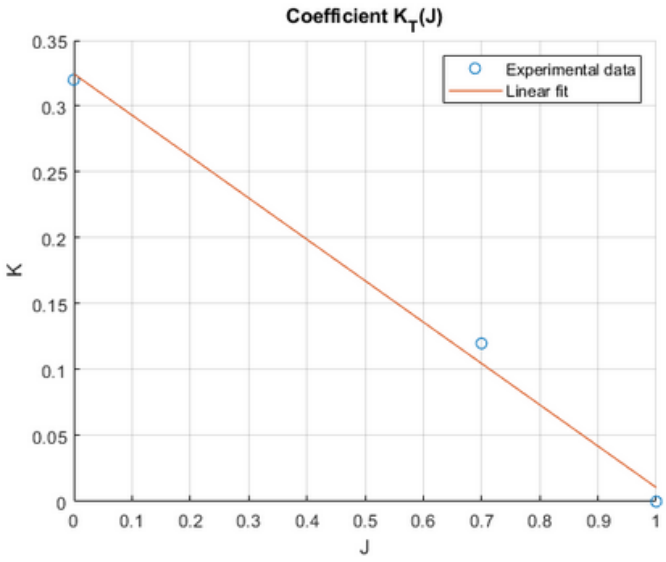

Fig. 7. Coefficient K_T

Thus, eq. (8) is used to convert the force τ_1 to rotation n_p which is the actual input to the simulator.

V. Control

A. Yaw Control

For yaw control, a digital PID controller is used, as shown in eq. (10).

$$\delta(\tilde{\theta}) = -K_p \cdot \tilde{\theta}_k - K_d \cdot \frac{\tilde{\theta}_k - \tilde{\theta}_{k-n}}{t_k - t_{k-n}} - enable(\tilde{\theta}_k) \cdot \left(K_i \cdot \sum_{i=0}^{k-1} \tilde{\theta}_i \cdot \Delta_t \right) \tag{10}$$

Where: θ is the yaw angle measured from east anti-clockwise; $\tilde{\theta}_k$ is the error in instant k; $K_p = 1.6 > 0$ $[dimensionless]$, $K_d = 65 > 0$ $[s]$ and $K_i = 0.00075 > 0$ $[1/s]$ are the PID gains; $\Delta_t = 0.1$ s is the time step; $enable(\tilde{\theta})$ is an *anti-windup* strategy that disables the integral contribution for big errors; and n is a parameter used to change the approximation of the error derivative.

The final parameters K_p $[dimensionless]$, K_d $[s]$ and K_i $[1/s]$ are obtained after one step of design and two steps of tuning: (1) controller is designed for desired damping ratio and bandwidth as described in [13, Alg. 1]; (2) tuned with the surge plant separately; and (3) tuned in the production architecture. The $enable(\tilde{\theta})$ function is presented in eq. (11), and the error $\tilde{\theta}$ is in eq. (12).

Where: θ_d is the desired yaw angle; $C = 0.1$ is the integration threshold in $enable(\tilde{\theta})$ anti-windup strategy.

Algorithm 1 *PID* gains for a yaw controller [13]

procedure YAWCONTROLLERPIDGAINS(w_b, ζ) ▷ $w_b > 0$ and $\zeta > 0$
  $w_n \leftarrow \frac{1}{\sqrt{1 - 2\zeta^2 + \sqrt{4\zeta^4 - \zeta^2 + 2}}} w_b$
  $K_p \leftarrow m w_n^2 - k$ ▷ Proportional Gain
  $K_d \leftarrow 2\zeta w_n m - d$ ▷ Derivative Gain
  $K_i \leftarrow K_p \frac{w_n}{10}$ ▷ Integral Gain
  return (K_p, K_d, K_i)
end procedure

$$enable(\tilde{\theta}) = \begin{cases} 1 & \tilde{\theta} \leq C \\ 0 & \tilde{\theta} > C \end{cases} \tag{11}$$

$$\tilde{\theta} = \begin{cases} \theta - \theta_d & |\theta - \theta_d| \leq \pi \\ -(\pi - (\theta \,\%\, \pi - \theta_d \,\%\, \pi)) & \theta - \theta_d > \pi \\ \pi + (\theta \,\%\, \pi - \theta_d \,\%\, \pi) & \theta - \theta_d < -\pi \end{cases} \tag{12}$$

B. Surge Control

For surge control, *Sliding Mode Control* is used. The notation is a bit different here; x is the surge velocity and $u = \frac{\tau_1}{m - X_{\dot{x}}}$ is the input of the dynamical model of the sliding variable $s = \tilde{x} = x - x_{des}$ with x_{des} being the desired surge velocity.

