UNIVERSITA' POLITECNICA DELLE MARCHE

_____

FACOLTA' DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

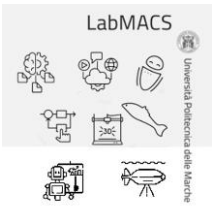
Laboratorio di Meccatronica

Studenti:	*Ruolo:*	*Docenti:*
Raffaele Berchicci	*Project Manager & Digital Twin Developer*	*David Scaradozzi*
Arianna Nazzarelli	*SOTA Supervisor & App Developer*	
Giuseppe Di Mauro	*Math Models Developer & Simulink Developer*	

PROJECT: SuperFin – Simulation, Control and User Interface for the Digital Twin of the "MAXFISH" Robotic Fish

PROGETTO: SuperFin - Simulazione, Controllo e Interfaccia Utente per il Digital Twin del Pesce Robotico "MAXFISH"

MAXFISH



Anno Accademico A.A.2023/2024

Table of Contents

Preface

Main Authors for the deliverable:

Name of partner organization	Main authors	Authors' role
UnivPM 	Raffaele Berchicci	Project Manager Digital Twin Developer
	Arianna Nazzarelli	SOTA Supervisor App Developer
	Giuseppe Di Mauro	Math Models Developer Simulink Developer

Supervisors for the deliverable:

Name of partner organization	Main contributors
UNIVPM - DII	David Scaradozzi
UNIVPM - SIMAU	Nicolò Ciuccoli
UNIVPM - DIISM	Daniele Costa

History of Changes:

Date	Version	History of change
16/10/2023	1.0	Initial version of the deliverable
30/10/2023	1.1	Minor corrections and structural changes, integrating input and remarks from main contributors.
20/11/2023	1.2	Change of "desiderata": the project is entirely in simulation.
04/12/2023	1.3	Further suggestions from main contributors: tail steering control is added.
11/12/2023	1.4	Official template received and minor remarks introduced in the App. Update: trajectory evaluation is defined.
18/12/2023	1.5	Final version integrating input and remarks from main contributors.

Introduction

Scope and objectives of the deliverable

The deliverables present the results of work done for the 3 months "Mechatronics Laboratory" course: the title of the project is "SuperFin - Simulation, Control and User Interface for the Digital Twin of the MAXFISH Robotic Fish."

In the course of the present project, we engaged in the in-depth study of the kinematic and dynamic behaviour of an articulated tail with three degrees of freedom, consisting of a configuration of three double gimbals and three links. This system, powered by a single motor inducing sinusoidal motion, underwent a detailed analysis aimed at understanding its kinematics and dynamics.

Subsequently, we extended this analysis to understanding the whole body of the fish, adopting the model proposed by Fossen. This step allowed us to integrate previously acquired information on the kinematics and dynamics of the tail into the evaluation of the global behaviour of the whole organism, opening new perspectives for control.

In addition, as part of this project, we developed a simulation of the fish using Simulink. This simulation provides the ability to execute a series of programmed trajectories, enabling an analysis of the dynamic behaviour of the fish itself. During this simulation, we are able to visualize the movement of the fish, thus realizing an accurate digital twin of the organism under investigation.

Finally, in order to facilitate the use of the simulation and allow greater control over the data analysis, we have developed an application (GUI). This intuitive interface gives users the ability to modify the physical parameters of the fish, such as masses and component lengths, and easily start the desired simulation.

At the end of the simulation, the App generates a series of graphs that allow analysis of positions and velocities and other parameters useful for understanding the Tool's behaviour, as well as simulation of tail kinematics.

The integrated product has been designed, verified, and validated:

- firstly, TCs and TPs have been introduced;
- then the product has been tested through the Factory Acceptance Tests (FATs), in a controlled and simulated environment;
- finally, an appropriate validation plan with GANTT charts and V&V model has been defined.

The main objective of validation is to demonstrate that the system is suitable for its targeted/intended purpose and operates in a smooth and reliable manner.

Regarding the methodology for the FAT, it is defined in Chapter 3a, while the methodological framework for validation is defined through demonstration at the operational environment.

The validation methodology mainly focuses on Key Performance Indicators **(KPIs)** and **instruments to be used** for:

(i) evaluating the **safety**, the **technical performance** and **reliability** of the integrated system (with both safety and technical/performance indicators);

(ii) assessing the value offered by the system in terms of its **appreciation** by domain stakeholders and end-users;

(iii) assessing the value offered by the system in terms of its **impact** on regular use.

Structure of the deliverable

The deliverable is structured as follows:

- *Chapter-1:* Describes the scope of work and main objectives while highlighting the challenges of the involved work
- *Chapter 2a:* Defines theoretical background and the state of the art of the proposed system
- *Chapter 2b:* Defines mechatronic infrastructure and the operational environment for its validation
- *Chapter 2c:* Defines digital-twin model or user interface of the proposed system and the operational environment for its validation
- *Chapter 3a:* Discusses the methodological framework for validation (TCs, TPs)
- *Chapter 3b:* Outlines the validation plan to be performed at operational sites (GANTT, V&V model)
- *Chapter 3c:* Provides the list of KPIs for technical evaluation
- *Chapter 3d:* Elaborates the indicators for End-User/Stakeholder acceptance and evaluation
- *Chapter 4:* Validation plan results in terms of KPIs, Technical evaluations, improvement between Verification and Validation phases.
- *Chapter 5:* Final conclusions and future improvements.
- *Bibliography*.
- *Appendix 1:* User Manual - Reports notes on software installation, configuration and use of
- the prototype.
- *Appendix 2:* Software - Reports the software directories and files summary with links to MS Teams and GitHub.
- *Appendix 3:* Questionnaires - Reports the forms and questionnaires to be used for the safety and technical assessment and for the user evaluation.

Scope of work and main objectives

In the design of this work, the main objective was to establish a solid foundation for the creation of a highly versatile tool that is essential for research activities in marine biology and underwater science.

The goal is to provide an advanced and controllable tool that can simulate in precise detail the movement of fish in different underwater contexts.

Therefore, we mainly focused on the implementation of a digital twin and an App GUI, pushing into the purely software domain to create a highly representative virtual model of fish movement.

The project is carried out in 5 phases:

1. Analysis of the robotic fish model presented in the article "A novel class of bio-inspired underwater robots" by Engineer Daniele Costa, analysis of the GUI fish model 6.0 shown in the thesis "Study and simulation of NCG systems for biomimetic robots" by Engineer Waqas Ali Waqar. These articles provided us with the theoretical expertise needed for the development of the project.

2. Study of MAXFISH robotic fish tail kinematics and implementation and simulation of robotic fish tail kinematics in MATLAB.

3. Implementation of the MAXFISH robotic fish model based on Fossen's underwater vehicle theory in Simulink.

4. MAXFISH control, specifically definition of a trajectory to be executed by the fish and implementation of a frequency PID controller for surge motion and a PID on the tail's bias for steering.

5. Implementation of a GUI application with MATLAB's "App Designer," specifically:
    a. Importing user input data from an Excel file;
    b. Starting the simulation "RoboticFish_NGC.slx";
    c. Plot of the variables of interest in the simulation;
    d. Plot of the trajectory taken by the fish and the desired trajectory;
    e. Calculation of the mean and Standard Deviation of the trajectory;
    f. Starting the simulation of the tail kinematics on Matlab.

We started from an initial research phase of TRL level 1, and then progressed to TRL 3, in fact we have an initial offering that is getting closer and closer to the desired practical application.

However, significant uncertainties remain regarding the maturity and effectiveness of the technology outside the controlled testing environment.

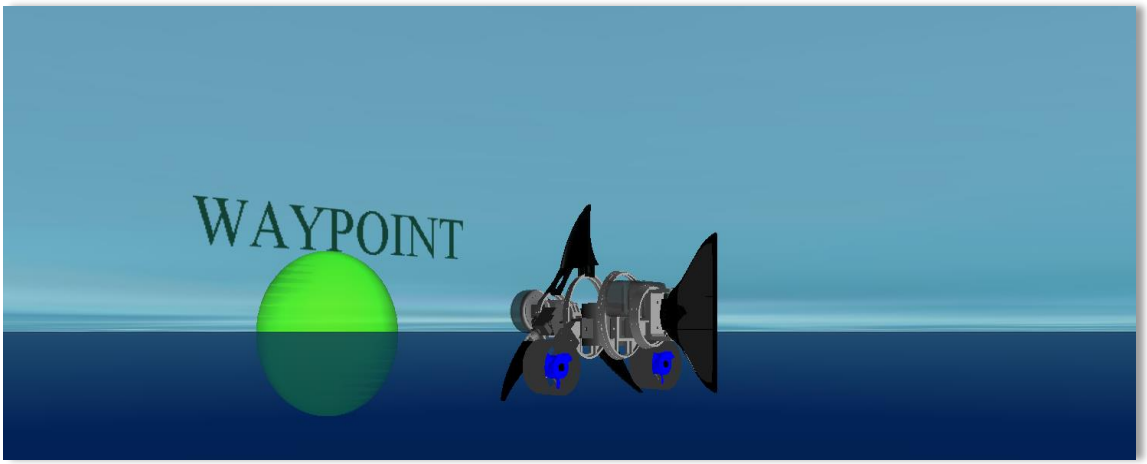


Figure 1 - MAXFISH in simulation environment

MAXFISH Implementation

2.a Theoretical background and the state of the art

2.a.1 A Novel Class of Bio-Inspired Underwater Robots

Autonomous Underwater Vehicles (AUVs) are robotic systems that operate independently in the marine environment, executing planned missions without human intervention. Efficient propulsion systems are critical for AUVs because it allows the vehicle to move and perform its mission and it influences power consumption and autonomy. Over the past three decades, bio-inspired solutions, inspired by aquatic animals' swimming abilities, have been explored for enhanced AUV maneuverability and efficiency. However, the propulsive efficiency achieved by biological swimmers remains a distant goal for artificial systems.

The fluid mechanics principles of aquatic animals' locomotion guide the creation of swimming AUVs. Thrust is generated through the relative motion between the fish body and water, due to the generation of a momentum. The swimming pattern of most marine animals is classified into body and caudal fin (BCF), which is made up of creating a backward-moving propulsive wave that extends to their caudal fin. Those marine animals with BCF swimming motion are divided into five swimming modes:

- Anguilliform: This type involves waves along the entire body, enabling backward movement by inverting the direction of the wave.
- Sub-carangiform: Similar to the Anguilliform model, but with a smaller percentage of the body involved in the undulation.
- Carangiform: Characterized by a stiffer body, these fish are faster compared to other models.
- Thunniform: This swimming model excels in terms of efficiency, enabling swimmers to maintain high cruising speeds for extended periods.
- Ostraciiform: Identified as the slowest and least efficient among the BCF swimming modes.

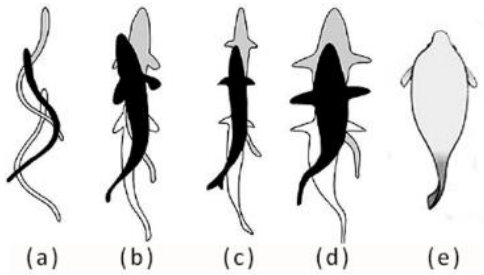


Figure 2 - BCF swimming modes: (a) anguilliform, (b) subcarangiform, (c) carangiform, (d) thunniform, (e) ostraciiform

In particular, thunniform swimming model represents a compromise choice for a bioinspired marine vehicle because of its efficiency.

The AUV analysed in SuperFin project consists of a rigid body and a tail composed of a three-joint mechanism, each of which oscillates according to the following harmonic function:

$$p_j(t) = a_j \cos(2\pi f t + \phi_j)$$

Where p_j represents the angle between link j with respect to the reference system at the base of the tail, a_j the amplitude of the oscillation, f the frequency and ϕ_j the constant phase difference between the laws of harmonic motion of the links.

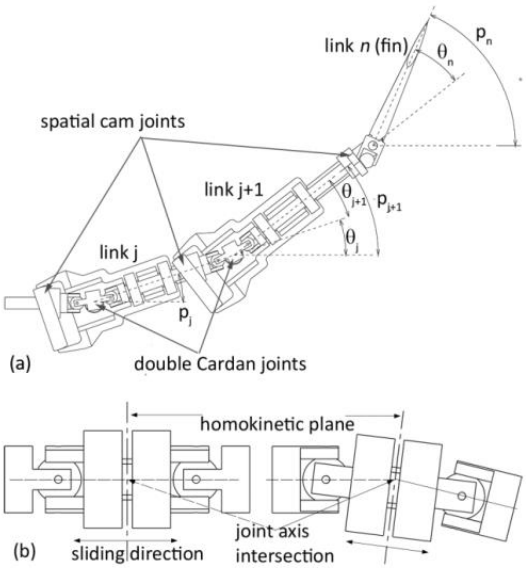
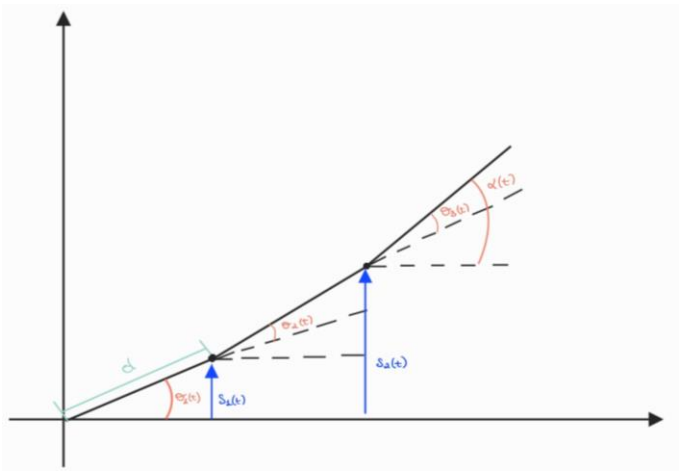


Figure 3 - (a) CAD model of the transmission system. (b). Double Cardan joint: unbent tail (left), bent (right)

The transmission mechanism is able to transform the continuous rotation of the input shaft into a harmonic oscillation of the output shaft. This is achieved by means of spatial-cam kinematic joints actuated by a single motor in order that those joints have the same input shaft which passes through the entire robotic tail, while spinning at a consistent angular velocity.

This configuration ensures that the links oscillate at the same frequency (f), meeting the synchronization requirement between joints. To accommodate the bending of the robotic tail according to the propulsive wave pattern, Double Cardan joints are strategically placed along the shaft, corresponding to each joint. These joints facilitate the transmission of driving torque through a variable angle while maintaining a constant rotational velocity.

As said before, the MAXFISH is made of a rigid body and a tail divided into three links (where the third linkage is the caudal fin), actuated by a single motor. Thus, the tail system has three degrees of freedom, represented by the three joints: $\vartheta_1(t)$, $\vartheta_2(t)$, $\vartheta_3(t)$, which are respectively the angle of displacement between the reference system and link one, the angle of displacement between link one and two and the angle of displacement between link two and three. Considering the Cartesian space, there are three variables $s_1(t), s_2(t)$ and $\alpha(t)$, respectively the variance between the x-axis and the end of the first link of tail, the variance between the x-axis and the end of the second link of tail and the angle between the x-axis and the caudal fin (third link) [Figure 3].



Cartesian space

$$p = \begin{bmatrix} s_1 \\ s_2 \\ \alpha \end{bmatrix}$$

Joint space

$$q = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Figure 4 - Design of MAXFISH tail

Now, the kinematic model of the robotic tail can be described, figure out both direct and inverse kinematics. The direct kinematics delineate the three Cartesian variables as functions of the three joints. Conversely, the inverse kinematics perform the reverse operations, determining the joint configurations based on the desired Cartesian variables.

$$\begin{cases} s_1 = d\sin(\theta_1) \\ s_2 = d\sin(\theta_1) + d\sin(\theta_1 + \theta_2) \\ \alpha = q_1 + q_2 + q_3 \end{cases}$$

$$\begin{cases} \theta_1 = \operatorname{asin}\left(\frac{s_1}{d}\right) \\ \theta_2 = \operatorname{asin}\left(\frac{s_2 - s_1}{d}\right) - \operatorname{asin}\left(\frac{s_1}{d}\right) \\ \theta_3 = a - \operatorname{asin}\left(\frac{s_2 - s_1}{d}\right) \end{cases}$$

Where d is the length of the pieces of tail.

These systems are obtained by simple geometrical transformation and math calculations by looking at [Figure 4].

To compute the velocity and acceleration of the individual joints at each time instant t, the Jacobian was calculated with the *Jacobian* function in MATLAB.

$$J = \begin{bmatrix} d * \cos(\theta_1) & 0 & 0 \\ d * \cos(\theta_1 + \theta_2) + d * \cos(\theta_1) & d * \cos(\theta_1 + \theta_2) & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

After that, the velocity in both joint space and operational space through the Jacobian was determined. The velocity in joint space refers to the speed at which each joint moves. The

representation of this velocity is useful for controlling and understanding the movement of individual parts of the AUV. Velocity in operating space refers to the speed with which the end-effector moves in its operating environment. Understanding velocity in operating space is crucial for controlling the movement of the AUV in relation to surrounding objects and surfaces.

$$\dot{x} = \{J\}\dot{\theta}$$

$$\dot{\theta} = J^{-1}\dot{x}$$

Similarly, acceleration was calculated in the joint space and in the operating space, by using the Jacobian:

$$\ddot{\theta} = J^{-1}(\ddot{x} - \dot{J}\dot{\theta})$$

$$\ddot{x} = \{J\}\ddot{\theta} + \{\dot{J}\}\dot{\theta}$$

Finally, the determinant of the Jacobian was calculated so that singularities could be detected. The determinant is non-zero so there are no points of singularity.

As mentioned previously, thunniform locomotion exhibits the highest propulsive efficiency among BCF swimming modes. In this mode, the caudal fin executes a roto-translation motion known as flapping, driven by the undulation of the tail. Specifically, the fin follows the undulating path depicted in [Figure 6a] to modulate its angle of attack and prevent flow separations, which can lead to efficiency losses.

In order to have the caudal fin follow the desired trajectory, the drive mechanism must be composed as follows: the central component of the system is the spatial-cam kinematic joint depicted in [Figure 5a]. In this arrangement, driving disk A includes a drive-sphere B that interacts with the rectangular groove of the driven member C. The oscillating output shaft is firmly linked to driven member C and can freely pivot in support block D. The maximum rotation of the output shaft is twice the angle θ_0, as shown in [Figure 5b]. In a broader context, the output angle θ and the motor rotation φ are connected by the following expression:

$$\tan(\theta) = \frac{h}{L}\cos(\varphi) = \lambda\cos(\varphi)$$

Provided that λ is smaller than one, this expression can be approximated by:

$$\theta \approx \lambda\cos(\varphi) = \lambda\cos(\omega t)\ , \qquad \omega = \dot{\varphi}$$

Where h and L are geometric parameters [Figure 5b] and λ_i can be expressed as in the follow:

$$\lambda_i = \frac{h_i}{L_i} \approx \theta_0$$

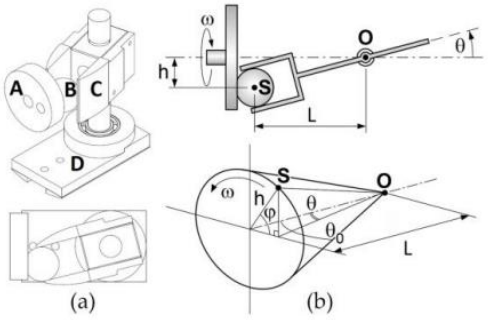


Figure 5 - (a) CAD model of the spatial cam kinematic joint. (b) Functional scheme.

The motion of a flapping caudal fin is characterized by the following expressions.

$$s(t) = s_0 \cos(2\pi f t) = \sum_{i=1}^{2} d_i \sin\left(\sum_{k=1}^{i} \theta_k\right) \approx \sum_{i=1}^{2} d_i \left(\sum_{k=1}^{i} \lambda_k \cos(2\pi f t + \delta_k)\right)$$

$$\alpha(t) = \alpha_0 \cos(2\pi f t) = \sum_{i=1}^{3} \theta_i \approx \sum_{i=1}^{3} \lambda_i \cos(2\pi f t + \delta_i)$$

Where d_i is the lengths of the links, s_0 and α_0 are the amplitudes of the harmonic functions, f is their common frequency.

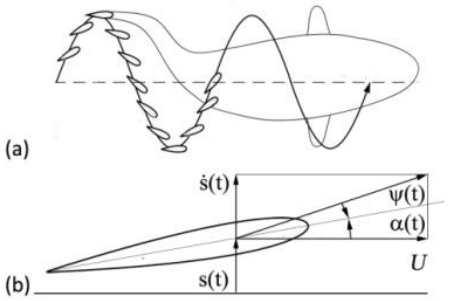


Figure 6 - (a) Caudal fin undulating wave pattern in thunniform locomotion. (b) Flapping motion components and instantaneous angle of attack

The $s(t)$ equation of indicates that the translation amplitude, θ_0 , is maximized when the initial rotations of the first two links are equal. Due to the necessity to maintain geometric parameters λ_1 and λ_2 smaller than one, while maximizing θ_0, this leads to the in-phase oscillation of the first two tail members. In this way, the tail linkage oscillates in a manner similar to a fluke, resembling the observed behavior in the caudal peduncle of biological counterparts. For convenience, δ_1 and δ_2 will be set to zero.

After solving the previous equations, the cam joint parameters are obtained as follows:

$$\lambda_{1,2} = \frac{s_0}{3d}$$

$$tan(\delta_3) = \frac{\alpha_0 \sin(\Delta)}{\alpha_0 \cos(\Delta) - 3\lambda_{1,2}}$$

$$\lambda_3^2 = \alpha_0^2 + 9\lambda_{1,2}^2$$

where $\lambda_{1,2}$ represents the common value of λ_1 and λ_2, *d* is the value associated to d_1 and d_2 and Δ is the phase shift between the components of the flapping motion.

To provide the model, it has been chosen the following values for the parameters:

- $\psi_{max} = 15°$
- $\alpha_0 = 23°$
- $c = 100\ mm$
- $s_0 = 1.5c$
- $f = 1.67\ Hz$
- $\Delta = 90°$
- $\lambda_{1,2} = 0.16$
- $\delta_{1,2} = 0\ rad$
- $d = 100\ mm$
- $\lambda_3 = 0.47$
- $\delta_3 = 2.13\ rad$

As shown in [Figure 7], to analyse the behaviour of the tail of the robotic fish, it was approximated to a trapezoid. Thanks to the studies previously carried out by Engineer Daniele Costa, it has been discovered a quadratic relationship between the average thrust coefficient and the Strouhal number (*St*).

The Strouhal number (*St*) is a dimensionless parameter commonly used in fluid dynamics to characterize oscillating flows and phenomena, particularly in the context of vortex shedding.

The Strouhal number is defined as in the follows:

$$St = \frac{2s_0 f}{U}$$

Where *f* is the frequency of oscillation, s_0 is the translation amplitude, $U = [u \quad v \quad w]^T$ is the module of the robotic fish's linear speed.

Thus, the average thrust coefficient varies proportionally with the square of the Strouhal number. Additionally, numerical predictions affirm that the propulsive force's trends over a flapping cycle can be effectively approximated by a harmonic function featuring a constant phase-shift. This significant observation implies that the thrust force generated by a three-dimensional flapping fin, characterized by a rectangular planform, can be accurately expressed using the following mathematical expression:

$$T_R = \frac{1}{2}\rho U^2 C_T c b$$

Where C_T is the thrust coefficient and it is defined as:

$$C_T = K_T(s_0, \alpha_0)St^2(4\pi ft + \beta)$$

ρ is the water density, c is the fin chord and b is its span.

In this scenario, the computation of propulsive thrust involves the dependency of K_T on both the rotation and translation amplitudes of the fin motion, with a constant phase shift β. is versatile, extending its applicability to calculate the propulsive thrust generated by a trapezoidal, tapered-planform caudal fin, as illustrated in [Figure 7].

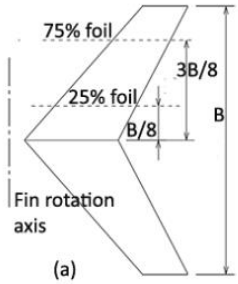


Figure 7 - Caudal fin geometry and foils definition

Drawing insights from aircraft theory, propulsive thrust is determined by partitioning the fin into multiple sections and approximating each section with a rectangular wing. The thrust force computation specifically involves dividing the flapping fin with two horizontal planes, positioned at 25% and 75% of its span (B/2), as depicted in [Figure 7].

The origin of the moving reference frame coincides with the revolute joint linking the second link of the tail to the caudal fin. In simpler terms, the fin's flapping motion is induced by the moving reference frame, while the freestream velocity boundary condition accommodates the swimming motion of the robotic system. This careful alignment ensures that the computational model appropriately captures the dynamic interactions and fluid behaviours associated with the flapping fin.
In light of the aforementioned considerations, the computation of the thrust force can be expressed by the following equation:

$$T_{FIN} = \frac{1}{2}\rho U^2(C_{25}c_{25} + C_{75}c_{75})\, B/2$$

Where C_{25} and C_{75} represent the thrust coefficients of the foils obtained from the horizontal sections of the fin, while c_{25} and c_{75} are the respective chord lengths of these foils. Nevertheless, since both foils share the same translation amplitude due to the rigid motion of the fin, they are characterized by the same value of the Strouhal number (*St*), which is a property of the entire fin.