The controller is defined as shown in eq. (13).

$$u = -\hat{f}_p - k \cdot sat\left(\frac{s(x)}{\phi}\right) \tag{13}$$

Where: $\hat{f}_p = -1.9091 \cdot 10^{-4} \cdot x|x|$ is the point estimate of $f(x) = \frac{X_{|x|x} x|x|}{(m - X_{\dot{x}})} + w_{\tau_1}$; k needs to be big enough to attend the determined acceleration requisites and compensate uncertainties of $f(x)$, to guarantee sliding; ϕ is a parameter that dictates the thickness of the boundary layer; and the $sat(x)$ function[3] is present to avoid shattering of the input u.

The equation for k is shown in eq. (14).

$$k = |\eta| + F \tag{14}$$

Where: η is the imposed acceleration, obtained by $\eta = \frac{x - x_{des}}{t_{est}}$ with t_{est} being the estimated time from one *waypoint* to the next; $F = 0.08$ accounts for the uncertainty in $f(x)$ expressed in w_{τ_1}. The value for F is obtained after two steps of tuning: (1) with the surge plant separately and (2) in the production architecture.

VI. Guidance

Guidance involves providing set-points for the yaw controller and surge controller. Particularly, the set-points for the yaw controller make the ship converge to a desired path. The guidance method used for yaw is the *enclosure-based-LOS (Line-Of-Sight)*. The essence of this method is to trace

[3]The sat function is $sat(x) = max(-1,\ min(1,\ x))$.

a circumference with radius R_{los} around the ship with the path line in order to obtain the desired yaw angle. The main equations can be seen in [13, eq. (15)] and [13, eq. (16)].

$$\psi_d(t) = \chi_d(t) + \beta \tag{15}$$

$$\chi_d(t) = arctan2(y_{los} - y(t), x_{los} - x(t)) \tag{16}$$

Where: $\psi_d(t)$ is the desired yaw angle; β is the angular *offset* between the yaw angle and the ship velocity (caused by the latitudinal velocity imposed by the environment); (x_{los}, y_{los}) is the spot where the ship's circumference intercepts the path line; and $(x(t), y(t))$ is the ship's position in the t instant. All the variables can be seen in [13, Fig. VI].

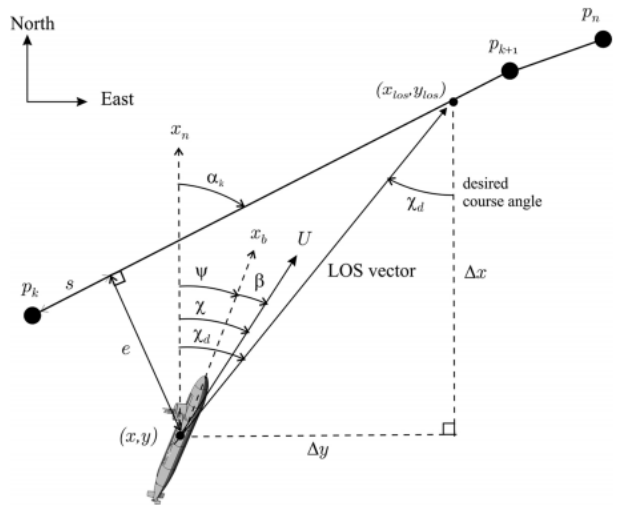

Fig. 8. Diagram representing LOS guidance [13]

The set-points given to the surge controller are simply the desired velocity of the next *waypoint* the ship is trying to reach.