Given this, the expression for the Strouhal number (*St*) can be substituted into the expression of the thrust coefficient, and subsequently, the resulting equation can be employed in the equation of the thrust force as follows:

$$T_{FIN} = \rho s_0^2 f^2(K_{T25}c_{25} + K_{T75}c_{75})Bsin(4\pi ft + \beta)$$

In this context, the coefficients K_{T25} and K_{T75} are influenced by the shape and motion of the foil, specifically by the translation and rotation of the fin. Notably, these coefficients remain unaffected by the dimensions of the foil, which are represented by the chord length. In situations where the fin sections are uniform except for a scale factor, as observed for the fin of MAXFISH, the K_T coefficient can be factored out in the thrust force's equation. This simplification results in the following expression:

$$T_{FIN} = 2\rho s_0^2 f^2 K_T(s_0, \alpha_0)\, S\, sin(4\pi f t + \beta)$$

where S is the fin planform area and $K_T(s_0, \alpha_0) = 2.43$, as predicted by the numerical simulations made by engineer Daniele Costa.

The simplified approach employed to compute the propulsive thrust can be extended to calculate both the sway force and the yawing torque generated by the fin flapping. Therefore, the sway force and the yawing torque equations are the following ones:

$$L_{FIN} = 2\,\rho s_0^2 f^2 K_L\,(s_0, \alpha_0)\, S\, sin(4\pi f t + \varsigma)$$

$$M_{FIN} = 2\,\rho s_0^2 f^2 K_M\,(s_0, \alpha_0)\, S c_{av} \sin(2\pi f t + \xi)$$

Where c_{av} is the fin average chord length, $K_L\ and\ K_m$ are the sway force and yawing torque coefficients, ς and ξ the corresponding phase shifts.

2.a.2 Fossen mathematical model of MAXFISH

The mathematical model of the MAXFISH robotic fish is based on the theory of underwater vehicles developed by Norwegian Thor I. Fossen in his book 'Handbook of Marine Craft Hydrodynamics and Motion Control'. According to this theory, the non-linear dynamic model of MAXFISH has six degrees of freedom.

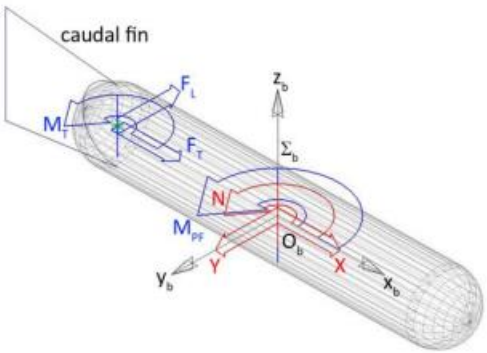


Figure 8 - Propulsive (blue) and hydrodynamic (red) loads acting on a cylindrical swimming robot.

Firstly, for the analysis of the kinematics and dynamics of the AUV MAXFISH, it was considered to consist of a cylindrical body and a caudal fin connected to the body through a rotational joint. As can be seen in [Figure 8], the basic reference system Σ_B: $0_B - x_B y_B z_B$ coincides with the centre of the body and the centre of gravity of the robotic fish. The linear velocity of the origin point 0_B is

expressed by the vector $v_1 = [u\ v\ w]^T$, while the angular velocity of the same is expressed by the vector $v_2 = [p\ q\ r]^T$.

The equation of motion for the marine vehicle can be expressed in a vectorized form, following the framework presented by Fossen, as:

$$\dot{\eta} = J_\Theta(\eta) * v$$

$$M\dot{v} + C(v)v + D(v)v + g(\eta) = \tau$$

Where in the subsequent paragraphs, detailed descriptions of the various matrices, vectors, and their inherent properties will be provided.

2.a.2.1 Kinematics

Kinematics is the branch of classical mechanics that focuses on the study of motion, without considering the forces that cause the motion. It involves the analysis of the spatial and temporal properties of motion, describing how objects move and the patterns of their displacement, velocity, and acceleration.

During the analysis of a marine robot's motion in six degrees of freedom, it is advantageous to establish two coordinate frames, as illustrated in Figure 9. The mobile frame, denoted as $\{b\}$, remains fixed to the craft's body, with its origin $\{O_b\}$ conventionally positioned at the vehicle's center of gravity.

The motion of the body-fixed frame is described in relation to the Earth-centered frame, known as the North-East-Down (NED) frame $\{n\}$, which can be treated as inertial due to the negligible impact of Earth's motion on the vehicle at low speeds. Consequently, it is recommended to express the position and orientation of the robotic fish relative to the $\{n\}$ frame, while linear and angular velocities should be represented in $\{b\}$.

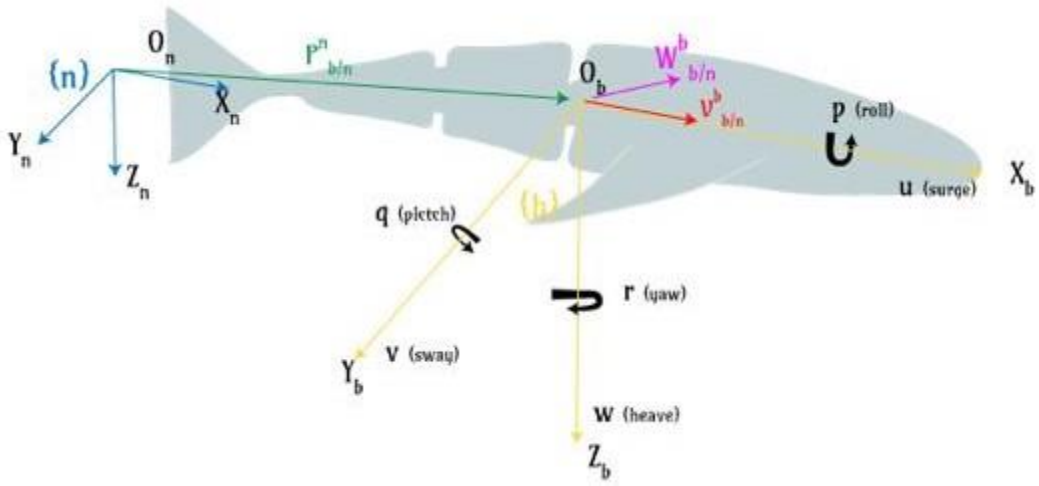


Figure 9 - MAXFISH reference frames: {b} (body-fixed) and {n} (NED).

The notation employed for vectors in $\{b\}$ and $\{n\}$ frames for the marine robot is as follows:

- $v_{b/n}^{b} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$: linear velocity of point O_b relative to $\{n\}$ expressed in $\{b\}$;
- $w_{b/n}^{b} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$: angular velocity of point O_b relative to $\{n\}$ expressed in $\{b\}$;
- $p_{b/n}^{n} = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix}$: position of O_b in the $\{n\}$ frame;
- $\Theta_{nb} = \begin{bmatrix} \phi \\ \theta \\ \varphi \end{bmatrix}$: Euler angles between $\{n\}$ and $\{b\}$.

The kinematics model delineates the geometric relationship between the two reference frames, $\{b\}$ and $\{n\}$. These relationships are articulated through the vectors η and υ.

$$\eta = \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} = \begin{bmatrix} p_{b/n}^{n} \\ \Theta_{nb} \end{bmatrix}, \quad \upsilon = \begin{bmatrix} \upsilon_1 \\ \upsilon_2 \end{bmatrix} = \begin{bmatrix} v_{b/n}^{b} \\ w_{b/n}^{b} \end{bmatrix}$$

Here, η represents the position and attitude of the craft, where the position vector $p_{b/n}^{n}$ denotes the distance between the NED frame and the body-fixed frame expressed in the NED coordinate frame $\{n\}$. Θ_{nb} is the Euler angles vector (roll (ϕ), pitch (θ), and yaw (ψ)) and υ denotes the linear and angular velocity vector.

If the vectors υ_1 and υ_2 are provided, it is feasible to determine the time derivative of the position and attitude vectors η_1 and η_2 through the following transformation:

$$\begin{bmatrix} \dot{\eta}_1 \\ \dot{\eta}_1 \end{bmatrix} = \begin{bmatrix} J_1(\eta_2) & 0_{3\times3} \\ 0_{3\times3} & J_2(\eta_2) \end{bmatrix} \begin{bmatrix} \upsilon_1 \\ \upsilon_2 \end{bmatrix}$$

where $J_1(\eta_2)$ and $J_2(\eta_2)$ are the transformation matrices.

EULER PARAMETERS REPRESENTATION (UNIT QUATERNIONS)

Euler parameters, or unit quaternions, emerge as a sophisticated and efficient solution for AUV navigation. Their ability to avoid singularities, provide a compact yet powerful representation, and enable seamless transitions make them well-suited for the challenges presented by underwater environments.

Euler parameters provide a compact and efficient means of representing three-dimensional orientation, offering a four-parameter mathematical structure. These parameters, denoted as $q = [q_0\ q_1\ q_2\ q_3]^T$, adhere to the unit quaternion constraint $\|q\| = 1$. This unit constraint ensures a singularity-free representation, a crucial advantage in dynamic underwater environments where traditional representations like Euler angles may encounter challenges. In fact, the transformation matrix $J_2(\eta_2)$ is undefined at pitch angles $\theta = \pm 90°$ and becomes ill conditioned near these singularities.

Furthermore, Euler parameters avoid the gimbal lock problem associated with Euler angles, where certain orientations lead to a loss of one degree of freedom. In the context of AUVs, navigating

through complex underwater terrain demands continuous and smooth orientation changes, making Euler parameters a more robust choice.

Thus, the linear velocity transformation $\dot{\eta}_1$ is defined as:

$$\dot{\eta}_1 = R_b^n(q)v_1$$

$$\text{where} \quad R_b^n(q) = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_3q_0) & 2(q_1q_3+q_2q_0) \\ 2(q_1q_2-q_3q_0) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_1q_0) \\ 2(q_1q_3-q_2q_0) & 2(q_2q_3-q_1q_0) & 1-2(q_1^2+q_2^2) \end{bmatrix}.$$

And the angular velocity transformation $\dot{\eta}_2$ can be expressed as:

$$\dot{\eta}_2 = \dot{q} = T_q(q)v_2$$

where

$$T_q(q) = \frac{1}{2}\begin{bmatrix} -\vec{q}^T \\ q_0 I_{3\times3} + S(\vec{q}) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix}$$

and S: $\mathbb{R}^3 \to \mathbb{R}^{3\times3}$ is the skew-symmetric operator.

Hence, the kinematics equation can be formally represented through the utilization of Euler parameters as follows:

$$\begin{bmatrix} \dot{p}_{b/n}^n \\ \dot{q} \end{bmatrix} = \begin{bmatrix} R_b^n(q) & 0_{3\times3} \\ 0_{4\times3} & T_q(q) \end{bmatrix} \begin{bmatrix} v_{b/n}^b \\ w_{b/n}^b \end{bmatrix} \Leftrightarrow \dot{\eta} = J_q(\eta)v$$

2.a.2.2 Dynamics

The dynamics analysis encapsulates the intricate relationship between the forces acting upon a robot and the subsequent motion exhibited by the robotic system. It involves a comprehensive analysis of how external forces, internal torques, and varying conditions influence the robot's movement, in this case MAXFISH.

Before analysing the dynamics of the robotic fish, it is necessary to consider that, unlike a robot moving in an empty space, MAXFISH is a marine vehicle and therefore exposed to hydrodynamic forces. By hydrodynamic forces it is meant forces and moments due to the presence of a fluid (in this case water) in which the vehicle is submerged.

The hydrodynamic forces exerted on the vehicle can be categorized into various components: disturbances stemming from the environment (wind, waves, sea currents), restoring forces attributed to gravity and buoyancy, and forces associated with the added mass resulting from surrounding fluid inertia and potential damping induced by the energy dissipated from surface-generated waves.

Nevertheless, this added mass should not be considered as a quantity of fluid physically attached to the vehicle, forming a combined system with a mass greater than the original. Instead, it should be interpreted as the forces generated by pressure resulting from a compelled harmonic motion of the body, proportional to the body's acceleration.

Considering this array of forces, the equation of motion for the marine robot, endowed with 6 degrees of freedom in the body-fixed frame, can be expressed as follows:

$$(M_{RB} + M_A)\dot{v} + \big(C_{RB}(v) + C_A(v)\big)v + D(v)v + g(\eta) = \tau_E + \tau$$

Where $v = [u, v, w, p, q, r]^T$ and $\dot{v} = [\dot{u}, \dot{v}, \dot{w}, \dot{p}, \dot{q}, \dot{r}]^T$ are respectively the vector of linear and angular velocity and the vector of linear and angular accelerations expressed in reference frame Σ_B, while $\tau = [X, Y, Z, K, M, N]^T$ is the vector of the external forces and moments exerted by the thrusters. Environmental forces and moments τE are neglected (thus τE = 0).

In the case of MAXFISH, however, the vector of external forces is $\tau = [X, Y, 0,0,0, N]^T$ since only the plane motion of the robotic fish was implemented, as it uses only the caudal fin to generate motion. Therefore, the force along the Z-axis and the moments generated by the rotation along the Roll and Pitch angle are set to zero.

M_{RB} represents the inertia matrix of the rigid body, which is unique and satisfies the following properties:

$$M_{RB} = M_{RB}^T > 0 \;\; and \; \dot{M}_{RB} = 0$$

Thus, M_{RB} is defined as positive, symmetrical, and constant**.**

$$M_{RB} = \begin{bmatrix} mI_{3x3} & -mS(r_G) \\ mS(r_G) & I_0 \end{bmatrix}$$

Where m is MAXFISH's mass and $r_G = \begin{bmatrix} x_G \\ y_G \\ z_G \end{bmatrix}$ is a 3 × 1 vector indicating the displacement of the centre of gravity CG relative to the body-fixed frame Σ_B. As stated above, the centre of gravity and the reference system Σ_B were chosen to be coincident, therefore $r_G = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$.

I_{3x3} is the identity matrix, while I_0 is the inertia tensor of the robotic fish in relation to the body-fixed frame, which is symmetric and defined as positive ($I_o = I_o^T > 0$).

$$I_0 = \begin{bmatrix} I_x & -I_{xy} & -I_{xz} \\ -I_{yx} & I_y & -I_{yz} \\ -I_{zx} & -I_{zy} & I_z \end{bmatrix}$$

Therefore, the inertia matrix of the rigid body is composed as in the follow:

$$M_{RB} = \begin{bmatrix} mI_{3x3} & 0_{3x3} \\ 0_{3x3} & I_0 \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & -I_{xy} & -I_{xz} \\ 0 & 0 & 0 & -I_{yx} & I_y & -I_{yz} \\ 0 & 0 & 0 & -I_{zx} & -I_{zy} & I_z \end{bmatrix}$$

M_A is the added mass inertia matrix, which is defined by the by the derivatives of the components of the vector τ with respect to linear and angular accelerations $\dot{v}$:

$$M_A = -\begin{bmatrix} X_{\dot{u}} & X_{\dot{v}} & X_{\dot{\omega}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{v}} & Y_{\dot{\omega}} & Y_{\dot{p}} & Y_{\dot{q}} & Y_{\dot{r}} \\ Z_{\dot{u}} & Z_{\dot{v}} & Z_{\dot{\omega}} & Z_{\dot{p}} & Z_{\dot{q}} & Z_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{v}} & K_{\dot{\omega}} & K_{\dot{p}} & K_{\dot{q}} & K_{\dot{r}} \\ M_{\dot{u}} & M_{\dot{v}} & M_{\dot{\omega}} & M_{\dot{p}} & M_{\dot{q}} & M_{\dot{r}} \\ N_{\dot{u}} & N_{\dot{v}} & N_{\dot{\omega}} & N_{\dot{p}} & N_{\dot{q}} & N_{\dot{r}} \end{bmatrix}$$

Thus, i.e. the hydrodynamic added mass force X along the x_B axis due to an acceleration $\dot{u}$ in the x direction is written as

$$X_A = -X_{\dot{u}}\,\dot{u}\,,\; X_{\dot{u}} = \frac{\partial X}{\partial \dot{u}}$$

Typically, all components of the matrix M_A are non-null. However, considering the relatively low speeds at which the vehicle operates and adopting the simplifying assumption that the vehicle's body is cylindrical, a significant portion of hydrodynamic effects does not significantly impact the resulting motion. This characteristic is verified also for $C_A(v)$ and $D(v)$ as well.

Consequently, the added mass inertia matrix is reduced to**:**

$$M_A = -\begin{bmatrix} X_{\dot{u}} & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{\dot{v}} & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{\dot{\omega}} & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{\dot{p}} & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{\dot{q}} & 0 \\ 0 & 0 & 0 & 0 & 0 & N_{\dot{r}} \end{bmatrix}$$

Where $M_A = M_A^T > 0 \; and \; \dot{M}_A = 0$.

Therefore, the total inertia matrix of MAXFISH is the sum of M_{RB} and M_A:

$$M = M_{RB} + M_A$$

$C_{RB}(v)$ is the Coriolis and centripetal matrix of the rigid body and it is a skew-symmetrical matrix, so $C_{RB}(v) = -C_{RB}^T(v) > 0$.

$C_{RB}(v)$ is defined as in the follow:

$$C_{RB}(v) = \begin{bmatrix} 0_{3x3} & -mS(v_1) - mS(S(v_2)r_G) \\ -mS(v_1) - mS(S(v_2)r_G & mS(S(v_1)r_G) - S(I_0 v_2) \end{bmatrix} = \begin{bmatrix} 0_{3x3} & -mS(v_1) \\ -mS(v_1) & -S(I_0 v_2) \end{bmatrix}$$

$C_A(v)$ is the added-mass Coriolis and centripetal matrix and it's defined as follows:

$$C_A(v) = \begin{bmatrix} 0_{3x3} & -S(M_{11}v_1 + M_{12}v_2) \\ - - S(M_{11}v_1 + M_{21}v_2) & -S(M_{21}v_1 + M_{22}v_2) \end{bmatrix} =$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & -Z_{\dot{\omega}}\,\omega & Y_{\dot{v}}\,v \\ 0 & 0 & 0 & Z_{\dot{\omega}}\,\omega & 0 & -X_{\dot{u}}\,u \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -Z_{\dot{\omega}}\,\omega & Y_{\dot{v}}\,v & 0 & -N_{\dot{r}}\,r & M_{\dot{q}}\,q \\ Z_{\dot{\omega}}\,\omega & 0 & -X_{\dot{u}}\,u & N_{\dot{r}}\,r & 0 & K_{\dot{p}}\,p \\ -Y_{\dot{v}}\,v & X_{\dot{u}}\,u & 0 & -M_{\dot{q}}\,q & -K_{\dot{p}}\,p & 0 \end{bmatrix}$$

Where $C_A(v) = C_A^T(v) > 0 \; and \; \dot{C}_A(v) = 0.$

$D(v)$ is total hydrodynamic and centripetal matrix.

In high-speed motion, the damping characteristics of a marine vehicle are typically nonlinear and interconnected. One feasible approach involves approximating the marine robot's motion as uncoupled, wherein second-order terms are deemed negligible. This approximation results in a simplified structure for the matrix $D(\boldsymbol{v})$, composed only by linear and quadratic damping terms positioned along the diagonal:

$$D(v) = -\begin{bmatrix} X_u + X_{u|u|}|u| & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_v + Y_{v|v|}|v| & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_w + Z_{w|w|}|w| & 0 & 0 & 0 \\ 0 & 0 & 0 & K_p + K_{p|p|}|p| & 0 & 0 \\ 0 & 0 & 0 & 0 & M_q + M_{q|q|}|q| & 0 \\ 0 & 0 & 0 & 0 & 0 & N_r + N_{r|r|}|r| \end{bmatrix}$$

Moreover, under the assumption that the vehicle body is cylindrical, it has been considered that the linear damping terms are zero, so that $X_u = Y_v = Z_w = K_p = M_q = N_r = 0$.

$$D(v) = -\begin{bmatrix} X_{u|u|}|u| & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{v|v|}|v| & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{w|w|}|w| & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{p|p|}|p| & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{q|q|}|q| & 0 \\ 0 & 0 & 0 & 0 & 0 & N_{r|r|}|r| \end{bmatrix}$$

Furthermore, when a rigid body is fully or partially submerged in a fluid, two additional forces come into play: gravitational force and buoyancy force. The gravitational force f_G is generated by the weight W and it takes action on the center of gravity $r_G = \begin{bmatrix} x_G \\ y_G \\ z_G \end{bmatrix}$ of MAXFISH, while the buoyance

force f_B is generated by buoyancy B and it takes action on the center of buoyancy $r_B = \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix}$ of the robotic fish.

Let m represent the mass of the vehicle, Δ denote the volume of the fluid displaced, g symbolize the acceleration due to gravity, and ρ represent the density of the water. Consequently:

$$W = mg$$

$$B = \rho g \Delta$$

Both of these forces act along the z_b axis, with the first (W) exerting a positive sign and the second (B) a negative sign. Additionally, applying the transformation matrix based on the Euler angles allows the definition of the restoring force concerning the body-fixed frame as follows:

$$g(\eta) = \begin{bmatrix} (W-B)\sin(\theta) \\ (W-B)\cos(\theta)\sin(\phi) \\ (W-B)\,cos\theta\cos(\phi) \\ -(y_G W)\cos(\theta)\cos(\phi) + z_G W)\cos(\theta)\cos(\phi) \\ (z_G W)\sin(\theta) + (x_G W)\cos(\theta)\cos(\phi) \\ -(x_G W)\cos(\theta)\sin(\phi) + (y_G W)\sin(\theta) \end{bmatrix}$$

Under the given assumption $W = B$ e $xg = yg = 0$, so:

$$g(\eta) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ (z_G W)\cos(\theta)\cos(\phi) \\ (z_G W)\sin(\theta) \\ 0 \end{bmatrix}$$

2.a.2.3 Propulsion system

The MAXFISH robotic fish is driven by a single motor, which is responsible for the movement of the caudal fin.

As said before, the average thrust coefficient has a quadratic relationship with the Strouhal number (*St*).

$$St = \frac{2s_0 f}{U} = f\frac{2c}{U}\tan(\theta_0)$$

Where *c* is the value of the mean fin's chord, *f* the oscillation frequency, *U* is the fish swimming velocity and θ_0 is the amplitude of oscillation.

The efficiency of the actuator has been investigated by examining its behavior across varying Strouhal numbers. This analysis enables the modeling of hydrodynamic forces and torque within a single oscillation period, expressed as a function of the angular position (θ) of the fin:

$$\theta = \theta_0 \cos(2\pi f t) + \bar{\theta}$$

Where the bias $\bar{\theta}$ denotes the mean value of θ, representing the contribution that facilitates the robotic fish in steering or yawing.

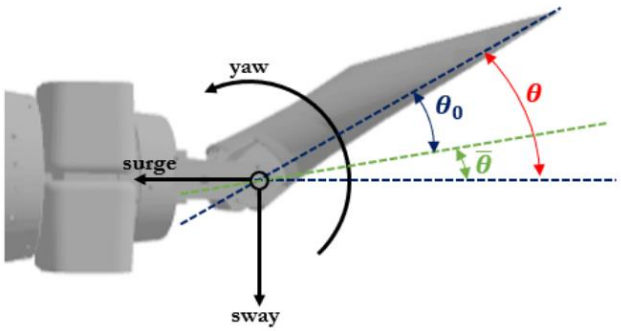


Figure 10 - Propulsive forces and torque decomposition

Taking into account the impact of the steering angle, the previously described forces in [Sub-Charter 2.a.1] can be expressed in the body-fixed frame through the following transformation:

$$\tau_{surge} = F_{FIN} \cos(\bar{\theta}) - L_{FIN} \sin(\bar{\theta})$$

$$\tau_{sway} = F_{FIN} \sin(\bar{\theta}) - L_{FIN} \sin(\bar{\theta})$$

$$\tau_{sway} = M_{FIN} - \frac{1}{4}\rho U^2 SL[(a_0 + a_1 St + a_2 St^2)] \sin(\bar{\theta})$$

where S is the fin planform area, L the fin length and *St* the Strouhal number.

Where a_0, a_1, a_2 $[Kg\ m^2]$ are caudal fin parameters achievable through multibody simulation, by using MSC Adams View. For the purpose of this project, these parameters are set to 0.

In compact form the following is obtained:

$$\tau = \begin{bmatrix} \tau_{surge} \\ \tau_{sway} \\ 0 \\ 0 \\ 0 \\ \tau_{yaw} \end{bmatrix}$$

2.a.3 Line of Sight (LOS) Guidance

The guidance system holds a crucial position in giving autonomy upon a system. Over the years, numerous guidance laws using different design concepts have been employed. Notably, missile guidance technology stands out as a mature field, boasting an abundance of implemented guidance laws in real systems. In the area of high-speed missiles, the prevailing technique is the widely adopted line-of-sight (LOS) guidance. This guidance method will be utilized for the investigation and implementation of the system for the examined robotic fish.

In a broad sense, line-of-sight (LOS) guidance is typically categorized as a three-point guidance system, involving a stationary reference point, an interceptor, and the target. The interceptor aims to achieve interception by maneuvering along the LOS vector formed between the reference point and the target.

In the realm of autonomous underwater vehicles, a widely adopted scheme is the waypoint law by line of sight. This form of guidance entails providing a heading command to the vehicle's steering system, directing it to approach the line of sight between its current position and the designated waypoint.

This guidance strategy proves effective for path following, particularly when the guidance involves progressing through multiple waypoints in a specific sequence. In such cases, a LOS vector is defined as the alignment between the current position of the vehicle and the intended waypoint.

For motions confined to a 2D horizontal plane, a straightforward approach involves considering a straight-line path defined by two waypoints $p_k = [x_k, y_k]^T$, $p_{k+1} = [x_{k+1}, y_{k+1}] \in \mathbb{R}^2$.