VII. Filters

A. Wave Filter

The wave filter implemented in this work is a digital 6^{th} order *notch* filter, which was projected in the continuous domain and then passed to the discrete domain via a bi-linear transformation.

The filter in the continuous domain is as follows in [13, eq. (17)].

$$h_n(s) = \prod_{i=1}^{3} \frac{s^2 + 2\zeta s + w_i^2}{(s + w_i)^2} \tag{17}$$

Where: $\zeta = 0.7\ [dimensionless]$ is the damping ratio; $w_1 = 0.29124\ rad/s$; $w_2 = 0.52124\ rad/s$; $w_3 = 0.89124\ rad/s$. The central frequency, $w_2\ [rad/s]$, is the wave's frequency of $0.083\ Hz$, extracted directly from the ship's configuration *.p3d* file.

After applying the bi-linear transform the filter is digitized and expressed in eq. (18).

$$h_n(z) = \frac{a_n + b_n \cdot z^{-1} +; c_n \cdot z^{-2} + d_n \cdot z^{-3} + e_n \cdot z^{-4} + f_n \cdot z^{-5} + g_n \cdot z^{-6}}{h_n + i_n \cdot z^{-1} + j_n \cdot z^{-2} + k_n \cdot z^{-3} + l_n \cdot z^{-4} + m_n \cdot z^{-5} + n_n \cdot z^{-6}} \tag{18}$$

where the coefficients are described until the second decimal place and are the following: $a_n = 0.75$; $b_n = -4.68$; $c_n = 12.17$; $d_n = -16.89$; $e_n = 13.18$; $f_n = -5.48$; $g_n = 0.95$; $h_n = 0.71$; $i_n = -4.51$; $j_n = 11.959$; $k_n = -16.87$; $l_n = 13.39$; $m_n = -5.67$; $n_n = 1$.

The Bode plot of the filter can be seen in Fig. VII-A. It's possible to notice that the gain is at it's bottom in a frequency between $0.09\ Hz$ and $0.1\ Hz$, which is pretty close to the $0.083\ Hz$ intended.

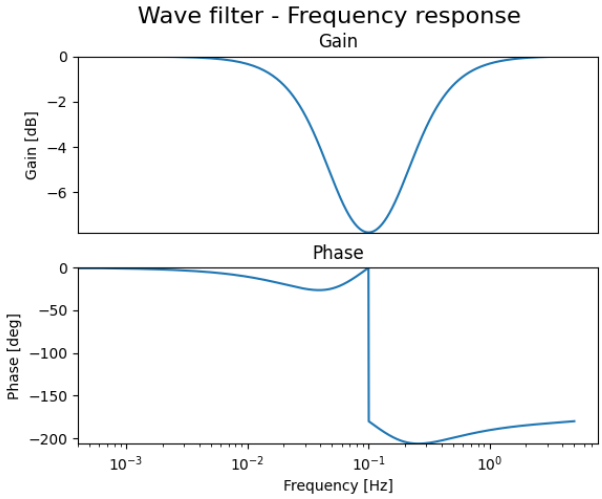

Fig. 9. Bode plot of the wave filter

B. Noise Filter

The noise in the sensors actually comes from an emulation in the architecture. A Gaussian noise is added to the real state, in order to approximate the simulation to a real-world scenario. As a way of removing this noise, a 6^{th} order low-pass digital *Butterworth* filter was used.

The digital filter is expressed by eq. (19).

$$h_{lp}(z) = \frac{1}{h_{lp} + i_{lp} \cdot z^{-1} + j_{lp} \cdot z^{-2} + k_{lp} \cdot z^{-3} + l_{lp} \cdot z^{-4} + m_{lp} \cdot z^{-5} + n_{lp} \cdot z^{-6}} \tag{19}$$

where the coefficients are described until the second decimal place and are the following: $h_{lp} = 0.77$; $i_{lp} = -4.83$; $j_{lp} = 12.61$; $k_{lp} = -17.55$; $l_{lp} = 13.74$; $m_{lp} = -5.74$; $n_{lp} = 1$.

The Bode plot of the filter can be seen in Fig. VII-B. The gain starts to reduce at $0.1\ Hz$.