In addition, when considering a path-fixed reference frame, its origin is rotated by this angle:

$$a_k = atan2(y_{k+1} - y_k, x_{k+1} - x_k)$$

(atan2 is a function that yields an angle value within the range of $[-\pi, \pi]$).

This rotational adjustment is crucial for aligning the reference frame with the specific path or trajectory being followed. By introducing this angle rotation, the guidance system ensures that the vehicle's movements and positional references are accurately interpreted within the context of the designated path, enhancing precision and effectiveness in navigation.

With respect to the x_E axis, the vehicle's coordinates in this newly defined reference frame are determined as follows:

$$\begin{bmatrix} s(t) \\ e(t) \end{bmatrix} = \begin{bmatrix} \cos(\alpha_k) & -\sin(\alpha_k) \\ \sin(\alpha_k) & \cos(\alpha_k) \end{bmatrix} \begin{bmatrix} x_{k+1} - x_k \\ y_{k+1} - y_k \end{bmatrix}$$

where $s(t)$ represents the along-track distance (tangential to the path), and $e(t)$ denotes the cross-track error (normal to the path). Specifically for path-following purposes, the focus is on $e(t)$, as its reduction to zero signifies the vehicle's convergence to the straight path. To ensure that $e(t)$ approaches zero, both the course angle and heading can be strategically employed, contributing to the effective guidance and control of the vehicle along the desired trajectory.

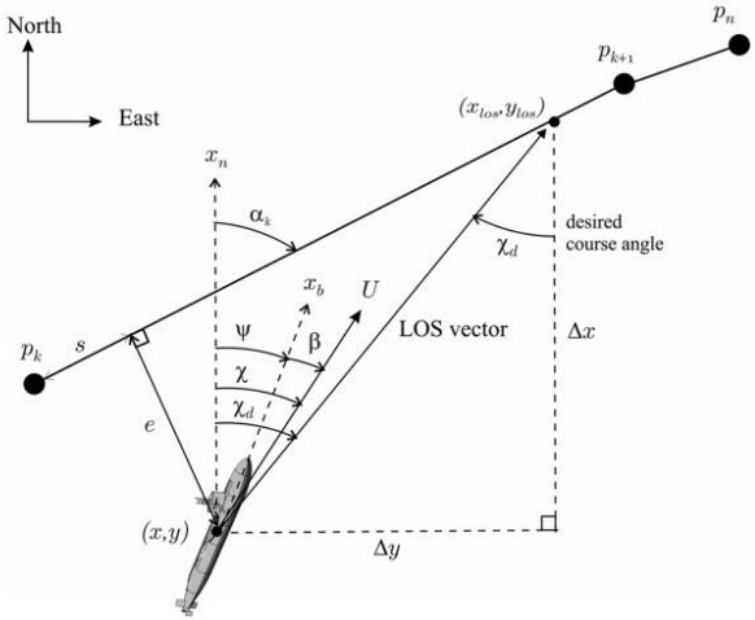


Figure 11 - General scheme for LOS Guidance

2.a.3.1 MAXFISH Guidance Law

In this section, it will be outlined a straightforward application of the robotic fish employing a combined Line-of-Sight (LOS) and waypoint guidance. Waypoint guidance is a widely used and effective strategy across various applications.

The k-th waypoint's position in 3D space is denoted as $[x_k, y_k, z_k]$. To navigate towards the waypoint, it has been introduced a new guidance law designed to address the challenge of separating the horizontal plane (x-y plane) from the depth.

- x-y plane: In this scenario, the LOS guidance strategy for a two-dimensional space is applied. The objective is to reach the x and y coordinates of the k-th waypoint, defined as $p_k = [x_k, y_k]^T$. To achieve this, the guidance law calculates the distance D between the k-th waypoint position $[x_k, y_k]$ and the vehicle's position at time t $[x(t), y(t)]$:

$$D = \sqrt{\left(x_k - x(t)\right)^2 + \left(y_k - y(t)\right)^2}$$

The computed distance is then utilized as input for the surge force controller.
To reach the k-th waypoint, it becomes necessary to determine the steering reference. The steering law, in this context, will adopt the structure of an enclosure-based technique:

$$\psi_d(t) = atan2(y_k - y(t), x_k - x(t))$$

(atan2 is a function that yields an angle value within the range of $[-\pi, \pi]$).

Where $\psi_d(t)$ represents the desired heading at time t, $[x_k, y_k]$ is the position of the k-th waypoint in the 2D space, and $[x(t), y(t)]$ is the position of the fish at time t. The calculated $\psi_d(t)$ is then supplied to the steering force controller.

- Depth: In this case, the guidance system refrains from actively manipulating the depth information. Instead, it directly supplies the z coordinate of the k-th waypoint (z_k) to the heave controller.

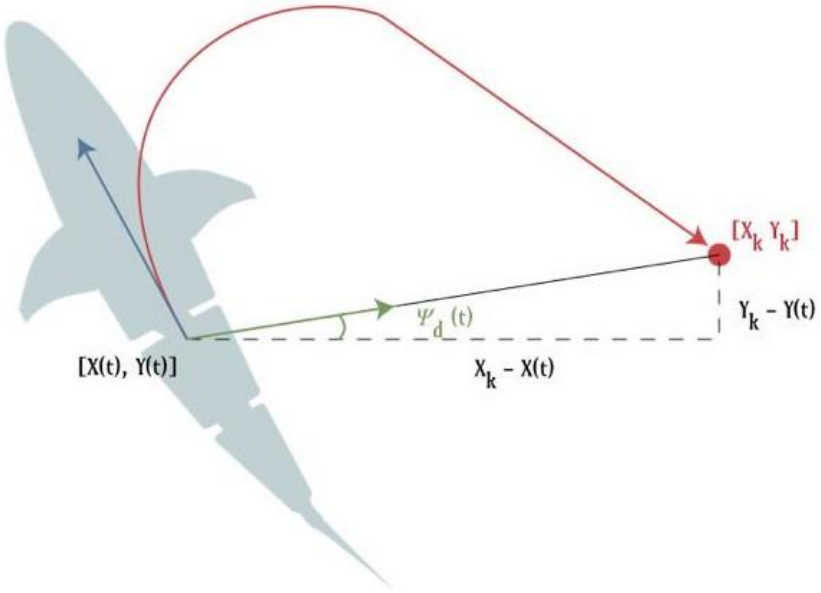


Figure 12 - LOS Guidance scheme of MAXFISH on x-y plane

2.a.4 Dynamic Trajectory Evaluation using Velocity-Weighted Sampling Approach

Trajectory tracking is a critical aspect of control system performance, particularly in applications such as robotics and autonomous vehicles. The conventional approach to trajectory analysis typically involves computing integral-based metrics that emphasize cumulative position errors over the entire trajectory. However, this method may lack sensitivity to variations in velocity, which can significantly impact the system's overall performance.

The trajectory of a dynamic system is inherently characterized by its position $y(t)$ and instantaneous velocity $y'(t)$. The significance of velocity becomes evident when considering the varying speeds at which different segments of a trajectory are traversed. Errors in high-velocity segments can disproportionately affect overall trajectory tracking performance. Recognizing this, the proposed approach introduces a dynamic sampling scheme based on local velocities.

The velocity-weighted sampling method involves dividing the trajectory into *n* distinct samples, each with a duration Δt_i proportional to the local average velocity v_i.

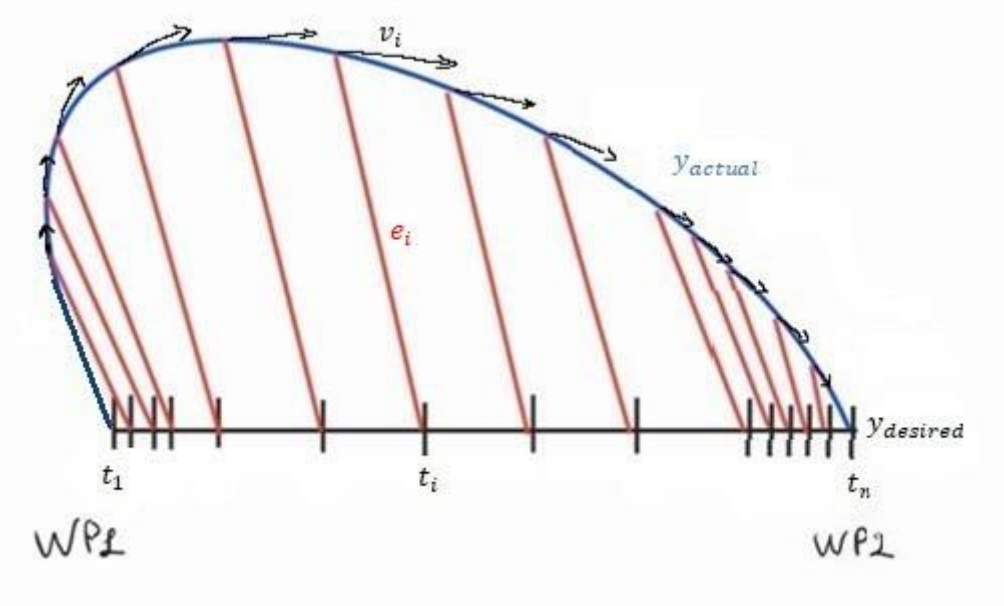


Figure 13 - Actual trajectory (blue) with respect to the desired trajectory (black) between two waypoints WP1 and WP2.

This dynamic segmentation ensures that samples with higher velocities contribute more significantly to the overall error computation. The trajectory tracking error within each sample is then computed using integral-based metrics, considering both position and velocity discrepancies.

For each sample, the trajectory tracking error is calculated as the absolute difference between the actual and desired positions over the sample duration:

$$e_i = |y_{actual}(t_i) - y_{desired}(t_i)| \quad \forall\, i = 1, \dots, n$$

Where:

- t_i is the time of the sample i;
- $y_{actual}(t)$ and $y_{desired}(t)$ represent the actual and desired positions at time t, respectively.

The dynamic sample lengths implicitly introduce a weighting factor, emphasizing the importance of accurate tracking in high-velocity segments.

To further characterize the error distribution, we computed the mean and standard deviation of the trajectory tracking errors. The mean error $\bar{e}$ provides a measure of the average magnitude of errors across all samples:

$$\bar{e} = \frac{1}{n}\sum_{i=1}^{n} e_i$$

The standard deviation σ quantifies the spread or variability of the trajectory tracking errors e_i across all samples, thus it indicates how far the actual trajectory deviates from the desired trajectory:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$$

The velocity-weighted sampling approach offers several advantages over traditional integral-based methods. It enables a more granular evaluation of trajectory tracking performance, accounting for the dynamic nature of systems with varying velocities. This approach is particularly relevant in applications where high-speed maneuvers and precise trajectory tracking are critical.

Furthermore, the technique provides valuable insights into how well a control system adapts to changes in velocity, a crucial aspect of dynamic systems' performance. The analysis afforded by this approach contributes to a deeper understanding of control system behavior under varying trajectory dynamics.

In conclusion, the velocity-weighted sampling approach provides a sophisticated way to evaluate trajectory tracking performance in dynamic systems. By incorporating velocity information into the segmentation of trajectory samples, this method offers a more accurate representation of a system's ability to track trajectories with varying dynamics. As technological advancements demand increasingly precise control, the proposed technique becomes a valuable tool in ensuring the reliability and efficiency of control systems in dynamic environments.

2.b Mechatronic infrastructure and the operational environment

The project was conceived and carried out using only numerical computation, modeling and simulation environments, such as **Matlab and Simulink**. This methodological choice was driven by the desire to reduce development time, save resources, and test the product under varying conditions without the need for physical prototypes. Simulations were performed using the **3D World Editor** (native VRML editor), which is included in the Simulink 3D Animation product. These simulations allowed an accurate representation of the system's behavior in water. This made it possible to identify potential issues, optimize performance, and evaluate the robustness of the system in a detailed and controlled manner.

During the testing and simulation phases, a system with certain technical specifications was used. The system employed Simulink's **Ode4 solver** with the **Runge Kutta method** to handle the model dynamics precisely.

The **simulation time step was set to 0.00025 seconds** to ensure good accuracy in the analysis. The computer used has **16 GB of RAM** and a **256 GB SSD** drive, providing sufficient capacity for data processing and storage.

The **11th Generation Intel i7** processor ensured adequate performance for computing data and simulations. The presence of an integrated **NVIDIA GeForce MX450** GPU supported the graphical and parallel calculations required for model processing.

These technical specifications contributed to the robustness and accuracy of the simulations, enabling detailed analysis of model dynamics.

2.c User Interface and Digital Twin model

2.c.1 Simulink Model

The mathematical model of MAXFISH was implemented in Simulink, a software integrated with MATLAB specifically designed for modeling, simulation, and analysis of dynamic systems. The adoption of this software is motivated by several reasons.

Firstly, it is widely used in academia; secondly, its graphical interface simplifies the complexity of the simulator, offering an intuitive approach to programming. Finally, it offers the ability to automatically translate models into languages suitable for real-time applications.

Starting with the Simulink model of the GUIZZO 6.0 fish, substantial modifications to the blocks were undertaken to adapt them to the mathematical model of MAXFISH. This process involved revising and adapting the existing blocks to accurately reflect the equations and features to the mathematical model of MAXFISH {Fig. 14}.

In particular, the individual blocks of the Simulink model were customised and redefined, ensuring that each of them accurately reflected the predicted behaviour and variables considered in the mathematical model of MAXFISH. This adaptation allowed the transition from the initial model to the new MAXFISH robotic fish context, ensuring higher accuracy in simulation and analysis processes.

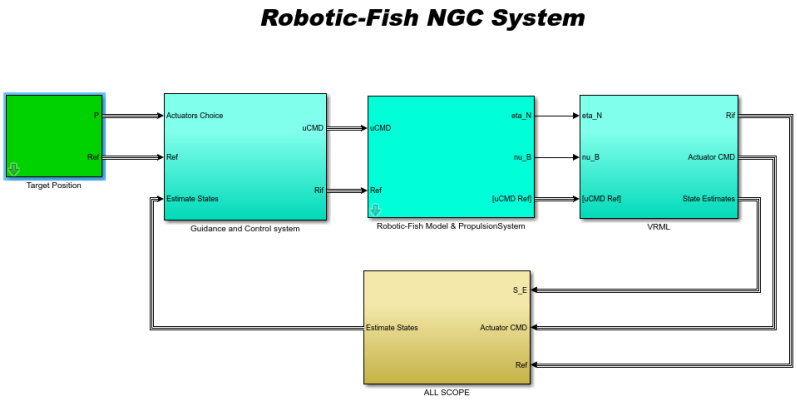


Figure 14 - Simulink Model of MAXFISH

The following paragraphs will give an overview of each implemented subsystem.

2.c.1.1 Target Position Subsystem

This subsystem {Fig. 15} allows to program the desired path for the fish to run, double-clicking on the block will open a window {Fig. 16} that allows to choose:

- **Actuator choice**:
  - *Normal*: Which involves the use of the caudal fin for surge motion, the lateral propellers for steering. The pectoral fins and vertical propeller are used for depth control.
  - *Only Propellers*: Where the propellers are used for both surge motion and steering. The pectoral fins and vertical propeller are used for depth control.
  - *Only Caudal Fin*: Operation which involves only the use of the caudal fin for surge motion and steering. Depth control is not allowed.

- **Select Target**:
  - *Way-Points based Path*: Allows to choose Path consisting of multiple waypoints that can be generated from the "Signal Builder" block, which is a Simulink block that creates a group of signals.
  - *Way-Point Position*: Consists in choosing the (x, y, z) coordinates of the $\{n\}$ reference system, essentially it consists of a single set-point to reach.

- **Way-Point Position**: Allows to manually enter parameters to generate the desired way-Point; it makes sense only if "Way-Point Position" was chosen previously.

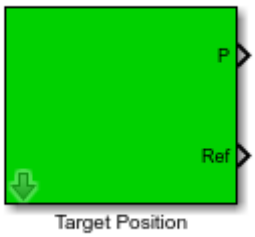


Figure 15 - Target Position Subsystem

As input to this subsystem there is the choice made by the user and as output there are: "Ref" which is the position (x_R, y_R, z_R) of the next waypoint and "P" which is the chosen actuator.

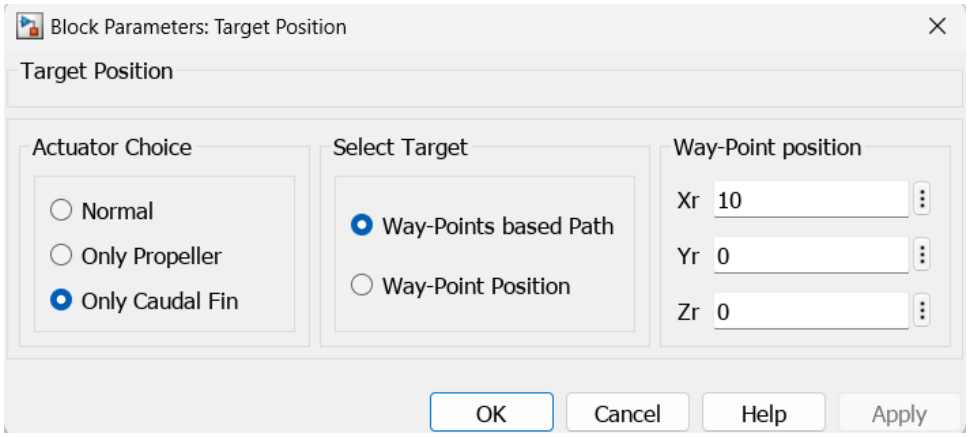


Figure 16 - Target Position parameters

In addition, the “Signal Builder” block was used to create the desired trajectory. Clicking on the block arrow expands the subsystem {Fig. 17} in which the block is located. Double-clicking on the block will open a window in which, by choosing "Launch signal editor" {Fig. 18} the desired Way-Points can be set.

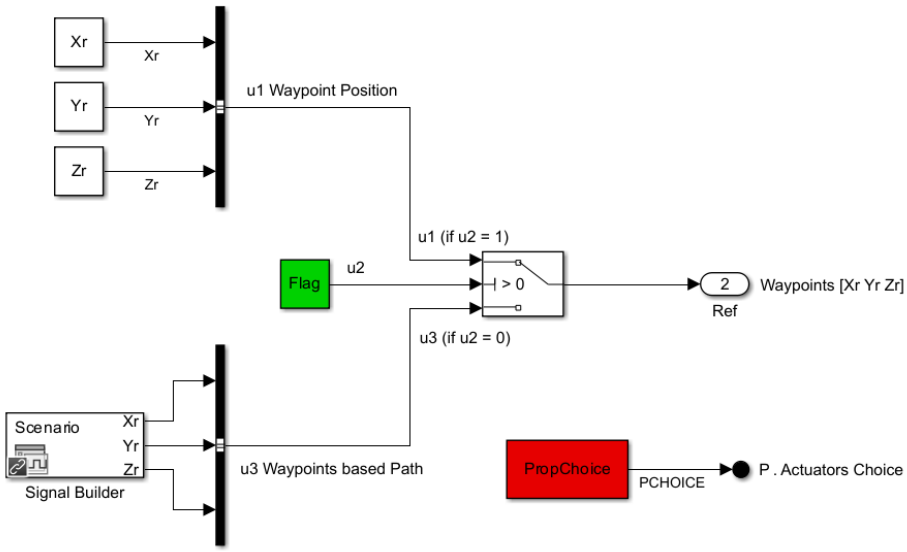


Figure 17 – Target Position Subsystem expanded

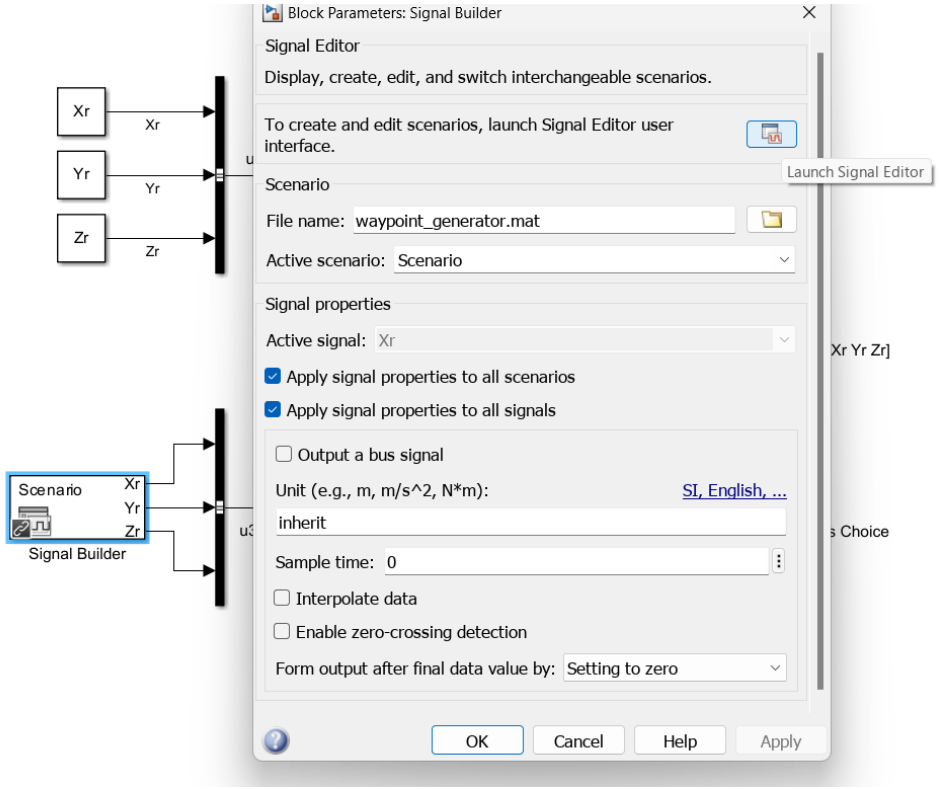


Figure 18 – "Signal Builder" window

In our case, the desired trajectory consists of a path of 7 waypoints. By checking the scopes, when the fish reaches a waypoint in a neighborhood of radius ε = 0.3 m, the next waypoint is generated.

Starting from the Start Point position $(0,0,0)$ on the {n} reference frame at the time t = 0 s, the 7 waypoints to be reached in sequence are:

- WP0 $(0,0,0)$ at the time t = 0 s
- WP1 $(10,0,0)$ at the time t = 0 s
- WP2 $(15,5,0)$ at the time t = 10.33 s
- WP3 $(20,0,0)$ at the time t = 20.08 s
- WP4 $(20,-10,0)$ at the time t = 34.2 s
- WP5 $(10,-10,0)$ at the time t = 44.37 s
- WP6 $(10,0,0)$ at the time t = 59.37 s
- WP7 $(0,0,0)$ at the time t = 73.09 s

As can also be seen from the figure below {Fig. 19}, the signal along the z-axis is kept constant at zero because in our control the robotic fish can only use its tail for surge motion and steering, so it cannot move in depth without the use of pectoral fins or vertical propeller.

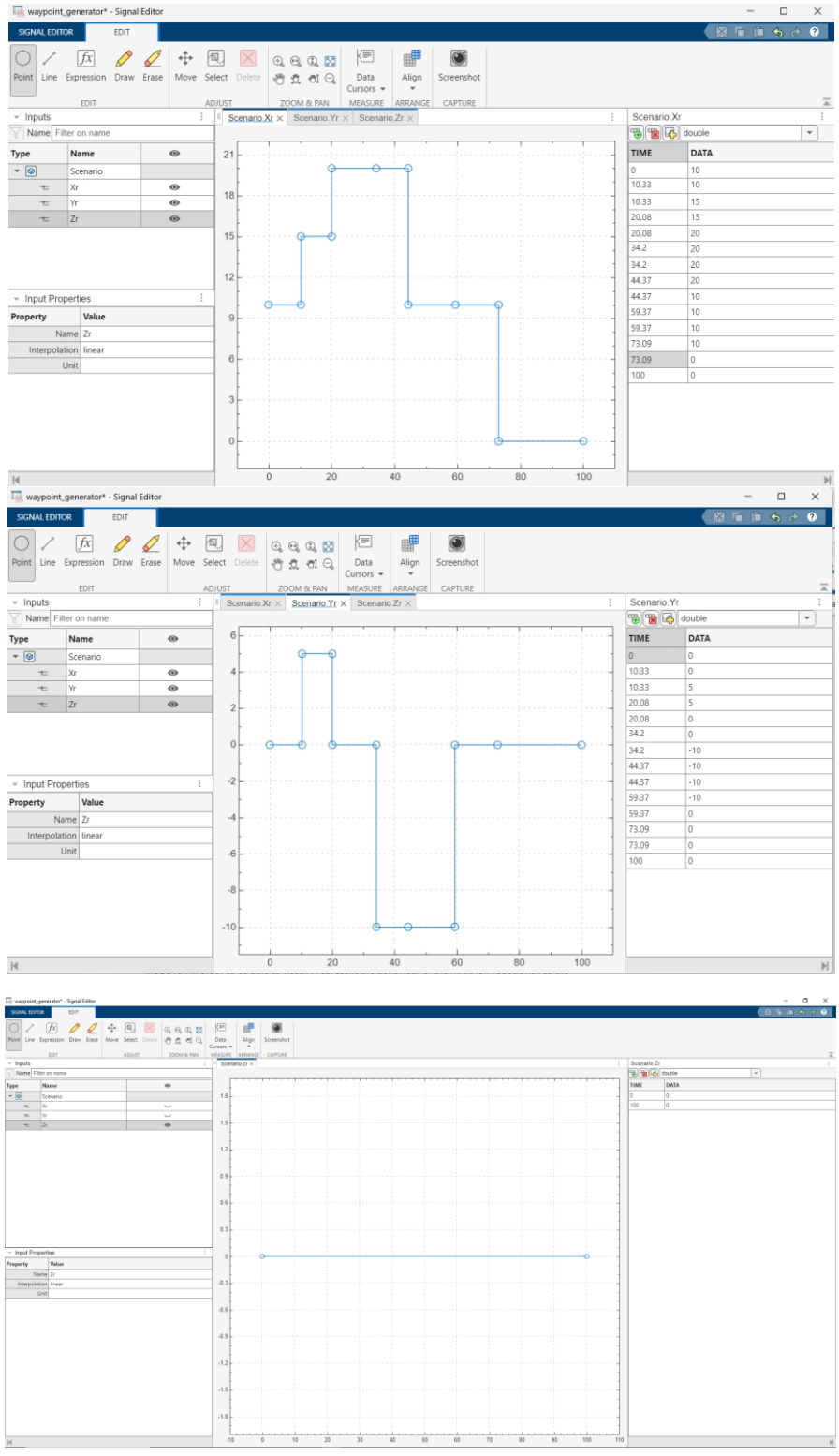


Figure 19 – Way-Points generated along the axes (x,y,z) of the {n} reference frame

2.c.1.2 Guidance and Control System

Within the Guidance and Control System, there are two other subsystems {Fig. 20}:

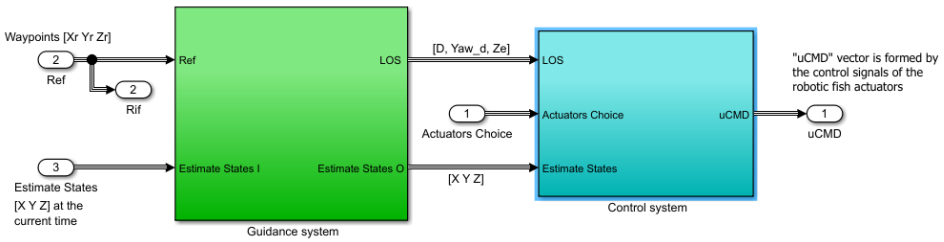


Figure 20 - Guidance and Control System expanded

❖ **Guidance system**:

In this subsystem {Fig. 21}, the LOS guidance law was implemented, previously described in section 2.a.3. This module takes as input the vector "Ref", consisting of (X_R, Y_R, Z_R), and the second input is the vector "Estimate States", which encapsulates the current position, orientation, linear and angular velocities of the robot. Only the (X, Y, Z) coordinates of the vehicle's current position are selected.