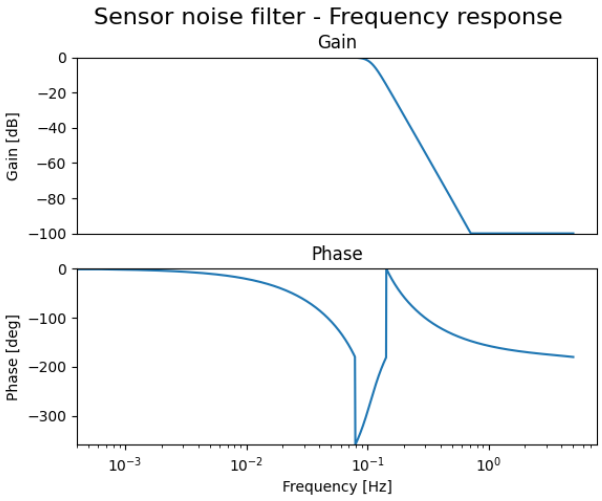

Fig. 10. Bode plot of the noise filter

VIII. Results and Discussion

The results were generated for the ship in several different conditions, called different cases. In this paper, only one specific case is presented and discussed. A link to the video of the ship following the path is given, then the conditions that define the case are described, following the error metrics and the plots obtained.

A. Path-following

The path that the ship has to follow can be seen in Fig. VIII-D. The video of the ship following this path can be found at https://drive.google.com/file/d/1SoxF2F4tg1XCXDPFP8dkcI4\qdgSZ-cwu/view?usp=sharing.

B. Case conditions

The conditions of the case are as follows:

- Zigzag *waypoints*.
- initial state:
  - $(x,\ y,\ \theta) = (0\ m,\ 0\ m,\ 1.2\ rad)$;
  - $(u,\ v,\ r) = (2\ m/s,\ 0\ m/s,\ 0\ rad/s)$;
- sensor noise emulation: on;
- $R_w = R_{los} = 2L_{pp}$;
- $R_{wFinal} = 50\ m$;
- initial coordinates: (-23.06255, -44.2772)[4];
- environmental conditions: *main environmental conditions*;
- reached final *waypoint*: yes.

C. Main environmental conditions

The main environmental conditions are the ones in which the modules of the architecture were tuned to operate, can be seen in Fig. VIII-C and are described below.

[4]Localization: "Angra dos Reis, Rio de Janeiro, Brazil".

- waves: $T_w = 12\ s$, $h = 2\ m$ and acting at 225 deg (starting east and anticlockwise). Where T_w is the wave's period and h is it's height;
- wind: $v_w = 8.74\ m/s$ and acting at 210 deg (starting east and anticlockwise). Where v_w is the wind's speed;
- current: $v_c = 1\ m/s$ and acting at 330 deg (starting east and anticlockwise). Where v_c is the current's speed.

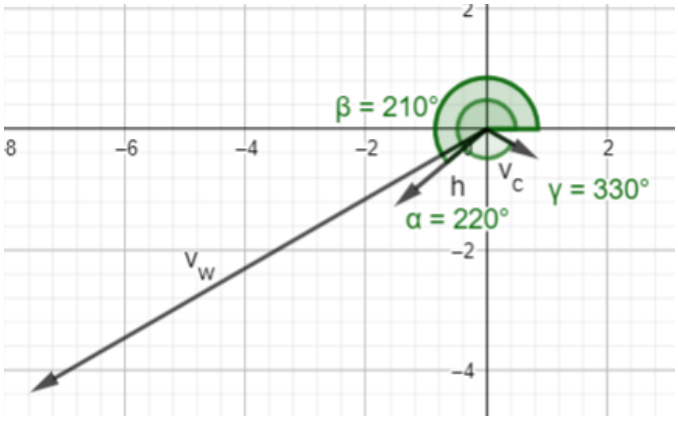

Fig. 11. Vectors of the environemntal conditions. *Errata*: α is actually 225 deg

D. Waypoints

The *waypoints* of the case studied are given in Table I and can be visualized in Fig. VIII-D along with the ship, the map and the resulting desired path.