In output, the module returns the vector "LOS," consisting of the distance D calculated by the "Distance" block, the yaw angle, called Yaw_d, calculated via the "LOS law" block, and the difference between Zr and Z, that is Ze. To calculate the Yaw_d angle, the Unwrap Simulink block is also employed, which corrects the input by adding or subtracting appropriate multiples of π to remove phase discontinuities.

In addition, the module returns a second output represented by the vector "Estimate States".

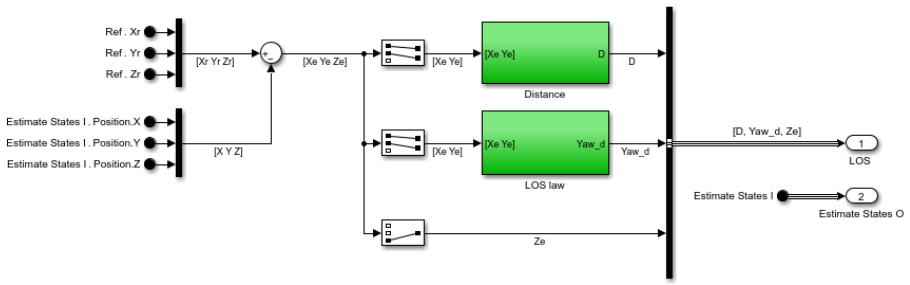


Figure 21 – Guidance System

❖ **Control system**:

The second subsystem takes as input the vectors returned by the "Guidance System", "LOS" and "Estimate States", and also takes as input the variable "Actuators Choice".

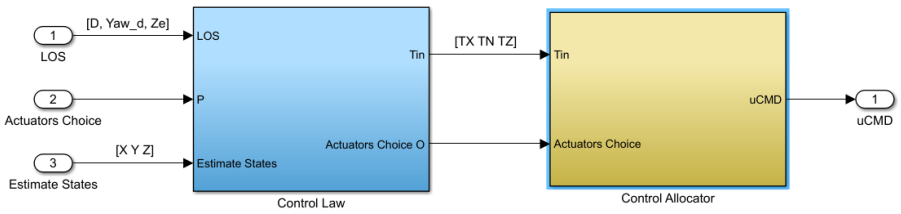


Figure 22 – Control System

In addition, the "Control System" is, in turn, composed of two other sub-systems, "Control Law" and "Control Allocator".

Control Law subsystem

The "Control Law" block {Fig. 25} contains the PID controllers.

The "Actuator Choice" variable assumes value:

- 0 If “Normal” operation is selected.
- 1 if "Only Propeller" operation is selected.
- 2 if "Only Caudal Fin" operation is selected.

This variable is used to select between the 3 PIDs of surge force TX, steering moment TN and depth force TZ.

The output signal from the PID on TX is saturated by the “Saturation TX” block between 0 and 4 Hz {Fig. 23}. This is necessary to prevent damage to the motor and to control its speed.

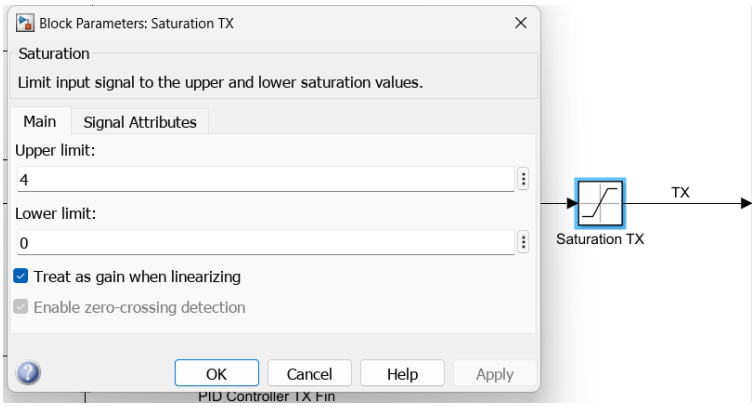


Figure 23 – “Saturation TX” Block

Similarly, the PID output signal on TN is saturated by the “Saturation TN” block between -1.3090 and 1.3090 radians {Fig. 24}, this signal acts on the tail bias $\bar{\theta}$ for steering. This boundary is due to the mechanical limitation of the tail, which cannot physically rotate more than 75° with respect to the x-axis of the reference system {b} attached to the fish.

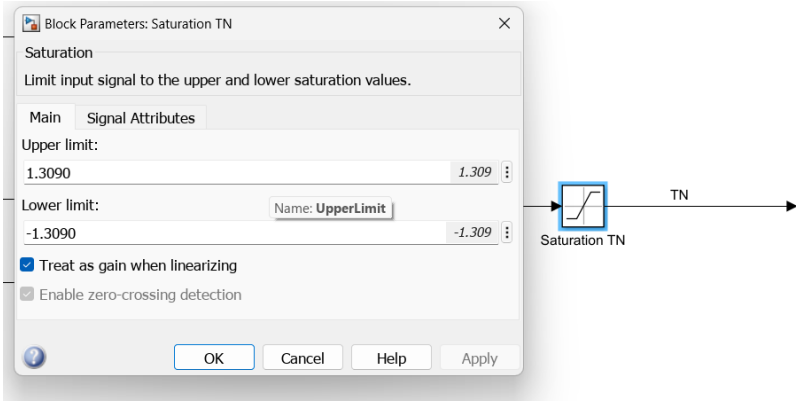


Figure 24 - "Saturation TN" Block

As output, the "Control Law" subsystem returns the vector "Tin" formed by surge force TX, steering moment TN and depth force TZ.

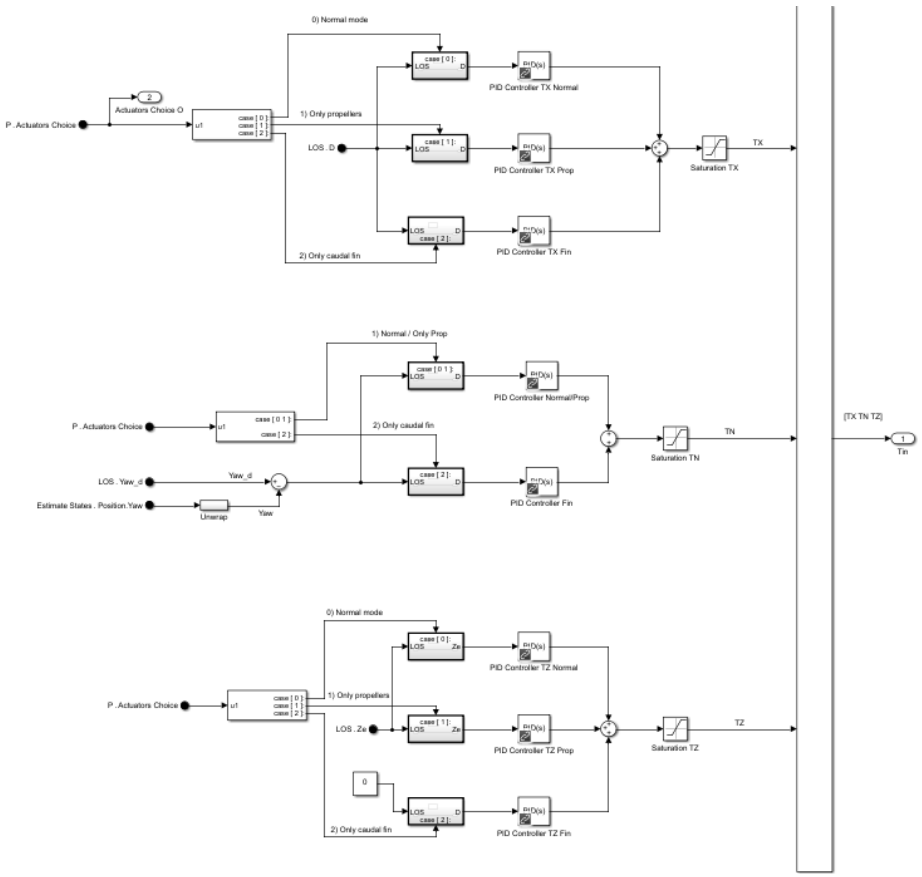


Figure 25 – Control Law subsystem

When the choice of actuator is "Only caudal fin", the fish uses only its tail to move forward and steer. As designers, we chose to use a proportional control on τ_X and a PD control on τ_N, as the derivative action responds very well to changes in direction. Care must be taken that increasing the gain K_d derivative too many risks making the control too sensitive to

abrupt changes in θ. Moreover, increasing the gain means that the fish take longer to reach its destination.

For the control on τ_X, it was decided not to use integral action because this leads to frequency saturation in a short time, thus presenting the phenomenon of wind-up {Fig. 26}.

The controller parameters were first tuned using the Ziegler-Nichols method and then adjusted empirically.

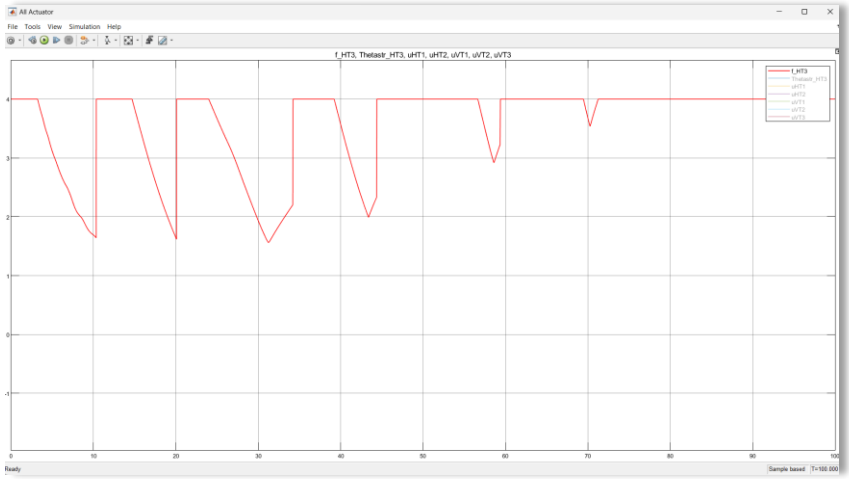


Figure 26 – Phenomenon of wind-up on frequency actuator TX

For further information on other choices of actuators, please refer to the thesis “STUDY AND SIMULATION OF NGC SYSTEMS FOR BIOMIMETIC ROBOTS” by Engineer Waqas Ali Waqar [5].

Control Allocator subsystem

The "Control Allocator" block {Fig. 27} takes the vector "Tin" and the variable "Actuator Choice" as input, generating as output the vector "uCMD" which contains the control signals of the robotic fish actuators.

These commands are seven in total:

1. f_uHT3 controls caudal fin (HT3) oscillation frequency.
2. Theta_uHT3 controls caudal fin (HT3) steering angle.
3. uHT1 controls lateral propeller (HT1) rotation speed.
4. uHT2 controls lateral propeller (HT2) rotation speed.
5. uVT1 controls vertical propeller (VT1) rotation speed.
6. uVT2 controls pectoral fin (VT2) angle referring to the vehicle’s hull.
7. uVT3 controls pectoral fin (VT3) angle referring to the vehicle’s hull.

In this module, the control allocation strategy is implemented.

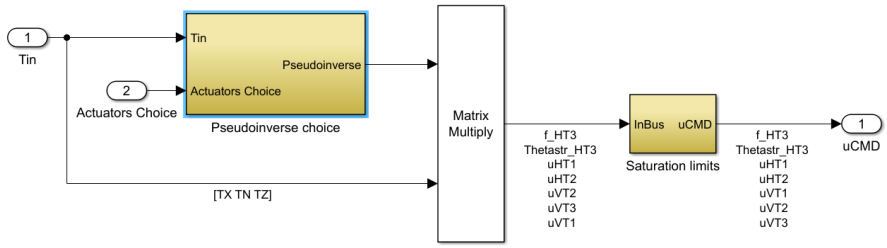


Figure 27 – Control Allocator subsystem

In the "Pseudoinverse choice" block {Fig. 28}, several force coefficient matrices T^+ are defined. These matrices are selected according to actuator choice and command power.

When this choice is “Only caudal fin”, it is always selected the pseudoinverse Matrix 5.

$$T_5^+ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Therefore, the solution to the control allocation problem is simply to solve the system of equations:

$$\begin{cases} u_{f\_HT3} = \tau_X \\ u_{\theta\_HT3} = \tau_N \\ u_{HT1} = 0 \\ u_{HT2} = 0 \\ u_{VT1} = 0 \\ u_{VT2} = 0 \\ u_{VT3} = 0 \end{cases} \quad with\ saturation\ limit\ u_{min} \leq u_i \leq u_{max}$$

In fact, the frequency control signal for surge force τ_X and the bias $\bar{\theta}$ control signal for steering moment τ_N are used.

The "Saturation Limits" block allows to set the saturation limits for each actuator control signal.

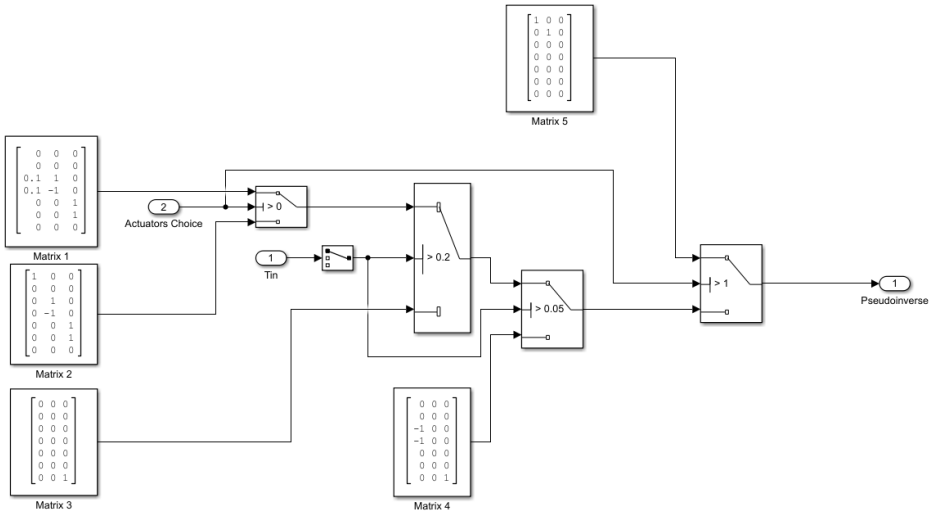


Figure 28 – "Pseudoinverse choice" block

For further information on all allocation matrices, please refer to the thesis "STUDY AND SIMULATION OF NGC SYSTEMS FOR BIOMIMETIC ROBOTS" by Engineer Waqas Ali Waqar [5].

2.c.1.3 Robotic-Fish Model & Propulsion System

Within the Robotic-Fish Model & Propulsion System {Fig. 29}, there are two linked blocks:

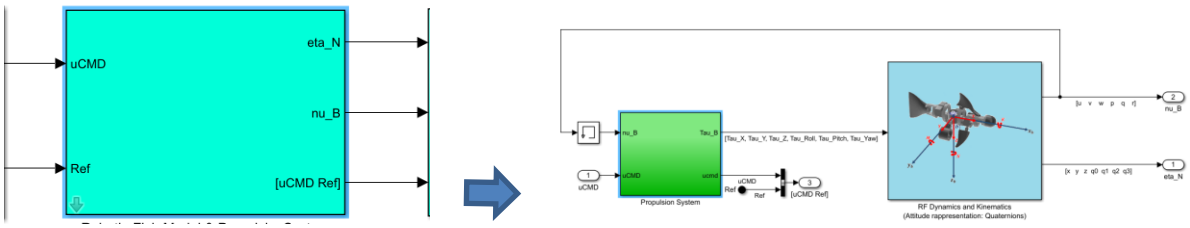


Figure 29 - Robotic-Fish Model & Propulsion System

- **Propulsion System** {Fig. 30}: The block takes "nu_B" and "uCMD" as input and produces "Tau_B" and "uCMD" as output.
  The vector "nu_B" is $[u\ v\ w\ p\ q\ r]$, where the first 3 components are the linear velocities and the remaining 3 are the angular velocities of the simulated fish at time $t-1$.
  The vector "Tau_B" is $[\mathrm{Tau\_X}, \mathrm{Tau\_Y}, \mathrm{Tau\_Z}, \mathrm{Tau\_Roll}, \mathrm{Tau\_Pitch}, \mathrm{Tau\_Yaw}]$, where the first 3 components are the external forces and the remaining 3 are the external moments exerted by the actuators at time t.
  This part was discussed earlier in section 2.a.2.3 Propulsion system.

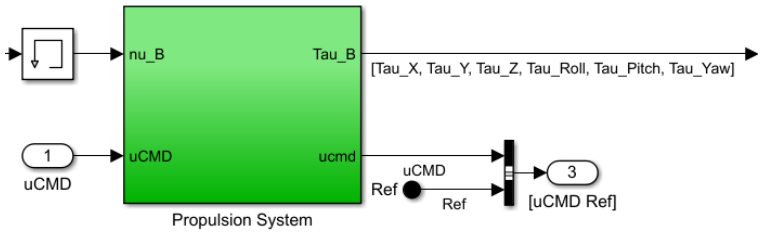


Figure 30 – Propulsion System

Inside the Propulsion System, there is another sub-diagram shown in Figure 31 that has two blocks:

- Horizontal Force and Moment (Tau_H)
- Vertical Force and Moment (Tau_V)

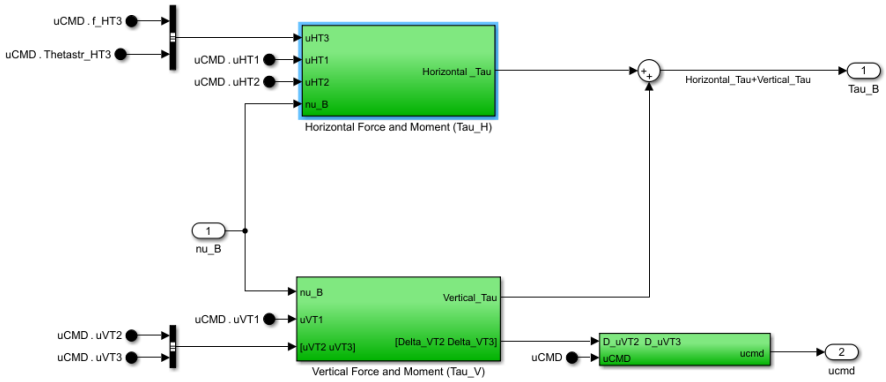


Figure 31 – Propulsion System expanded

Within the first block, there is another sub-diagram shown in Figure 32, in which all forces and moments generated by the horizontal thrusters, i.e. HT3 caudal fin, HT1 and HT2 lateral propellers, are calculated.

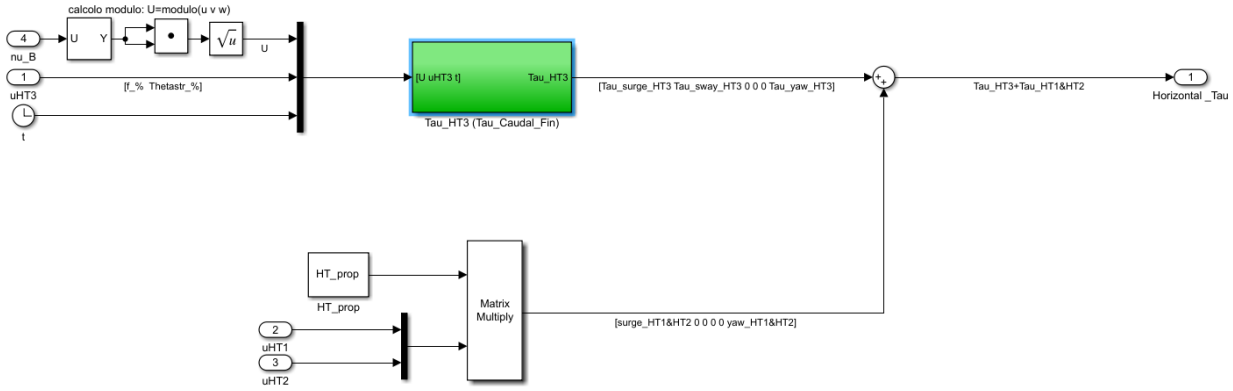


Figure 32 – "Horizontal Force and Moment (Tau_H)" block

We finally arrive at the block of interest “Tau_HT3 (Tau_Caudal_Fin)” {Fig. 33}, where the forces and moments generated by MAXFISH's tail thrust have been implemented.

Since the fish has not yet been physically realised, thanks to the Engineer Daniele Costa, we calculated these forces and moments numerically with the help of multibody simulation software, MSC Adams View.

The Surge Force:

$$T_{FIN} = K_{T1} f^2 [K_{T0} + K_{T2} \sin(4\pi f t + \beta)]$$

The Sway Force:

$$L_{FIN} = K_{L1} f^2 K_{L2} \cos(2\pi f t + \sigma)$$

The Yawing Torque:

$$M_{FIN} = K_{M1} f^2 K_{M2} \cos(2\pi f t + \varepsilon)$$

The numerical parameters can be found in the summary table in the user manual.

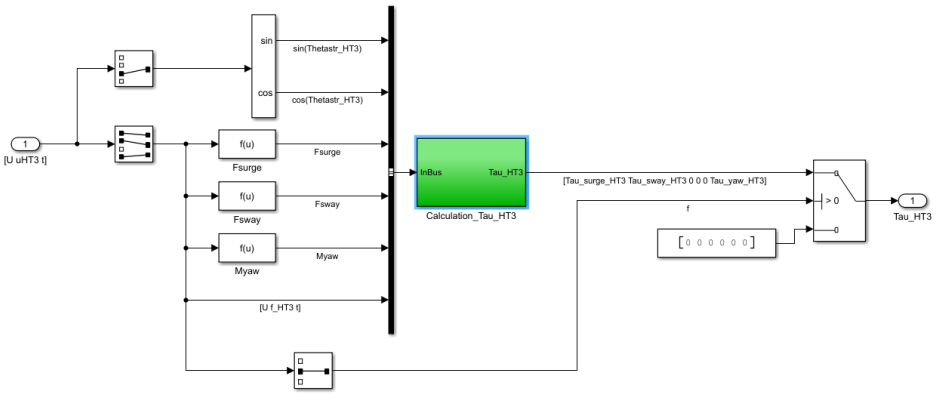


Figure 33 – “Tau_HT3 (Tau_Caudal_Fin)” block

Finally, within the “Calculation_Tau_HT3” block, there is the sub-diagram shown in Figure 34, where the thrust τ_{HT3} generated by the tail is calculated, taking into account the variation of the tail bias $\bar{\theta}$.

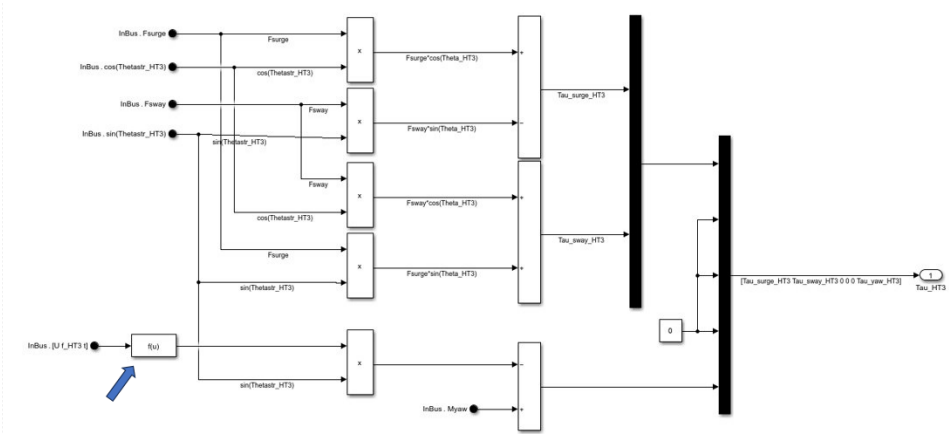


Figure 34 - “Calculation_Tau_HT3” block

It is important to note that the parameters a_0, a_1, a_2, S are currently not available, as they require the implementation of the robotic fish, and are therefore temporarily set to 0. However, the steering angle contribution on τ_{yawHT3} has already been implemented as follows and can be found in the block shown in Figure 35.

$$\tau_{yawHT3} = M_{yaw} - \frac{1}{4}\rho U^2 SL[(a_0 + a_1 St + a_2 St^2)]\sin(\bar{\theta})$$

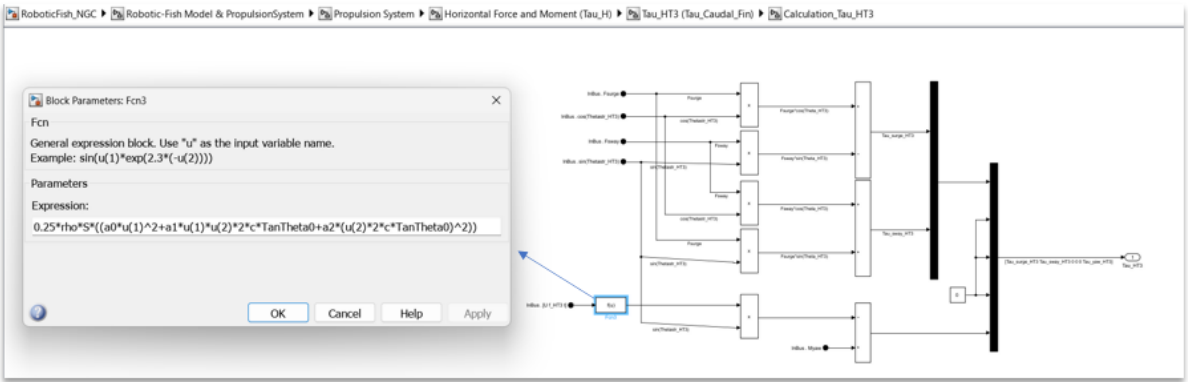


Figure 35 – Steering angle contribution on tau_yawHT3 implementation

To investigate the forces and moments generated by lateral and vertical propellers, please refer to the thesis "STUDY AND SIMULATION OF NGC SYSTEMS FOR BIOMIMETIC ROBOTS" by Engineer Waqas Ali Waqar [5].

- **RF Dynamics and Kinematics** {Fig. 36}: This subsystem takes "Tau_B" as input and produces "nu_B" and "eta_N" as output.
  The vector "Tau_B" is [Tau_X, Tau_Y, Tau_Z, Tau_Roll, Tau_Pitch, Tau_Yaw], where the first 3 components are the external forces and the remaining 3 are the external moments exerted by the actuators at time t. This vector is provided as feedback to the previous block.
  The vector "nu_B" is $[u\ v\ w\ p\ q\ r]$, where the first 3 components are the linear velocities and the remaining 3 are the angular velocities of the simulated fish at time t.
  The vector "eta_N" is $[x\ y\ z\ q0\ q1\ q2\ q3]$, where the first 3 components are the position and the remaining 4 are the orientation, expressed as quaternions, of the simulated fish at time t.

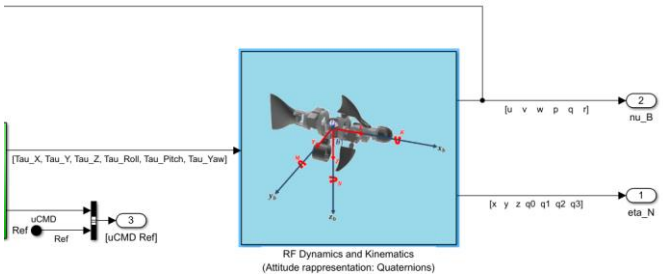


Figure 36 – "RF Dynamics and Kinematics" subsystem

Within this block, there is the sub-diagram shown in Figure 37.

This part was discussed earlier in Chapter 2.a.2 “Fossen mathematical model of MAXFISH”.

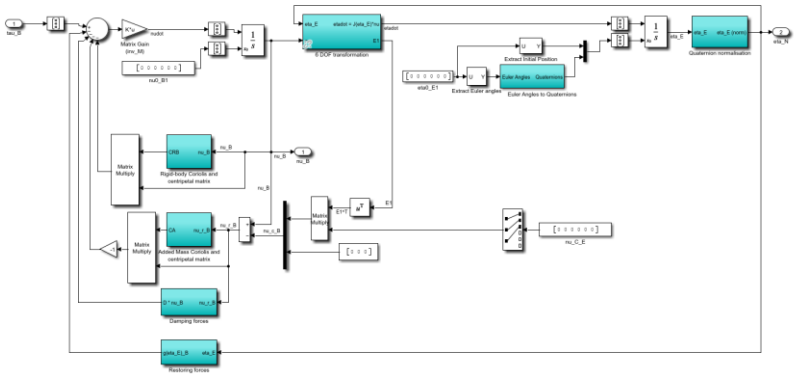


Figure 37 - “RF Dynamics and Kinematics” subsystem expanded

2.c.1.4 VRML Block

The VRML block {Fig. 38} (Virtual Reality Modelling Language) supports the simulation of GUIZZO 6.0 model in a virtual underwater environment, using the 3D World Editor (native VRML editor), which is included in the Simulink 3D Animation product.

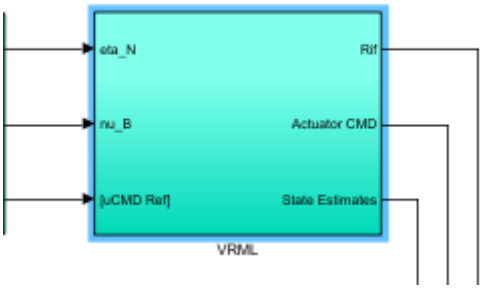


Figure 38 – VRML Block

Within this block, there is a sub-diagram shown in Figure 39. Here, the "RF Virtual Reality Display" block provides the GUI interface. GUIZZO 6.0 simulator generates signals data of the vehicle’s dynamics and kinematics (vectors $\eta(t)$ and $v(t)$). By connecting the Simulink model to a virtual world, this data can be used to control and animate the virtual underwater world.

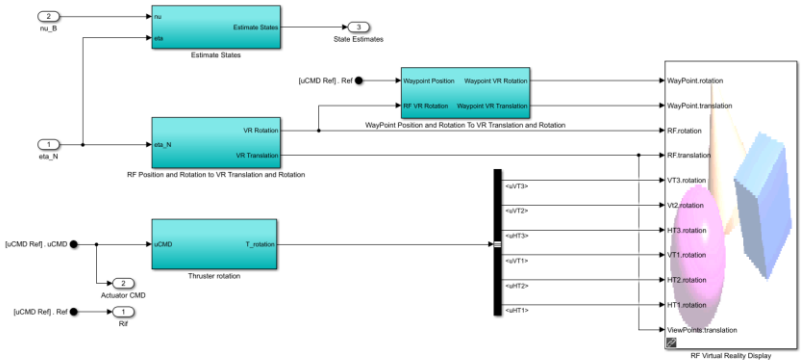


Figure 39 – VRML Block expanded

For more details on the 3D simulation aspect, please refer to chapter 5 “GUIZZO 6.0 Virtual Underwater World” of the thesis “STUDY AND SIMULATION OF NGC SYSTEMS FOR BIOMIMETIC ROBOTS” by Engineer Waqas Ali Waqar [5].

2.c.1.5 ALL SCOPE Block

This subsystem {Fig. 40} contains all the graphs relating to the position, orientation, linear velocities and angular velocities of the robotic fish. Also, it contains the graphics of the control signals of the actuators with which the vehicle is equipped {Fig. 41}.

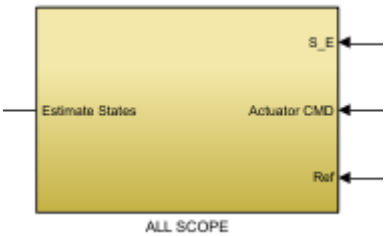


Figure 40 – ALL SCOPE Block

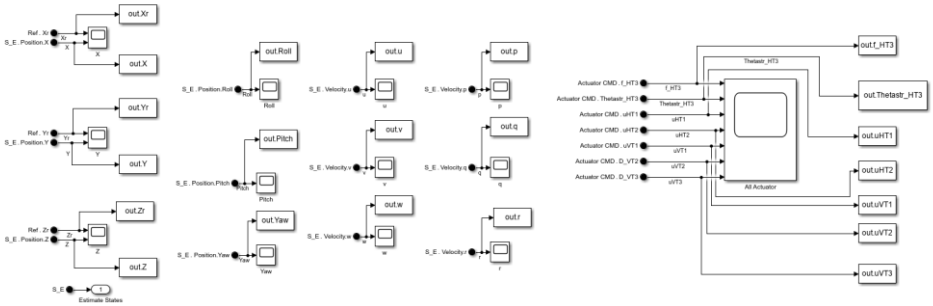


Figure 41 – ALL SCOPE Block expanded

2.c.2 User Interface and Digital Twin model

A key part of the SuperFin project was the implementation of a graphical interface with MATLAB's App Designer application and the integration of a Digital twin, previously developed by Engineer Waqas Ali Waqar in his thesis entitled "STUDY AND SIMULATION OF NGC SYSTEMS FOR BIOMIMETIC ROBOTS", in order to be able to visually represent the behaviour of the MAXFISH robotic fish as the simulation parameters change.

It is important to emphasise that this digital twin does not faithfully represent the AUV studied in this project, but rather represents the robotic fish GUIZZO 6.0. In fact, the current robotic tail in the Digital twin is not composed of three links, but of a single joint, i.e. the single caudal fin, as can be seen in [Figure 42].

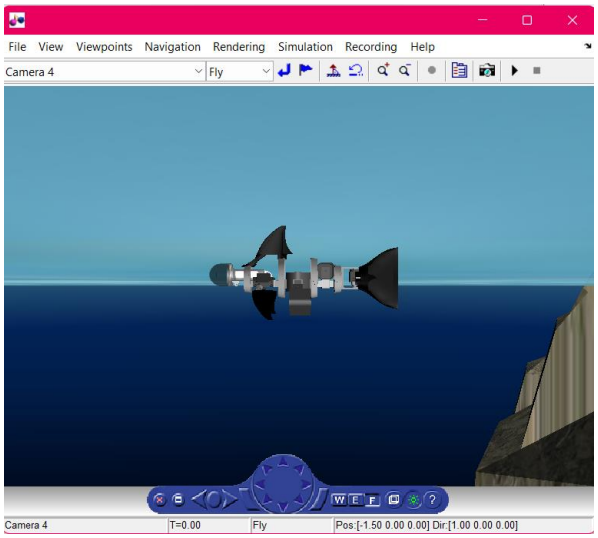


Figure 42 - Digital Twin of the robotic fish

Therefore, it is important to point out that the behaviour of the robotic fish during the simulation is faithful to what would be its real behaviour, except for the tail, as this is not actually the tail of the MAXFISH robotic fish. To compensate for this discrepancy, it was decided to simulate in the MATLAB environment what the behaviour of MAXFISH's three-jointed tail would be if it were represented in the simulated environment.

However, it was chosen to integrate this Digital twin as it is a useful tool in understanding and analysing the behaviour of the robotic fish and to be able to analyse its characteristics in a virtual environment, offering a detailed and realistic perspective of its interactions with its surroundings.

As far as the graphic interface is concerned, on the other hand, it was developed to simplify and significantly improve the end-users' experience in managing the parameters of the Simulink model "RoboticFish_NGC.slx", running the simulation of this model and analysing the results in an intuitive way.

2.c.2.1 App GUI

The main objective of this application is to provide a user-friendly environment to easily adjust Simulink model parameters, allowing users to evaluate the impact of changes made to the system. The graphical interface provides a comprehensive and detailed overview of the results obtained after running the simulation with these parameters.

The main features are as follows:

1. *Import the model parameters*: The app allows users to easily edit Simulink model parameters by importing them from a provided Excel file. Any changes are reflected in the app via the 'Load Parameters' button. In fact, after activating this button on the first page of the GUI, the parameters previously set by the user will be displayed on the screen via a table. It is important to emphasise that the names of the parameters that are changed by the user and displayed in the GUI are the names of the variables used in the model code, so diminutives are present. The table below shows the corresponding name of each parameter that can be changed:

Parameter's name	Excel parameter's name	Possible Choices	Initial Values
Robotic Fish Parameters			
Stop Time $[s]$	StopTime	Only positive numbers	89
Actuator Choice	AC	Normal	Only Caudal Fin
		Only Propeller	
		Only Caudal Fin	
Select Target	ST	Way-Points based Path	Way-Points based Path
		Way-point Position	
Way-Point position: Xr $[m]$	Xr	-	10
Way-Point position: Yr $[m]$	Yr	-	0
Way-Point position: Zr $[m]$	Zr	-	0
Inertia Tensor Ig $[Kg\ m^2]$	Ig	Matrix 3x3	[0.0198 0 0;

			0 0.9266 0; 0 0 0.9266]
Mass m $[Kg]$	m	Only positive numbers	11,3
Seawater Density rho $[Kg/m^3]$	rho	-	1000
Body Radius R $[m]$	R	Only positive numbers	0,06
Body Length L $[m]$	L	Only positive numbers	1
Drag Weight cdf	cdf	-	0,5
Drag Weight cdt	cdt	[0.8-1.2]	1
Weight $[N]$	W	-	107.91
Buoyancy $[N]$	B	-	107.91
rG_B $[m]$	rG_B	Matrix 3x1	[0;0;0]
rB_B $[m]$	rB_B	Matrix 3x1	[0;0;0]
Linear Damping Matrix	DL	Matrix 6x6	diag([0;0;0;0;0;0])
Caudal Fin Parameters			
Kt0	Kt0	-	1,16
Kt1	Kt1	-	0,5
Kt2	Kt2	-	1,26
Km1	Km1	-	0,5
Km2	Km2	-	0,073
Kl1	Kl1	-	0,5
Kl2	Kl2	-	5,16
beta	beta	-	-4,95
sigma	sigma	-	-3,52
epsilon	epsilon	-	-3,51
c-Foil Mean Chord $[m]$	c	-	0,1
S-Foil area $[m^2]$	S	-	0
a0 $[Kg\ m^2]$	a0	-	0

a1 $[Kg\ m^2]$	a1	-	0
a2 $[Kg\ m^2]$	a2	-	0
Theta0-Oscillation amplitude $[deg]$	Theta0deg	-	23
Pectoral Fins Parameters			
fin lift coefficient cL	cL	-	2
fin axial position xfin $[m]$	xfin	-	0,11
fin axial position yfin $[m]$	yfin	-	0,11
fin axial position zfin $[m]$	zfin	-	0,75
Max fin rotation angle $[deg]$	Max_Delta	-	45
Min fin rotation angle $[deg]$	Min_Delta	-	-45
fin platform area Sfin $[m^2]$	Sfin	-	0,017
Thrusters Parameters			
Max Thrust (HT1)	HT1Max	-	1,12
Max Thrust (HT2)	HT2Max	-	1,12
Max Thrust (VT1)	VT1Max	-	1,12
e1 (HT1)	e1	Matrix 3x1	[1;0;0]
e2 (HT2)	e2	Matrix 3x1	[1;0;0]
e3 (VT1)	e3	Matrix 3x1	[0;0;1]
r1 (HT1)	r1	Matrix 3x1	[0;-0.11;0]
r2 (HT2)	r2	Matrix 3x1	[0;0.11;0]
r3 (VT1)	r3	Matrix 3x1	[0;0;-0.03]

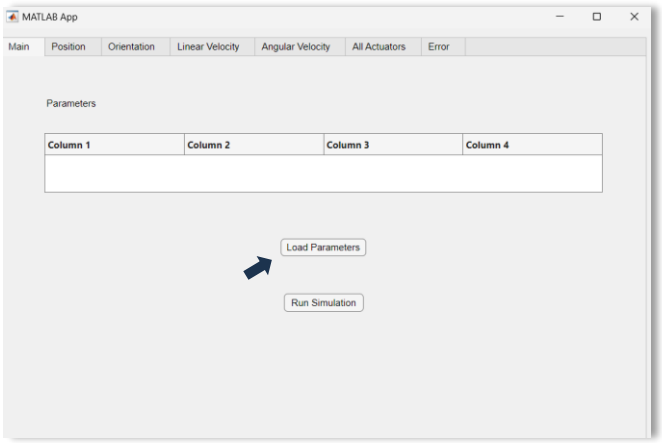


Figure 43 - How to load parameters from an excel file

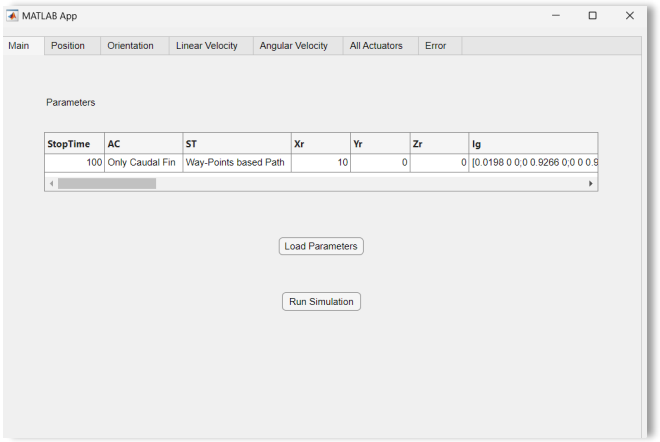


Figure 44 - Loaded parameters

2. *Starting the simulation of the Simulink model*: Through the "Run Simulation" button, users can start the simulation of the Simulink model "RoboticFish_NGC.slx". This button opens the Simulink model and starts the simulation. Therefore, the user will be able to see the behaviour of the robotic fish with the parameters set by him. Control via the PID controller is only guaranteed if users decide to start the simulation with the parameters provided. To be able to start the simulation with such data, simply go to start the simulation via the appropriate button without first loading the parameters from the Excel file. It should be emphasised that once the parameters have been changed via the "Load Parameters" button, if users decide to save the model with these parameters, they will remain stored; in this case, it will be necessary to reset the parameters with the values provided in order to display the behaviour of the robotic fish with the correct PID control.

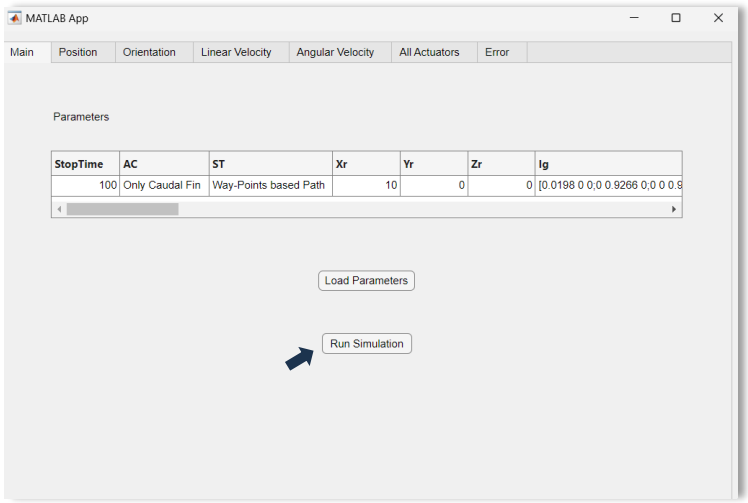


Figure 45 - How to run the simulation

3. *Plot of the variables of interest in the simulation*: after running the simulation, the app presents the results obtained via graphs. i.e. users can graphically display the position of the robotic fish along the x, y and z axes, or the rotations on Roll, Pitch and Yaw angles and the respective linear and angular velocities; as well as the behaviour of the steering angle $\bar{\theta}$ and the frequency of the motor's actuator.

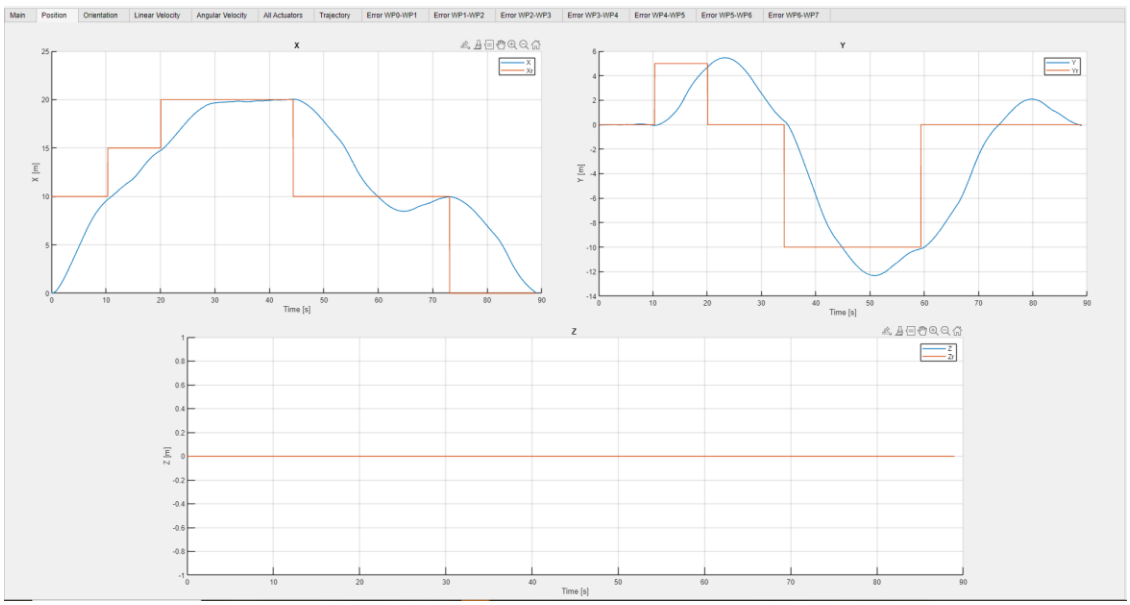


Figure 46 – MAXFISH position (blue) along the x, y and z axes with respect to the reference signal (red)