TABLE I
Case *waypoints*

Waypoints	Position x $[m]$	Position y $[m]$)	Velocity u $[m/s]$
1	500	600	3
2	1000	900	3.5
3	1500	1600	4
4	2000	1900	4.5
5	2500	2600	5

E. Error metrics

The results for the case studied are given in Table II. Where: cte_m is the mean *cross-track* error; cte_{max} is the max *cross-track* error; we_m is the mean *width* error; we_{max} is the max *width* error. The *width* error like the *cross-track* error, but instead of measuring the distance from the center of mass of the ship, it is measured from the point of the ship furthest from the path-line.

TABLE II
Case metrics

Run	cte_m $[m]$	cte_{max} $[m]$	we_m $[m]$	we_{max} $[m]$
1	14.8	35.6	39.3	87.3
2	15.3	34.6	40.6	95.5
3	16.3	47.2	43.2	98.7

Fig. 12. Path described by the *waypoints*

F. Plots

The most relevant plots are presented in the figures: Fig. 13, Fig. 14, Fig. 15, Fig. 16, Fig. 17, Fig. 18, Fig. 19.

Fig. 13 represents the real and sensor surge velocities, whereas Fig. 14 represents the filtered and sensor surge velocities. Similarly, Fig. 15 represents the real and sensor yaw angles, and Fig. 16 represents the filtered and sensor yaw angles.

The pair of figures Fig. 17 and Fig. 18 represent the propeller rotation and rudder angle respectively. And finally, Fig. 19 represents both the *cross-track* and *width* errors.

G. Discussion

Regarding the path-following ability, it's possible to say that the task was successful, as shown by Table I. According to the eq. (2) and the ship's beam of $32.2m$, the acceptable average cross-track error would be of $16.1\ m$ or less. As the average cross-track error obtained was of $15.47\ m$, it's inside the acceptable range, and the task was successful.

As per surge control, the ability to reach the desired velocities at the *waypoints* wasn't exactly successful, but the ship managed to get close to the desired values nonetheless. One of the reasons for this could be related with the decoupled models and presence of low-frequency noise. It's important to add that for collinear paths, the surge control worked exactly as expected. This is also an indicator that yaw dynamics influenced a lot surge dynamics in the other case; because, with colliner *waypoints*, the yaw angle of the ship stabilizes[5] quickly and what remains is mainly surge dynamics with low influence of yaw dynamics.

The large radius of acceptance made the ship adapt itself earlier to the next path line, exhibiting a resultant smooth behavior that interpolated between the lines. This behavior was crucial for obtaining acceptable mean *cross-track* error and was beneficial for the actuators.

When it comes to the actuators, the propeller rotation displayed a smooth operation, with rare peaks of high frequency. The rudder angle, on the other hand, had a high frequency component throughout the whole trajectory. The last behavior can wear the rudder faster and demand great attention if the purpose is real-world implementation.

It is important to highlight that the initial yaw angle was favorable and the environmental condition was the one used while tuning the modules.

[5]Stabilizes in a general sense, it actually keeps oscillating with low amplitude around the desired yaw angle

IX. Conclusion

The following *ROS2* packages were developed, and contain general usage documentation:

- standalone maneuvering simulation package: *pydyna_simple*;
- path following package: *path_following*.

Therefore, a new infrastructure for *TPN* applications was presented. Through a proof of concept, in the form of a case study, for *fast-time* simulations. Consequently, canal design processes can become more robust and efficient. With this project, the TPN team has a starting point to kickoff *ROS2* efforts regarding maritime projects with large impact on society.

The code for this project can be found at http://github.com/BrunoScaglione/TCC-Autonomous-Ship.