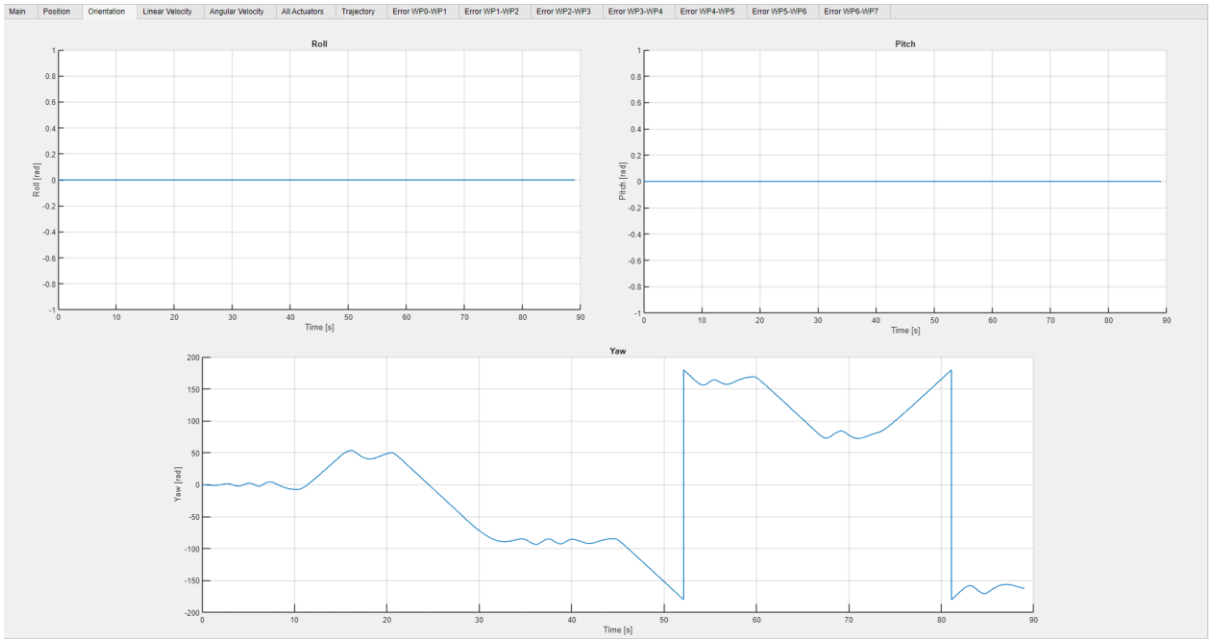


Figure 47 – MAXFISH rotation on roll, pitch and yaw

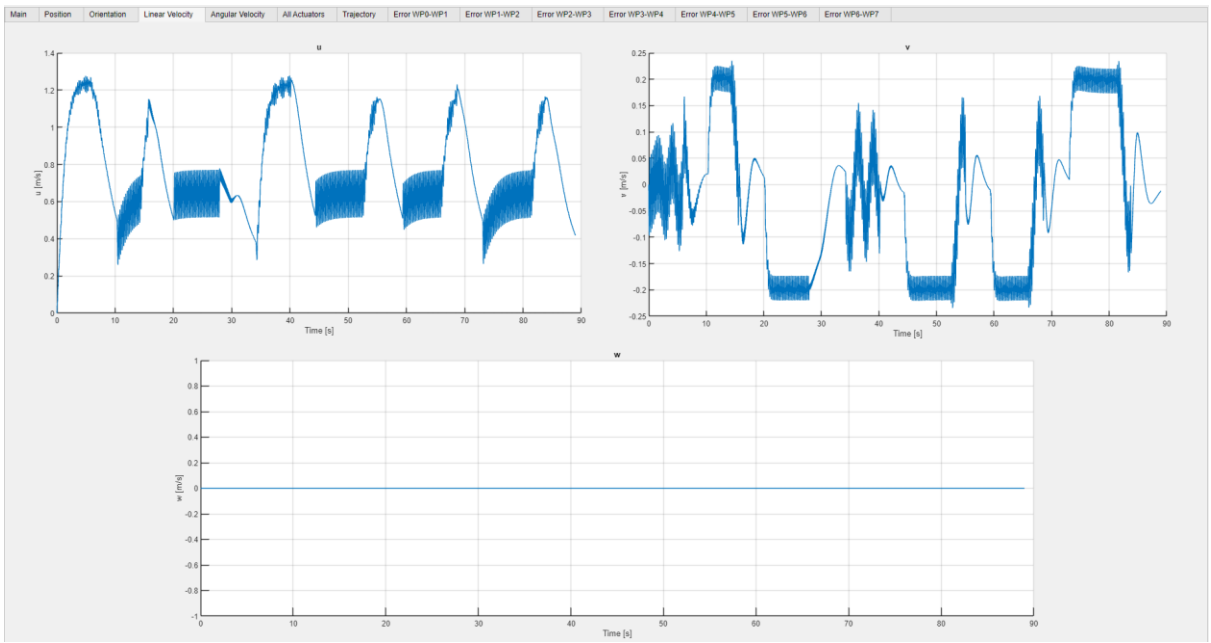


Figure 48 – MAXFISH linear velocities along the x, y and z axes

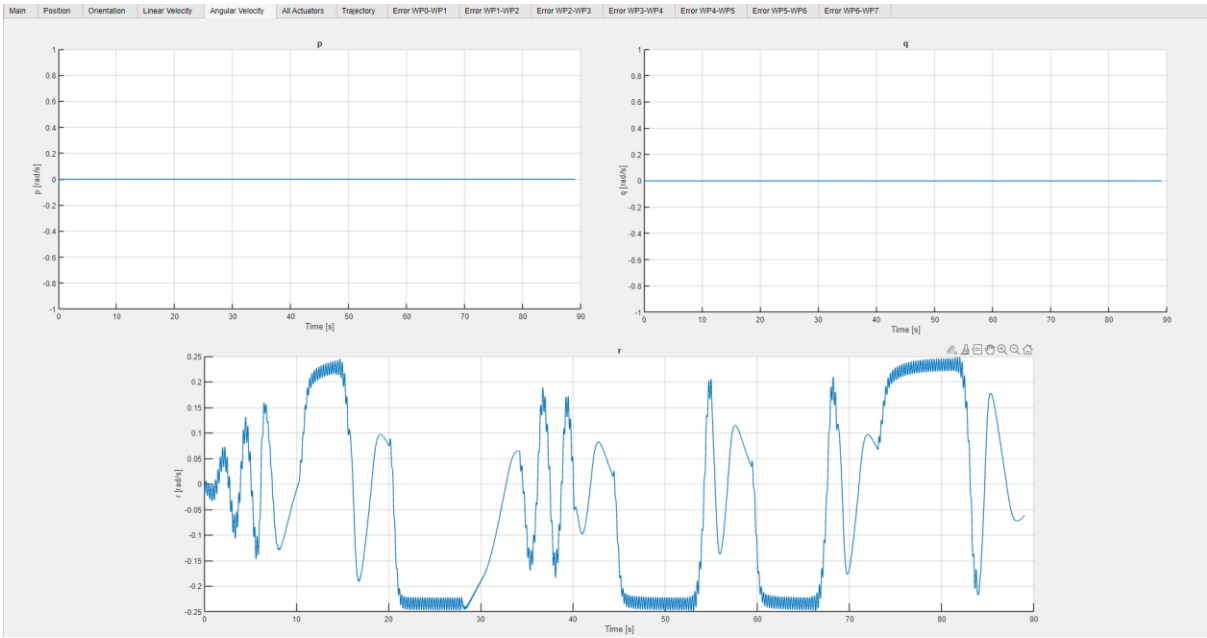


Figure 49 – MAXFISH angular velocities on roll, pitch and yaw

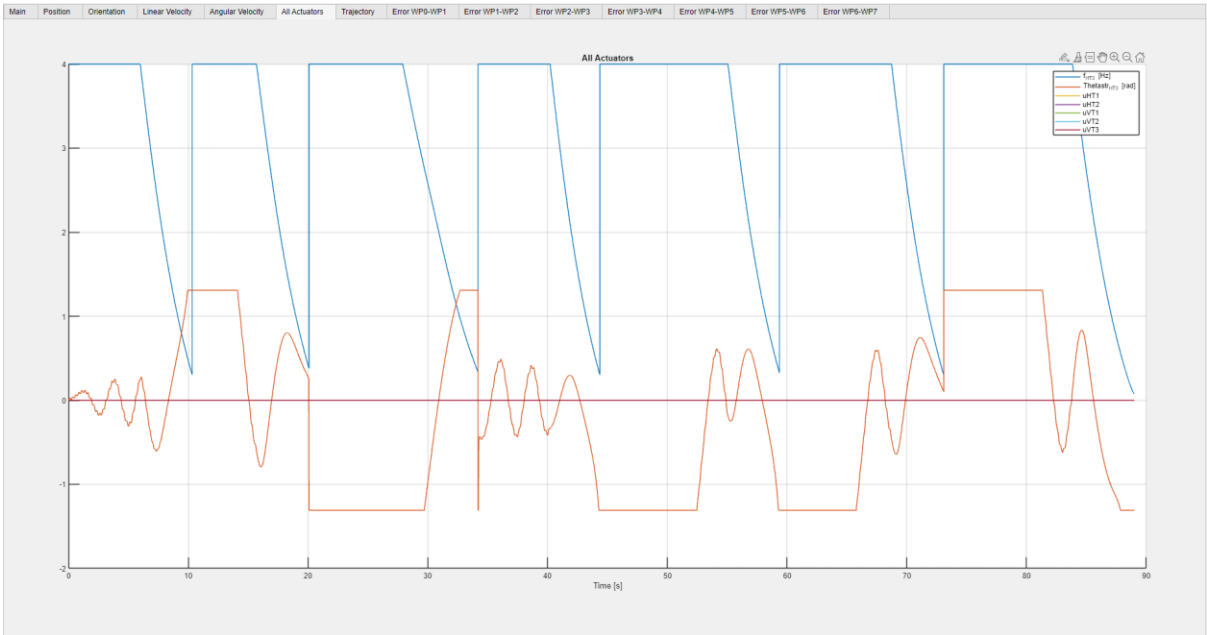


Figure 50 – All the actuators: motor frequency [Hz] in blue and steering angle [rad] in red. The other actuators are turned off since only the tail is being used.

4. *Plot of the trajectory taken by MAXFISH and the desired trajectory*: the user also has the option of visualising in the GUI the trajectory taken by the robotic fish in relation to the desired one. In this way, it is possible to see how faithful one is to the other.

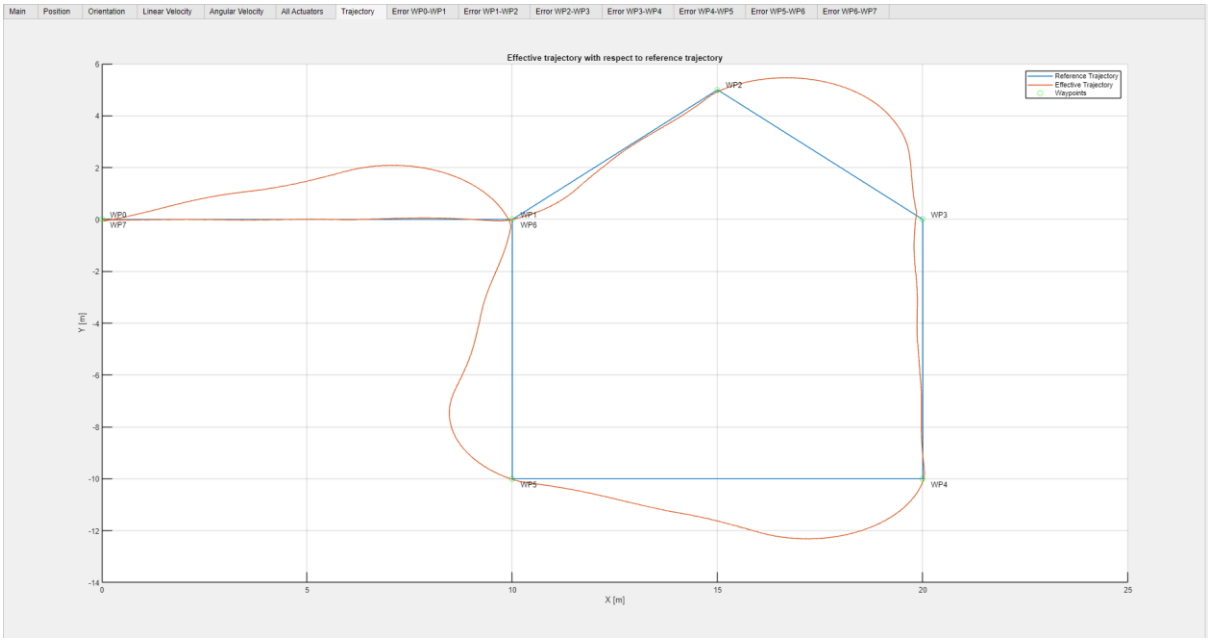


Figure 51 - The trajectory performed by the robotic fish (red) with respect to the desired one (blue).

5. *Calculation of the mean and Standard Deviation of the trajectory*: another important feature of the application is the ability to display the mean and standard deviation of the various trajectory sections made by MAXFISH. In particular, paths made between two successive waypoints were taken into account. Through these features, it is possible to mathematically assess the consistency of the trajectory made by the robotic fish with respect to the desired trajectory.

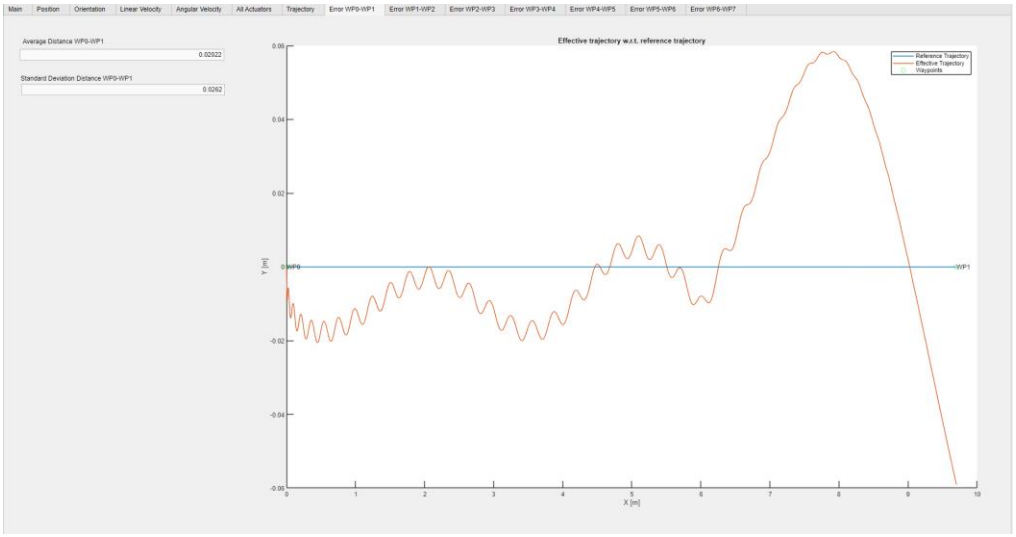


Figure 52 - Mean and Standard deviation of the trajectory from WP0 to WP1

6. *Starting the simulation of the tail kinematics on MATLAB*: to compensate the lack of a faithful graphic representation of the AUV MAXFISH tail in the digital twin, it was decided to represent the behaviour of the three-jointed tail in MATLAB. This representation starts after the end of the Simulink model simulation, since in order to have a faithful representation, some data are needed, which can only be obtained at the end of the simulation. The representation of the MAXFISH tail is, however, carried out in air and not in water. Finally, after simulating the tail of the robotic fish, graphs of the position, velocity and acceleration of the joints and the determinant of the Jacobian are displayed.

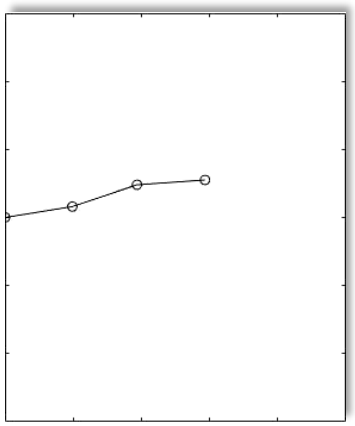


Figure 53 – Frame of the MAXFISH tail's simulation on MATLAB

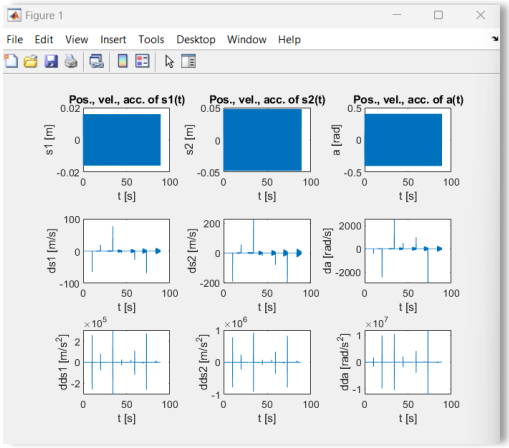


Figure 54 - Plot of position, velocity, and acceleration of the cartesian variables

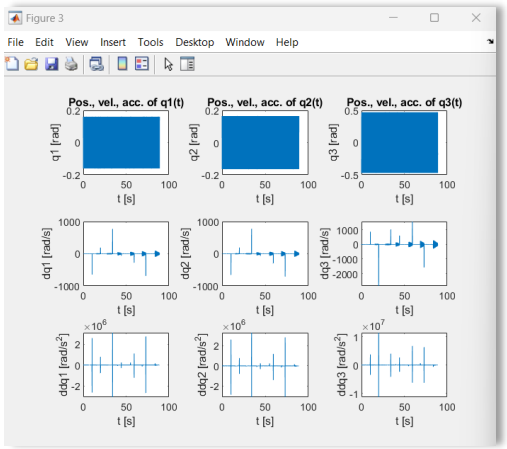


Figure 55 - Plot of position, velocity and acceleration of the joints variables

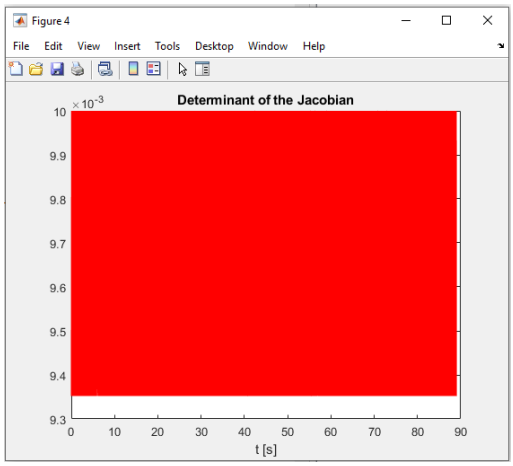


Figure 56 - Plot of the determinant of the Jacobian

3 Management

This chapter presents all the management aspects of the project and its validation plan.

It can be divided into five parts:

- The first sub-chapter describes the TC and TP used to evaluate the functionality of the product;

- The second sub-chapter shows the GANTT diagram, the V&V model and the TRL achieved;

- The third sub-chapter provides the list of KPIs and their level of achievement;

- The last sub-chapter elaborates on the end-user acceptance criteria.

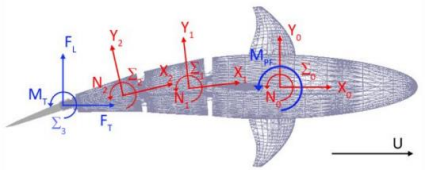
3.a Methodological framework for validation

To verify the accuracy of the presented model, specific procedures were implemented to determine whether the work had been performed correctly.

KPI.1 Has the trajectory control the precision required?

TC1.1 The fish must reach the first waypoint (located at position (10,0,0)) within 11 seconds and it shall arrive in a neighbourhood of the desired point of radius ε = 0,3 m.

TP1.1.1 Place a 'scope' block in Simulink to compare the trajectory taken by the fish with the desired trajectory. Check that the final position reached is no more than 0.3m away from the desired point and that this position is reached within 11 seconds.



Result Levels: [ko – ok]

Acceptance Level: $[\varepsilon < 0{,}3\ m\ \&\ t \leq 11\ s]$

Reports: [date, result]

18/12/2023 **Ok**

The robotic fish reaches the first waypoint in 10.33 s.

TC1.2 The fish must reach all waypoints within 90 seconds, and it shall arrive in a neighbourhood of the each one on a radius ε = 0,3 m.

TP1.2.1 Compare the graph of the trajectory taken by the robotic fish with the desired trajectory (pentagonal trajectory, 7 waypoints) in the application developed in "App Designer". Verify through the scopes that the position of the fish reaches each waypoint within an ε neighbourhood of 0.3 m and that the fish completes the entire desired path within a time limit of 90 s. The 7 waypoints to be reached in

sequence are as follows:
$WP0\ (0,0,0)\ WP1\ (10,0,0)\ WP2\ (15,5,0)\ WP3\ (20,0,0)\ WP4\ (20,-10,0)$

$WP5\ (10,-10,0)\ WP6\ (10,0,0)\ WP7\ (0,0,0)$

Result Levels: [ko – ok]

Acceptance Level: $[\varepsilon < 0{,}3\ m\ \forall\ WP\ \ \&\ t \leq 90\ s]$

Reports: [date, result]

18/12/2023 **Ok**

The robotic fish performs the pentagonal trajectory in 88.39s and respects the ε of 0.3 m of each waypoint.

KPI.2 Has the DC motor speed control the accuracy required?

TC2.1 Let the motor with torque 1.5 Nm, maximum frequency 4Hz and rated power 40 W, maintain a constant speed of 2m/s for at least 10m.

TP2.1.1 Test with a 'scope' on Simulink the motor speed during simulation, check that it maintains constant speed at 2m/s for at least 10m.

Result Levels: [ko – ok]

Acceptance Level: $[v = 2\ m/s \text{ for at least } 10 \text{ meters}]$

Reports: [date, time, result]

The project specifications changed after the KPIs were drawn up. This KPI was no longer required by the new specifications.

TC2.2 Transient extinction time must be less than 0.5 s.

TP2.2.1 Check with a scope on Simulink that the transient extinction time is < 0.5 s.

Result Levels: [ko – ok]

Acceptance Level: $[t < 0.5\ s]$

Reports: [date, time, result]

The project specifications changed after the KPIs were drawn up. This KPI was no longer required by the new specifications.

KPI.3 Does the simulation on Simulink take place in real time?

TC3.1 Verify that the simulation of the trajectory made by the fish on Simulink takes place in real time with a maximum error between simulation time and real time <0.1 s.

TP3.1.1 Check with a stopwatch that the simulation on Simulink respects the real time.

Result Levels: [ko – ok]

Acceptance Level: $[Error\ between\ simulation\ time\ and\ real\ time\ < 0.1\ s]$

Reports: [date, time, result]

18/12/2023 **Nearly**

On a PC with the characteristics shown below, the robotic fish performs the pentagonal trajectory in 50 s compared to the simulated 88.39 s. This means that real time is feasible, it was simply not completed due to lack of time.

Simulink solver: Ode4 with the Runge Kutta method

Step Time: 0.00025 s

RAM: 16 GB

SSD: 256 GB

Processor: Intel i7 11th Generation

GPU: NVIDIA GeForce MX450 (Integrated)

KPI.4 Is the graphical interface on MATLAB correctly implemented?

TC4.1 Operating app designer end-users

TP4.1.1 Opening the MATLAB app for end- user.

TP4.1.2 Start the simulation with default parameters.

TP4.1.3 The user has the possibility of modify fish parameters, but the correct PID control is only guaranteed with the parameters provided.

Result levels: [ko-ok]

Acceptance Level: [the app must work from start to finish without the slightest interruption]

Reports: [date, time, result]

18/12/2023 **Ok**

The application works correctly from start-up to closure.

KPI.5 2D digital twin of kinematic tail behaviour on MATLAB

TC.5.1 Check that the behaviour of the 3R jointed tail on MATLAB is consistent with the simulation previously carried out in Simulink.

TP5.1.1 Start the simulation of the tail kinematics in MATLAB and check that the bias in this simulation respects the movements made by the tail of the digital twin in Simulink.

TP5.1.2 Verify that the simulation time of the tail kinematics in MATLAB is the same as that of the MAXFISH simulation in Simulink.

Result levels: [ko-ok]

Acceptance Level: [Tail behaviour must be coherent with the simulation]

Reports: [date, time, result]

18/12/2023 **Ko**

The behaviour of the 3R jointed tail on MATLAB is not coherent with that of the tail of the digital twin on Simulink, as the tail in MATLAB is simulated in air and does not take hydrodynamic forces into account.

KPI TABLE

The following table summarizes the list of test cases and procedures, with their respective results.

FAT TEST		TEST CASES AND PROCEDURES			RESULTS
CODE	FUNCTION	TC NAME	PROCEDURE	THRESHOLD	OK / KO / NEARLY
KPI.1	Accuracy of the trajectory control	TC1.1	The fish must reach the first waypoint (located at position (10,0,0)) within 11 seconds and it shall arrive in a neighbourhood of the desired point of radius ε = 0,3 m.	$\varepsilon < 0{,}3\ m$ $\mathrm{t} \leq 11\ \mathrm{s}$	**OK**
		TC1.2	The fish must reach all waypoints within 90 seconds, and it shall arrive in a neighbourhood of the each one on a radius ε = 0,3 m.	$\varepsilon < 0{,}3\ m\ \forall\ WP$ $t \leq 90\ s$	**OK**
KPI.2	Accuracy of the DC motor speed control	TC2.1	Let the motor with torque 1.5 Nm, maximum frequency 4Hz and rated power 40 W, maintain a constant speed of 2m/s for at least 10 m.	$v = 2\ m/s$ $for\ at\ least\ 10\ m$	-

		TC2.2	Transient extinction time must be less than 0.5 s.	$t < 0.5\ s$	-
KPI.3	Check that the simulation on Simulink takes place in real time	TC3.1	Verify that the simulation of the trajectory made by the fish on Simulink takes place in real time with a maximum error between simulation time and real time <0.1 s.	Error between real time and simulation time $< 0.1\ s$	**NEARLY**
KPI.4	Implementation of a GUI on Matlab	TC4.1	Operating app designer end-users	The app must work from start to finish without the slightest interruption.	**OK**
KPI.5	2D digital twin of kinematic tail behaviour on MATLAB	TC5.1	Check that the behaviour of the 3R jointed tail on MATLAB is consistent with the simulation previously carried out in Simulink.	Tail behaviour must be coherent with the simulation.	**KO**

3.b GANTT, V&V Model & TRL

One of the key objectives of the project was to simulate the experience of working inside a company.

Therefore, the GANTT chart, the V-Model and the project's target Technology Readiness Level (TRL) were defined.

GANTT CHART

The Gantt chart is a visual planning tool used to represent the activities of a project along a time axis. The activities are displayed as horizontal bars, the length of which represents the expected duration of the activity. This tool provides a clear overview of the project timeline, the dependencies between activities and the allocated resources.

The realized GANTT is shown in Figure 57.

SuperFin's Gantt Chart

ACTIVITIES	EXPECTED START	EXPECTED DURATION	ACTUAL START	ACTUAL DURATION	PERCENTAGE OF COMPLETION
WP1 - Project Management	16/10/2023	64	16/10/2023	64	100%
Task 1.1: Initiating	16/10/2023	7	16/10/2023	7	100%
Task 1.2: Planning	23/10/2023	7	23/10/2023	7	100%
Task 1.3: Executing	30/10/2023	45	30/10/2023	45	100%
Task 1.4: Monitoring & Control	30/10/2023	45	30/10/2023	45	100%
Task 1.5: Closing	14/12/2023	5	14/12/2023	5	100%
WP2 -State of Art and Definition of Objectives	16/10/2023	59	16/10/2023	59	100%
Task 2.1: Technical SoA	16/10/2023	52	16/10/2023	52	100%
Task 2.2: Commercial SoA	16/10/2023	52	16/10/2023	52	100%
Task 2.3: Definitition of requirements	30/10/2023	7	30/10/2023	7	100%
Task 2.4: Drafting of the documentation	07/12/2023	7	07/12/2023	7	100%
WP3 - Development of Mathematical Model	30/10/2023	28	30/10/2023	28	100%
Task 3.1: Development of fin model	30/10/2023	14	30/10/2023	14	100%
Task 3.2: Development of fish model	30/10/2023	21	30/10/2023	21	100%
Task 3.3: Implementation of Mathematical Model on Matlab & Simulink	30/10/2023	21	30/10/2023	21	100%
Task 3.4: Drafting of the documentation	20/11/2023	7	20/11/2023	7	100%
WP4: Design of a velocity and position control on Simulink	20/11/2023	21	20/11/2023	21	100%
Task 4.1: Requirement's analysis	20/11/2023	2	20/11/2023	2	100%
Task 4.2: Implementation of PID control on Simulink	22/11/2023	12	22/11/2023	12	100%
Task 4.3: Testing	04/12/2023	5	04/12/2023	5	100%
Task 4.4: Drafting of the documentation	04/12/2023	7	04/12/2023	7	100%
WP5: Design of the Graphic User Interface on Matlab	04/12/2023	14	04/12/2023	14	100%
Task 5.1: Requirement's analysis	04/12/2023	2	04/12/2023	2	100%
Task 5.2: Implementation of the Graphic User Interface on Matlab	06/12/2023	7	06/12/2023	7	100%
Task 5.3: Testing	13/12/2023	5	13/12/2023	5	100%
Task 5.4: Drafting of the documentation	04/12/2023	14	04/12/2023	14	100%
WP6: Testing and validation	30/10/2023	50	30/10/2023	50	100%
Task 6.1: Requirement's analysis (KPI)	30/10/2023	21	30/10/2023	21	100%
Task 6.2: Validation Planning	30/10/2023	21	30/10/2023	21	100%
Task 6.3: FAT in simulation (TRL2)	04/12/2023	14	04/12/2023	14	100%
Task 6.4: Hazard identification and risk validation (HIRA)	04/12/2023	14	04/12/2023	14	100%
Task 6.5: Analysis of validation finding and final improvements (TRL3)	11/12/2023	7	11/12/2023	7	100%
Task 6.6: Drafting of the documentation	04/12/2023	15	04/12/2023	15	100%

Figure 57 - Gantt Chart: Work Packages and Tasks

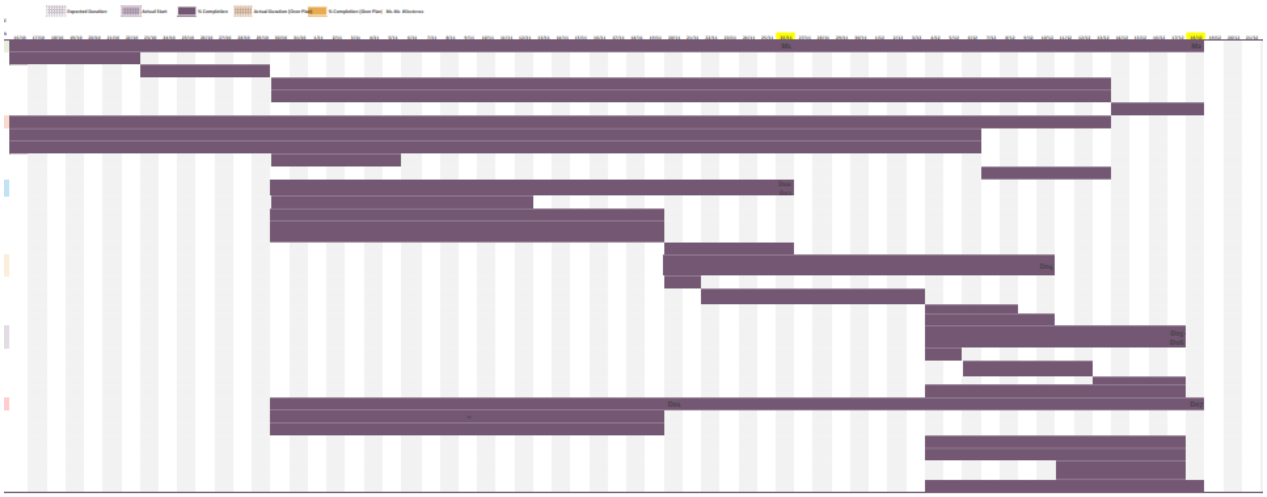


Figure 58 - Gantt Chart: Activities duration

The Gantt was organised into several Work Packages, in order to outline the various steps to be taken to achieve the project goals.

The project started on 16/10/2023 and ended on 20/12/2023. As can be seen from the diagram, the part of the project that occupied most of the time was the development of the mathematical model and the position and velocity control.

There was also a testing and validation part, which initially involved the definition of KPIs and a validation plan, and later, a verification part of what was developed.

Furthermore, the chart was formulated with the assumption of dedicating three to four sessions per week to project work. Each session commenced with a daily stand-up meeting aimed at setting daily goals and ensuring comprehensive communication among all project components. Additionally, on a weekly basis, an extended meeting was held to delve into the overall project trends and developments.

V&V Model

The V-Model, is a methodology that links the development and test phases, emphasising the importance of verification and validation throughout the project life cycle. The aim is to simulate a rigorous and structured approach, where quality and reliability are paramount.

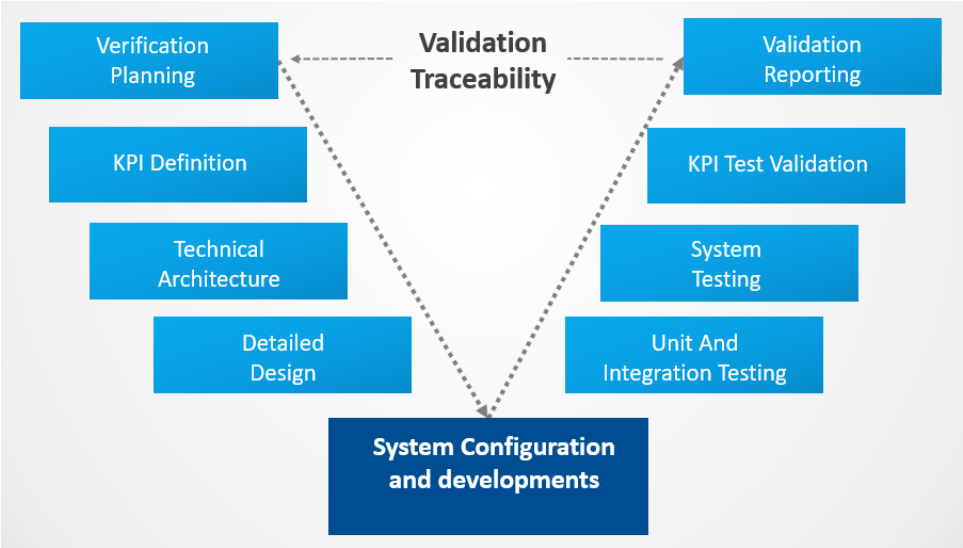


Figure 59 - V&V Model

Besides the project GANTT, a V&V model was created within which the various project phases were placed.

The tasks extracted from the Gantt chart are delineated below, categorized within the phases of the V model:

- Verification Planning: Organization and Work Division, Analysis of provided materials and Research.
- KPI Definition: Accuracy of trajectory control, DC motor speed control on Simulink, verify that the simulation on Simulink takes place in real time, GUI on MATLAB, 2D digital twin of kinematic tail behaviour on MATLAB.
- Technical Architecture: Study and analysis of the fish tail's model and the GUIZZO 6.0 mathematical model.
- Detailed Design: Implementation and simulation of fish tail kinematics model and fish mathematical model, implementation of a GUI.
- Unit and Integration Testing: Simulation tests on Matlab and Simulink.
- System Testing: Final test.
- KPI Test Validation: KPI check.
- Validation Reporting: Final check.

TRL

The TRL is a numerical scale that assesses the degree of technological maturity of a project.

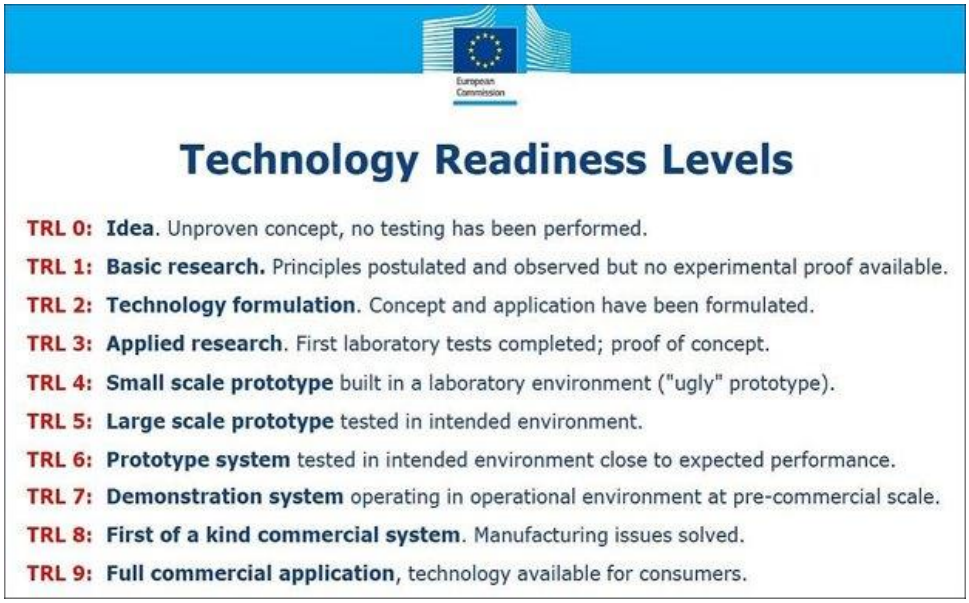


Figure 60 – TRL Table

The project begins at a Technology Readiness Level (TRL) of 2, owing to the extensive foundation of basic scientific research on the subject. After establishing the fundamental principles, the theory is then implemented to progress towards the development of the ultimate product.

The target achieved is a TRL 3 because the project stopped once the simulation of the MAXFISH robotic fish was carried out in Simulink. Therefore, it was not possible to perform physical tests in the laboratory, but several simulation tests were performed.

The validation of TRL 3 was confirmed through the presented results, serving as evidence that the performance criteria for the intended application were successfully met.

3.c KPI

To understand whether our goal was met, we defined some Key Performance Indicators at the beginning of the project.

Code	Function	Number	Description	Threshold	OK / KO / NEARLY
KPI.1	Accuracy of the trajectory control	**1.1**	Reaching the first waypoint within 11 s in a neighborhood of radius ε = 0,3 m of the desired point	$\varepsilon < 0{,}3$ m $t \leq 11$ s	**OK**
		1.2	Reaching all waypoints within 90 s in a neighborhood of radius ε = 0,3 m of each point.	$\varepsilon < 0{,}3$ m $\forall$ WP $t \leq 90$ s	**OK**
KPI.2	Accuracy of the DC motor speed control	**2.1**	The motor's speed shall be 2m/s for at least 10 m.	$v = 2\,m/s$ $for\ at\ least$ $10\,m$	-
		2.2	Transient extinction time must be less than 0.5 s	$t < 0.5\,s$	-
KPI.3	Real Time simulation on Simulink	**3.1**	Check if the simulation on Simulink is Real Time	Error between real time and simulation time $< 0.1\,s$	**NEARLY**
KPI.4	Implementation of a GUI on MATLAB	**4.1**	Operating app designer end-users	The app must work from start to finish without the slightest interruption	**OK**
KPI.5	2D digital twin of kinematic tail behaviour on MATLAB	**5.1**	Check if the behaviour of the 3R jointed tail on MATLAB is consistent with the simulation previously carried out in Simulink.	Tail behaviour must be coherent with the simulation.	**KO**

As shown in the table above, KPI.1 and KPI.4 were successfully achieved.

Regarding KPI.2, this was not implemented, as the customer's requirements changed during the project development.

With reference to KPI.3, the simulation in Simulink of the AUV MAXFISH does not fulfil the real time, as the simulation actually lasts 50 seconds on our system as opposed to the desired 88.39 s. Therefore, what was stated in the KPIs before the development is not fulfilled, but this can be considered an advantage, as the difference between the two timings is substantial. This implies that real-time simulation is absolutely feasible as the calculation process is not computationally complex at all. The KPI was not achieved due to lack of time and is suggested as future implementation.

Finally, KPI.5 is not satisfied as the behaviour of the three-jointed tail is not consistent with what was previously simulated in Simulink. This is since the tail in MATLAB was simulated in empty space and not in water, as required.

3.d Indicators for Stakeholder acceptance and evaluation

Our main stakeholders are researchers and engineers who want to study and build an AUV bio-inspired by the locomotion and behavior of real fish, capable of moving autonomously and performing desired trajectories within a natural habitat, such as seas and oceans.

In particular, with the implementation of a mathematical model, a Digital-Twin in Simulink and an App that allows to start the simulation even with different physical parameters of the fish from those implemented, we provided the basis for the design and realization of a robotic fish.

End-users can appreciate the satisfied KPI, discussed in the previous paragraph 3.c.

In addition, users are provided with the user manual presented in Appendix 1 to understand the proper use of the App.

We have also provided a questionnaire for end users of the product to fill out. It aims to gather more information about the results to possibly improve the App in future versions of the product. The questionnaire is presented in Appendix 3.

4 Validation plan results

An extensive study was conducted to achieve the main objectives of this project and this section reports the various changes made during the course of the work that then led to the validation of the final results.

MAXFISH had to reach the first waypoint (located at position (10,0,0)) within 11 seconds and arrive in a circle of the desired point of radius ε = 0.3 m. After the simulation, we checked the "scopes" on Simulink making sure that the fish carried out the planned trajectory with the desired accuracy.

In addition, the fish had to reach all waypoints within 90 seconds and arrive in a neighborhood with radius ε = 0.3 m for each waypoint. Again, after the simulation we checked the "scopes" by making sure that the fish made the trajectory of 7 Waypoints within 90 seconds.

Another objective was to validate the accuracy of the DC motor speed control, but this was not tested, as the customer's requirements changed during the project development.

Then, we had to ensure that the simulation on Simulink took place in real time. So, we repeated the simulation 10 times with a timer and verified that the fish completed the trajectory in about 60 actual seconds against the 89 seconds of simulation. This implies that real-time simulation is absolutely feasible as the calculation process is not computationally complex at all.

Finally, after introducing the latest changes suggested by our main contributors (as the trajectory evaluation), we tested the App making sure it worked consistently with the default fish parameters at least 10 times. Furthermore, we verified that the App did not give problems with different but consistent parameters at least 10 times.

The validation of the KPIs was then formalized in the KPI table in section 3.c.

5 Conclusions and future improvement

In conclusion, the implementation of a robotic fish in Simulink, along with the development of its digital twin within the same simulation environment, has proven to be a valuable and comprehensive approach. This project has provided a versatile platform for studying and refining the behaviour, control algorithms and performance of the robotic fish.

The insights gained from this work contribute to the ongoing advancements in the field of Autonomous Underwater Vehicles (AUVs) and pave the way for more sophisticated and efficient designs in the future, offering a powerful tool for researchers and engineers to iteratively refine and innovate in a simulated environment before deployment in real-world scenarios.

For future implementations, it is suggested to:

- Find fish tail parameters $a_0,\ a_1, a_2$ to consider the contribution of the steering angle $\bar{\theta}$
  $\tau_{yawHT3} = M_{yaw} - \frac{1}{4}\rho U^2 SL[(a_0 + a_1 St + a_2 St^2)] \sin(\bar{\theta})$
  where a_0, a_1, a_2 $[Kg\ m^2]$ are caudal fin parameters achievable through multibody simulation, by using MSC Adams View.
  This has already been discussed in the section *2.c.1.3 Robotic-Fish Model & Propulsion System*.

- Reassess the use of integral action for control on surge force τ_X.
  As already covered in the section *2.c.1.2 Guidance and Control System*, it was decided not to use integral action because this leads to frequency saturation in a short time, thus presenting the phenomenon of wind-up.

- Design the *MAXFISH* model and reproduce it in a simulative environment, as Simscape.
  In the context of this project, the utilization of *Guizzo 6.0* model was employed. While suitable for preliminary investigations and testing, it is imperative to employ a model that accurately represents the intrinsic characteristics of the actual robotic fish in order to validate the precise functionality of the tail within an underwater environment.

- Implement the pectoral fins for steering control (Yaw), described in "A NOVEL CLASS OF BIO-INSPIRED UNDERWATER ROBOTS" by Engineer Daniele Costa [1].

- Use pectoral fins for depth control (Pitch) already implemented in *Guizzo 6.0* model (and thus also in *MAXFISH* model), elucidated in the thesis "STUDY AND SIMULATION OF NGC SYSTEMS FOR BIOMIMETIC ROBOTS" by Engineer Waqas Ali Waqar [5].

- Introduce disturbances and water currents within the simulation framework to enhance the control of the robotic fish in progressively realistic environments and validate their behavioral responses.

- Implement Real-Time in simulation. The feasibility of this task has been substantiated in KPI.3, in section 3.c. To achieve this objective, we recommend utilizing Simulink's "Desktop Real-Time" add-on.

Bibliography

[1] Costa, D. (2022). A NOVEL CLASS OF BIO-INSPIRED UNDERWATER ROBOTS. INTERNATIONAL JOURNAL OF MECHANICS AND CONTROL, 23(2), 23-35.

[2] Costa, D., Palmieri, G., Palpacelli, M. C., Scaradozzi, D., & Callegari, M. (2020). Design of a carangiform swimming robot through a multiphysics simulation environment. Biomimetics, 5(4), 46.

[3] Fossen, T. I. (2011). Handbook of marine craft hydrodynamics and motion control. John Wiley & Sons.

[4] Ciuccoli, N. (2019). Intelligent Systems for the Exploration of Structured and Complex Environments.

[5] Waqar, W. A., Scaradozzi, D., Ciuccoli, N., & Costa, D. STUDY AND SIMULATION OF NGC SYSTEMS FOR BIOMIMETIC ROBOTS.

List of figures

Appendix 1 – User Manual

Before starting, ensure you have MATLAB version R2023b or newer.

Then, make sure that you have correctly installed the following add-ons on MATLAB:

- DSP System Toolbox;
- Signal Processing Toolbox;
- Simulink 3D Animation;
- Vehicle Dynamics Blockset.

To start, you have to open the folder "MAXFISH model" {Fig. 61}.

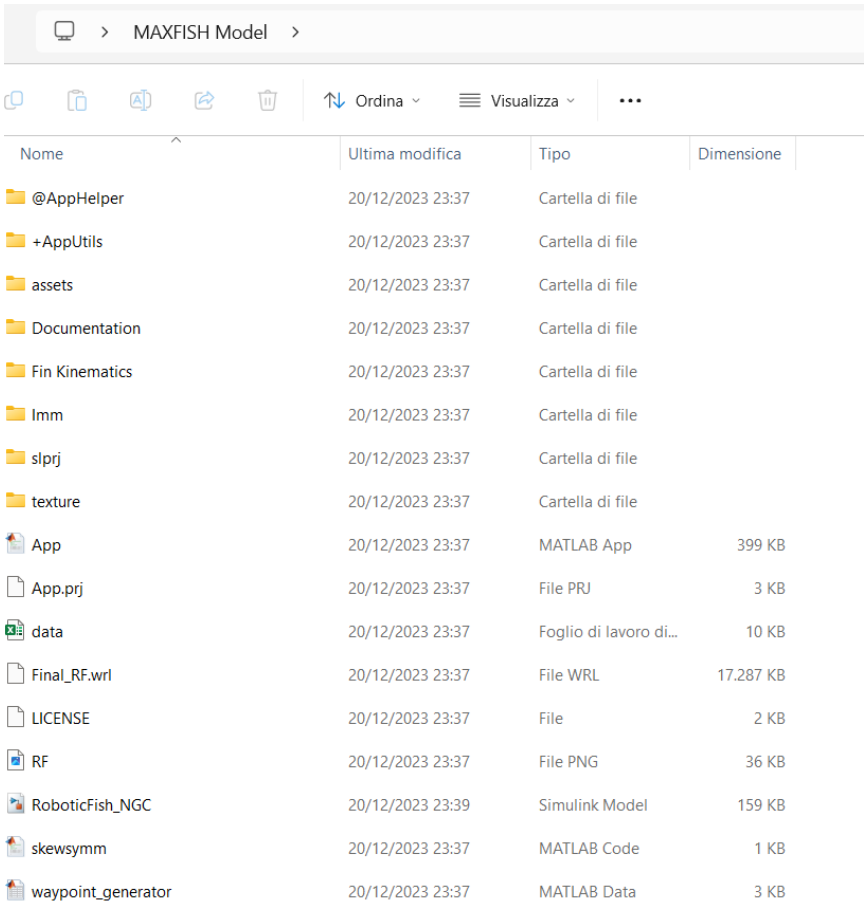
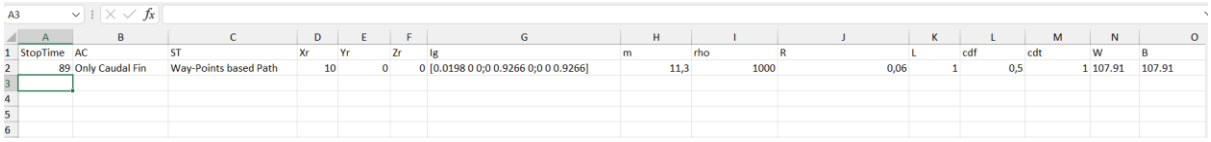


Figure 61 – "MAXFISH model" folder

Inside it, as you can see there is an Excel file called "data.xlsx", by opening it you will see the table of parameters of the robotic fish MAXFISH {Fig. 62}.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	StopTime	AC	ST	Xr	Yr	Zr	Ig	m	rho	R	L	cdf	cdt	W	B
2	89	Only Caudal Fin	Way-Points based Path	10	0	0	[0.0198 0 0;0 0.9266 0;0 0 0.9266]	11,3	1000	0,06	1	0,5	1	107.91	107.91
3															
4															
5															
6															

Figure 62 – "data.xlsx" Excel file: table of MAXFISH parameters

In this table, you can enter different parameters according to the fish model you want to use. In this case we recommend that you do not change them so that the demo will work properly.

The following table presents all modifiable parameters with the initial values with which MAXFISH was configured:

Parameter's name	Excel parameter's name	Possible Choices	Initial Values
Robotic Fish Parameters			
Stop Time $[s]$	StopTime	Only positive numbers	89
Actuator Choice	AC	Normal Only Propeller Only Caudal Fin	Only Caudal Fin
Select Target	ST	Way-Points based Path Way-point Position	Way-Points based Path
Way-Point position: Xr $[m]$	Xr	-	10
Way-Point position: Yr $[m]$	Yr	-	0
Way-Point position: Zr $[m]$	Zr	-	0
Inertia Tensor Ig $[Kg\ m^2]$	Ig	Matrix 3x3	[0.0198 0 0; 0 0.9266 0; 0 0 0.9266]
Mass m $[Kg]$	m	Only positive numbers	11,3
Seawater Density rho $[Kg/m^3]$	rho	-	1000

Body Radius R $[m]$	R	Only positive numbers	0,06
Body Length L $[m]$	L	Only positive numbers	1
Drag Weight cdf	cdf	-	0,5
Drag Weight cdt	cdt	[0.8-1.2]	1
Weight $[N]$	W	-	107.91
Buoyancy $[N]$	B	-	107.91
rG_B $[m]$	rG_B	Matrix 3x1	[0;0;0]
rB_B $[m]$	rB_B	Matrix 3x1	[0;0;0]
Linear Damping Matrix	DL	Matrix 6x6	diag([0;0;0;0;0;0])
Caudal Fin Parameters			
Kt0	Kt0	-	1,16
Kt1	Kt1	-	0,5
Kt2	Kt2	-	1,26
Km1	Km1	-	0,5
Km2	Km2	-	0,073
Kl1	Kl1	-	0,5
Kl2	Kl2	-	5,16
beta	beta	-	-4,95
sigma	sigma	-	-3,52
epsilon	epsilon	-	-3,51
c-Foil Mean Chord $[m]$	c	-	0,1
S-Foil area $[m^2]$	S	-	0
a0 $[Kg\ m^2]$	a0	-	0
a1 $[Kg\ m^2]$	a1	-	0
a2 $[Kg\ m^2]$	a2	-	0
Theta0-Oscillation amplitude $[deg]$	Theta0deg	-	23
Pectoral Fins Parameters			

fin lift coefficient cL	cL	-	2
fin axial position xfin $[m]$	xfin	-	0,11
fin axial position yfin $[m]$	yfin	-	0,11
fin axial position zfin $[m]$	zfin	-	0,75
Max fin rotation angle $[deg]$	Max_Delta	-	45
Min fin rotation angle $[deg]$	Min_Delta	-	-45
fin platform area Sfin $[m^2]$	Sfin	-	0,017
Thrusters Parameters			
Max Thrust (HT1)	HT1Max	-	1,12
Max Thrust (HT2)	HT2Max	-	1,12
Max Thrust (VT1)	VT1Max	-	1,12
e1 (HT1)	e1	Matrix 3x1	[1;0;0]
e2 (HT2)	e2	Matrix 3x1	[1;0;0]
e3 (VT1)	e3	Matrix 3x1	[0;0;1]
r1 (HT1)	r1	Matrix 3x1	[0;-0.11;0]
r2 (HT2)	r2	Matrix 3x1	[0;0.11;0]
r3 (VT1)	r3	Matrix 3x1	[0;0;-0.03]

The next step is to "Double click" on the file "App.mlapp" {Fig. 63}.