References

[1] *ANNUAL OVERVIEW OF MARINE CASUALTIES ANDINCIDENTS*, European Maritime Safety Agency, 2017. [Online]. Available: https://www.isesassociation.com/wp-content/uploads/2017/11/Annual-overview-of-marine-casualties-and-incidents-2017_final.pdf

[2] A. Steigrad. (2021, Mar.) Giant ship blocking suez canal freed, but economic impact looms. NYPost. [Online]. Available: https://nypost.com/2021/03/29/giant-ship-blocking-suez-canal-freed-but-economic-impact-looms/

[3] L. G. Kofi. (2021, Apr.) The ever given – suez canal accident: The genesis to the current legal proceedings. Odomankoma. [Online]. Available: https://odomankoma.com/2021/04/19/the-ever-given-suez-canal-accident-the-genesis-to-the-current-legal-proceedings/

[4] (2016) How much did the panama canal cost to build. What It Costs. [Online]. Available: https://www.whatitcosts.com/panama-canal-cost-build

[5] (2020, Jun.) Expertise. Numerical Offshore Tank. [Online]. Available: https://tpn.usp.br/simulador/Expertise.html

[6] (2021, Jun.) Profile. Numerical Offshore Tank. [Online]. Available: https://www.linkedin.com/company/numerical-offshore-tank-tpn-/?originalSubdomain=br

[7] (2020, Jun.) Simulators. Numerical Offshore Tank. [Online]. Available: https://tpn.usp.br/simulador/Simulators.html

[8] (2020, Jun.) Technical studies. Numerical Offshore Tank. [Online]. Available: https://tpn.usp.br/simulador/TechnicalStudies.html

[9] R. Wiki, *Documentation*, 1st ed., ROS, Nov. 2020.

[10] (2020, Jun.) Homepage. Numerical Offshore Tank. [Online]. Available: https://tpn.usp.br

[11] J. Amendola, L. S. Miura, A. H. R. Costa, F. G. Cozman, and E. A. Tannuri, "Navigation in restricted channels under environmental conditions: Fast-time simulation by asynchronous deep reinforcement learning," *IEEE Access*, vol. 8, pp. 149 199–149 213, 2020.

[12] Z. Liu, Y. Zhang, X. Yu, and C. Yuan, "Unmanned surface vehicles: An overview of developments and challenges," *Annual Reviews in Control*, vol. 41, pp. 71–93, 2016.

[13] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd, apr 2011.

Fig. 13. Real and sensor surge velocity. Real surge velocity is in orange and sensor surge velocity is in blue.

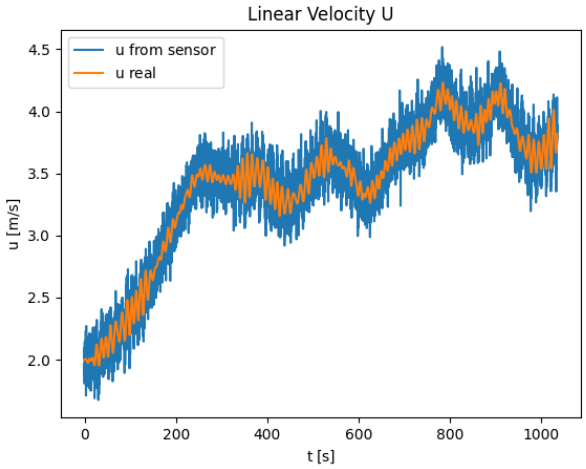

Fig. 14. Filtered and sensor surge velocity. Filtered surge velocity is in orange and sensor surge velocity is in blue.

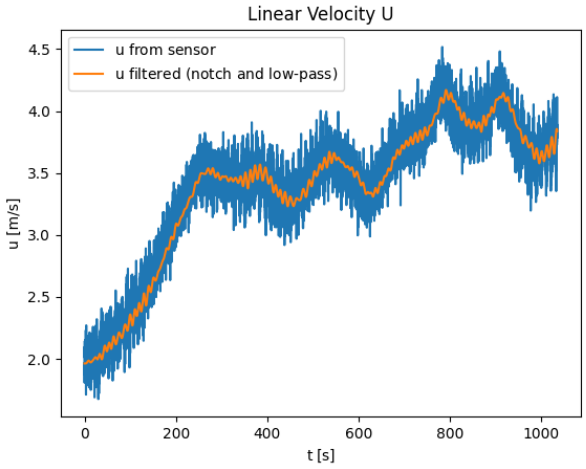

Fig. 15. Real and sensor yaw angle. Real yaw angle is in orange and sensor yaw angle is in blue.

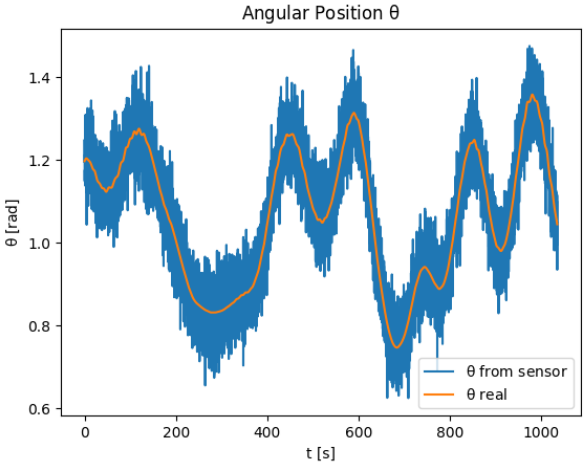

Fig. 16. Filtered and sensor yaw angle. Filtered yaw angle is in orange and sensor yaw angle is in blue.

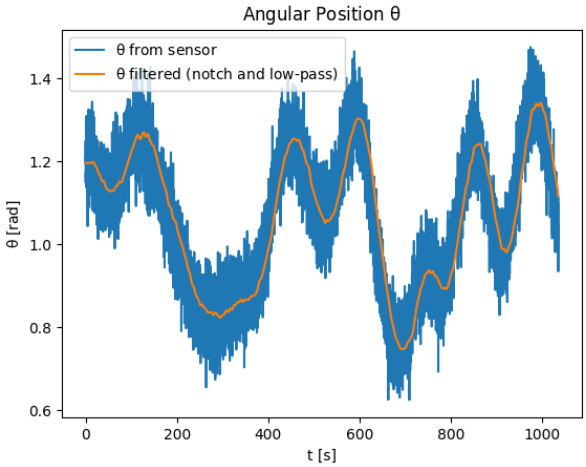

Fig. 17. Propeller rotation.

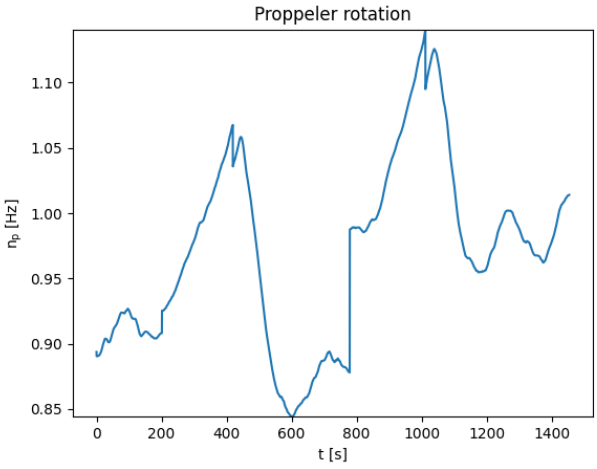

Fig. 18. Rudder angle.

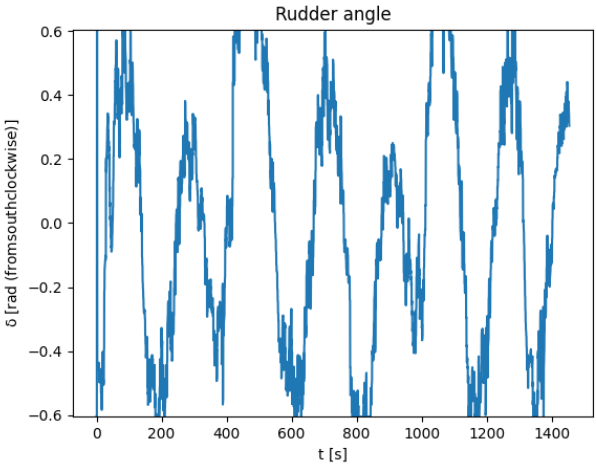

Fig. 19. *Cross-track* and *width* errors. *Cross-track* error is in blue and *width* error is in orange.