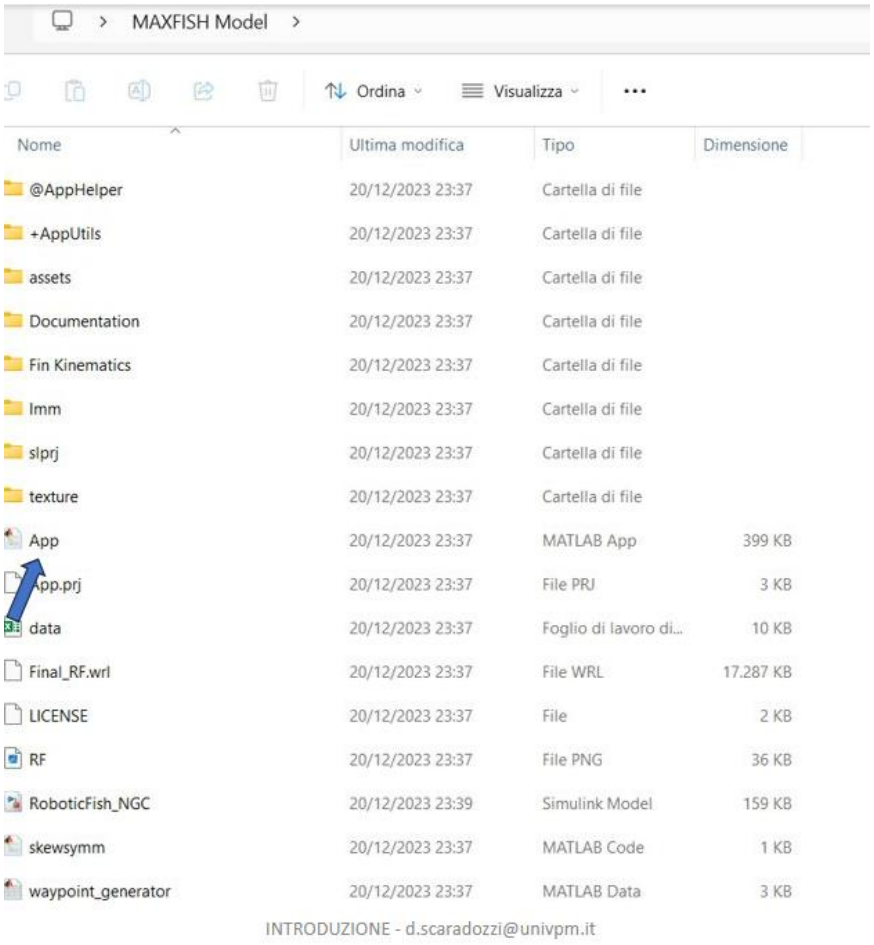


Figure 63 – "App.mlapp" in "MAXFISH Model" folder

It will open MATLAB. If MATLAB is already open, make sure you are in the path of the folder "MAXFISH model", it will also open at the same time the application {Fig. 50}.

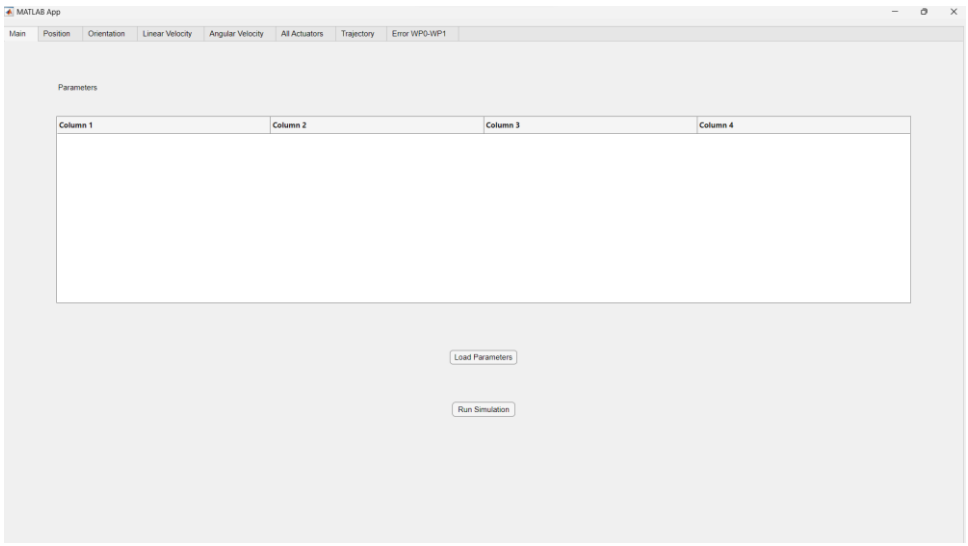


Figure 64 – MATLAB Application

At this point select "Load Parameters " to load the data from the Excel file {Fig. 51}.

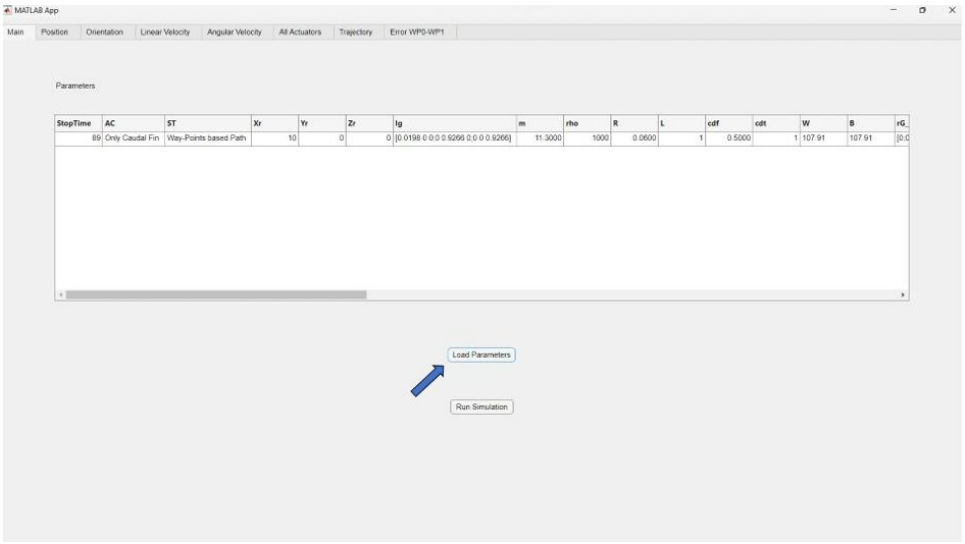


Figure 65 – "Load Parameters" on App

After the parameters are loaded, you can click on "Run Simulation" {Fig. 52}.

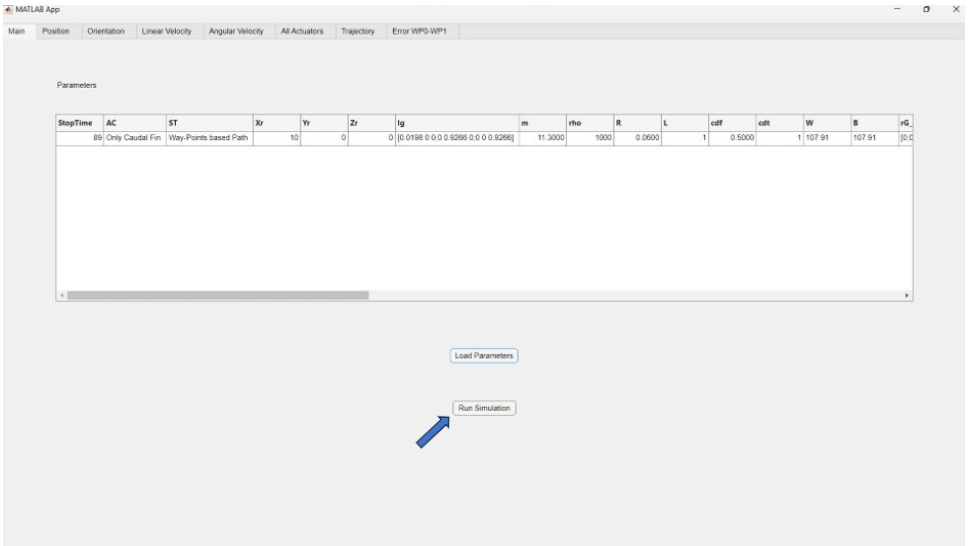


Figure 66 – "Run Simulation" on App

This button opens the Simulink model and starts the simulation {Fig. 53}. Therefore, the user will be able to see the behaviour of the robotic fish with the parameters set by him. Control via the PID controller is only guaranteed if user decides to start the simulation with the parameters provided.

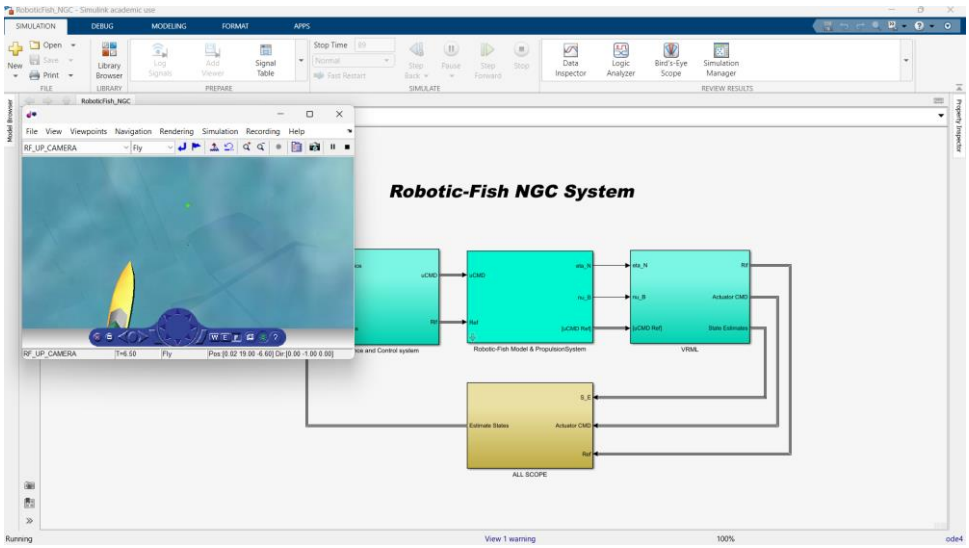


Figure 67 – MAXFISH simulation on Simulink

At the end of the simulation, after a few minutes (the time depends on the performance of the PC), a window will appear showing the 2D kinematics of the fish tail {Fig. 54}.

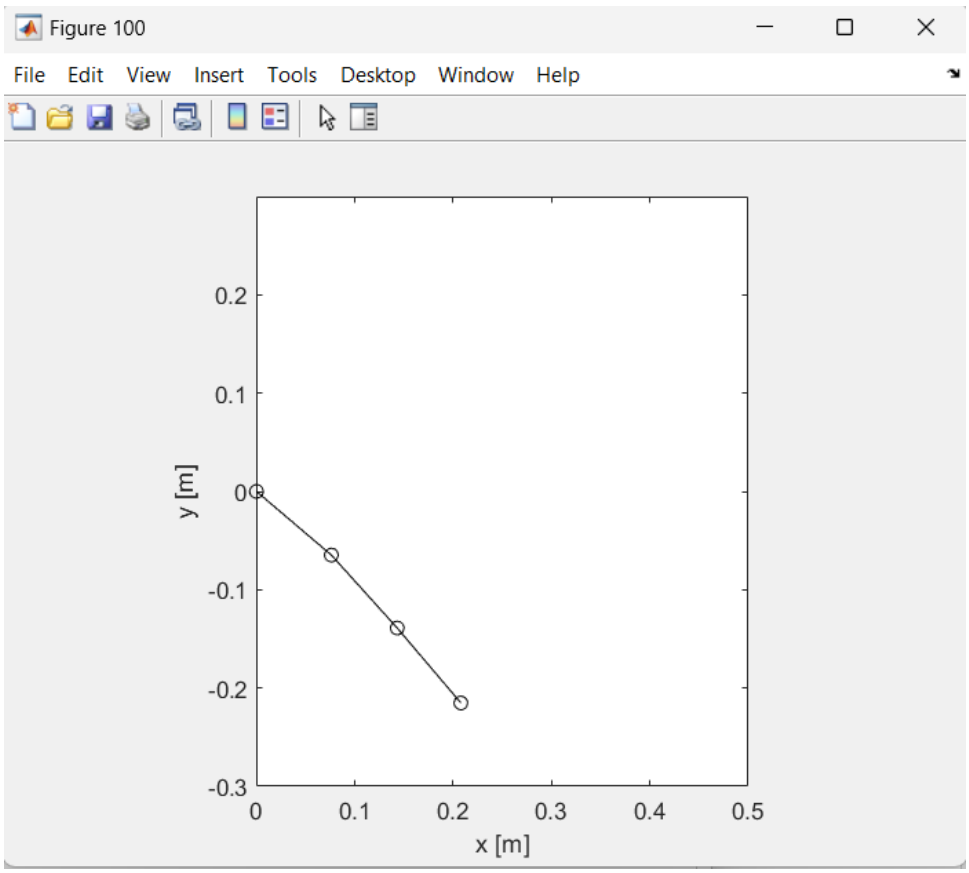


Figure 68 – Frame of the 2D kinematics of the fish tail

Now, by selecting the respective tab, it will be possible to display the position, orientation, linear velocities, and angular velocities graphs of the robotic fish. It will also be possible to display the graph of all actuators, to check the frequency and bias control of the fish's tail. In addition, it will be possible to compare the trajectory travelled in the simulation with the desired trajectory, and to analyse the trajectory errors for each section travelled {Fig. 55}.

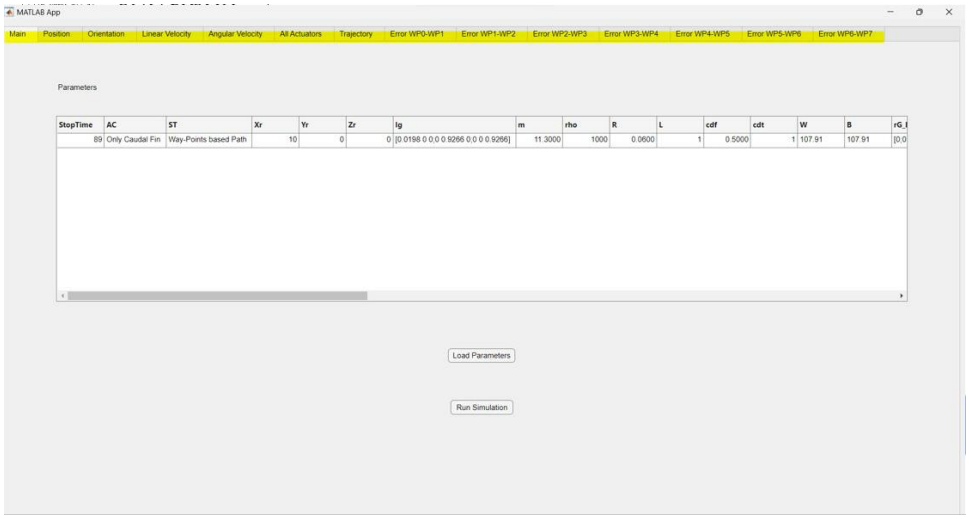


Figure 69 - The tabs concerned are highlighted in yellow.

Finally, graphs of the position, velocity and acceleration of the cartesian and joint variables and the determinant of the Jacobian are displayed.

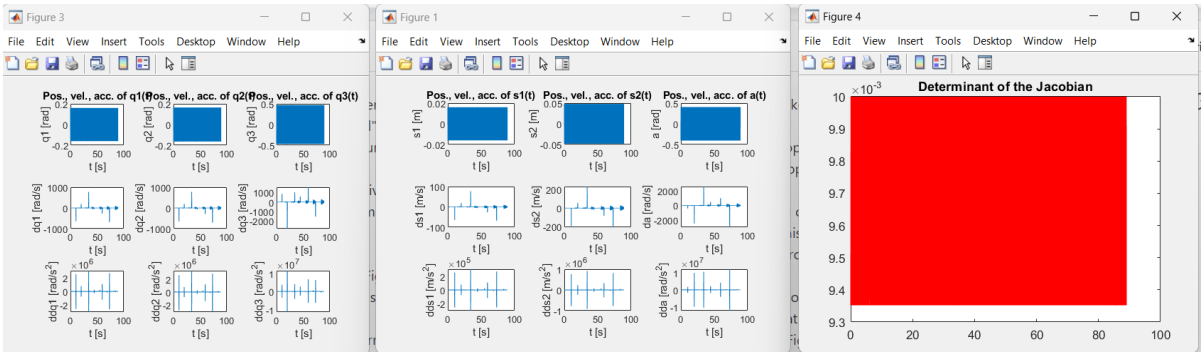


Figure 70 – Plots of cartesian and joint variables and the determinant of the Jacobian.

Appendix 2 – Software

Presented below are the paramount sections of code.

Fin Kinematics Code on MATLAB

```
% clc
% clear
% close all

%% Initialize variables

% Parameters
d = 0.1;                % link length [m]
a0 = deg2rad(23);       % amplitude of a(t) [rad]
s0 = 1.5*0.1;           % amplitude of s(t) [m]

% Geometric parameters of the cam joints
lambda_1 = 0.16;
lambda_2 = 0.16;
lambda_3 = 0.47;

% Initial rotations of the cam joints
delta_1 = 0;
delta_2 = 0;
delta_3 = 2.13;

% Input from simulation
t = simOut.tout;                % time [s]
dt = simOut.SimulationMetadata.ModelInfo.SolverInfo.FixedStepSize;
f = simOut.f_HT3;               % frequency [Hz]
theta = simOut.Thetastr_HT3;    % theta bias [rad]

% Initialize vectors
a_vect = 1:length(t);
s1_vect = 1:length(t);
s2_vect = 1:length(t);

for i=1:length(t)

    % Alpha vector [rad]
    % a_vect = a0*cos(2*pi*f*t+Delta);
    a_vect(i) = lambda_1*cos(2*pi*f(i)*t(i)+delta_1)+lambda_2*cos(2*pi*f(i)*t(i)+delta_2)+lambda_3*cos(2*pi*f(i)*t(i)+delta_3);

    % s1 vector [m]
    s1_vect(i) = d*(lambda_1*cos(2*pi*f(i)*t(i)+delta_1));

    % s2 vector [m]
    % s2_vect = s0*cos(2*pi*f*t);
    s2_vect(i) = d*(lambda_1*cos(2*pi*f(i)*t(i)+delta_1))+d*(lambda_1*cos(2*pi*f(i)*t(i)+delta_1)+lambda_2*cos(2*pi*f(i)*t(i)+delta_2));

end

%% Calculate the velocity of cartesian variables s1,s2,a (first derivative)
```

```
%% Calculate the velocity of cartesian variables s1,s2,a (first derivative)

% Note: we need to add 1 element to the vector because the function diff
% removes one element, for this reason we'll have a little problem with the graphs

% da
da_vect = diff(a_vect)/dt;
da_vect = [da_vect(1) da_vect];

% ds1
ds1_vect = diff(s1_vect)/dt;
ds1_vect = [ds1_vect(1) ds1_vect];

% ds2
ds2_vect = diff(s2_vect)/dt;
ds2_vect = [ds2_vect(1) ds2_vect];
```

```
%% Calculate the acceleration of cartesian variables s1,s2,a (second derivative)

% Note: we need to add 1 element to the vector because the function diff
% removes one element, for this reason we'll have a little problem with the graphs

% dda
dda_vect = diff(da_vect)/dt;
dda_vect = [dda_vect(1) dda_vect];

% dds1
dds1_vect = diff(ds1_vect)/dt;
dds1_vect = [dds1_vect(1) dds1_vect];

% dds2
dds2_vect = diff(ds2_vect)/dt;
dds2_vect = [dds2_vect(1) dds2_vect];
```

```
%% Plot pose, velocity and acceleration of cartesian variables s1,s2,a

figure(1)

% Plot s1_vect
subplot(3,3,1)
plot(t, s1_vect)
xlabel('t [s]')
ylabel('s1 [m]')
title('Pos., vel., acc. of s1(t)')
```

```
% Plot s2_vect
subplot(3,3,2)
plot(t, s2_vect)
xlabel('t [s]')
ylabel('s2 [m]')
title('Pos., vel., acc. of s2(t)')

% Plot a_vect
subplot(3,3,3)
plot(t ,a_vect)
xlabel('t [s]')
ylabel('a [rad]')
title('Pos., vel., acc. of a(t)')

% Plot ds1_vect
subplot(3,3,4)
plot(t, ds1_vect);
xlabel('t [s]')
ylabel('ds1 [m/s]')

% Plot ds2_vect
subplot(3,3,5)
plot(t, ds2_vect);
xlabel('t [s]')
ylabel('ds2 [m/s]')

% Plot da_vect
subplot(3,3,6)
plot(t, da_vect);
xlabel('t [s]')
ylabel('da [rad/s]')

% Plot dds1_vect
subplot(3,3,7)
plot(t, dds1_vect);
xlabel('t [s]')
ylabel('dds1 [m/s^2]')

% Plot dds2_vect
subplot(3,3,8)
plot(t, dds2_vect);
xlabel('t [s]')
ylabel('dds2 [m/s^2]')
```

```
% Plot dda_vect
subplot(3,3,9)
plot(t, dda_vect);
xlabel('t [s]')
ylabel('dda [rad/s^2]')
```

```
%% Calculate pose, velocity and acceleration of joint variables q1, q2, q3 and plot the movie

% Initialize vectors
q1_vect = 1:length(t);
q2_vect = 1:length(t);
q3_vect = 1:length(t);
dq1_vect = 1:length(t);
dq2_vect = 1:length(t);
dq3_vect = 1:length(t);
ddq1_vect = 1:length(t);
ddq2_vect = 1:length(t);
ddq3_vect = 1:length(t);
detJ_vect = 1:length(t);

% Plot the "movie"
figure(100)
myVideo = VideoWriter('Moto Pinna'); %open video file
open(myVideo)

% j and step are used to speed up the movie
j=1;
step = floor(length(t)*0.06/simOut.SimulationMetadata.ModelInfo.StopTime);

for i = 1:length(t)

    % Calculate q1, q2, q3 (pose in joint space) for each instance of time
    % note: choose 4 as solution
    [q1_vect(i),q2_vect(i),q3_vect(i)] = pos_inv_kin(s1_vect(i),s2_vect(i),a_vect(i),d,4);

    % Plot Fin
    if j == i
        plot_fin(q1_vect(i),q2_vect(i),q3_vect(i),d,theta(i))

        axis equal
        xlim([0 0.5])
        ylim([-0.3 0.3])
        xlabel('x [m]')
        ylabel('y [m]')

        % Save that frame of plot in F
        F = getframe;
        writeVideo(myVideo, F);
        % Clear the previous frame for a correct movie
        cla
        j=j+step;
    end

    % Calculate dq1, dq2, dq3 (velocities in joint space) for each instance of time
    [dq1_vect(i),dq2_vect(i),dq3_vect(i)] = vel_inv_kin(ds1_vect(i),ds2_vect(i),da_vect(i),q1_vect(i),q2_vect(i),d);

    % Calculate ddq1, ddq2, ddq3 (accelerations in joint space) for each instance of time
    [ddq1_vect(i),ddq2_vect(i),ddq3_vect(i)] = acc_inv_kin(dds1_vect(i),dds2_vect(i),dda_vect(i),q1_vect(i),q2_vect(i),dq1_vect(i),dq2_vect(i),dq3_vect(i),d);

    % Calculate Jacobian for each instance of time
    J = jacobian_fin(q1_vect(i),q2_vect(i),d);

    % Calculate the derivative of Jacobian for each instance of time
    dJ = d_jacobian_fin(q1_vect(i),q2_vect(i),dq1_vect(i),dq2_vect(i),d);

    % Calculate  determinant of J
    detJ_vect(i) = det(J);

end

close(100)
close(myVideo)
```

```
%% Plot pose, velocity and acceleration of joint variables q1, q2, q3

figure(3)

% Plot q1_vect
subplot(3,3,1)
plot(t, q1_vect)
xlabel('t [s]')
ylabel('q1 [rad]')
title('Pos., vel., acc. of q1(t)')

% Plot q2_vect
subplot(3,3,2)
plot(t, q2_vect)
xlabel('t [s]')
ylabel('q2 [rad]')
title('Pos., vel., acc. of q2(t)')
```

```
% Plot q3_vect
subplot(3,3,3)
plot(t, q3_vect)
xlabel('t [s]')
ylabel('q3 [rad]')
title('Pos., vel., acc. of q3(t)')

% Plot dq1_vect
subplot(3,3,4)
plot(t, dq1_vect);
xlabel('t [s]')
ylabel('dq1 [rad/s]')

% Plot dq2_vect
subplot(3,3,5)
plot(t, dq2_vect);
xlabel('t [s]')
ylabel('dq2 [rad/s]')

% Plot dq3_vect
subplot(3,3,6)
plot(t, dq3_vect);
xlabel('t [s]')
ylabel('dq3 [rad/s]')

% Plot ddq1_vect
subplot(3,3,7)
plot(t, ddq1_vect);
xlabel('t [s]')
ylabel('ddq1 [rad/s^2]')

% Plot ddq2_vect
subplot(3,3,8)
plot(t, ddq2_vect);
xlabel('t [s]')
ylabel('ddq2 [rad/s^2]')

% Plot ddq3_vect
subplot(3,3,9)
plot(t, ddq3_vect);
xlabel('t [s]')
ylabel('ddq3 [rad/s^2]')
```

```
%% Plot the Jacobian

figure(4)
plot(t,detJ_vect,'r')
title("Determinant of the Jacobian")
```

App GUI code

```
classdef App < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public) ...

    properties (Access = private)
        Property % Description
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: LoadParametersButton
        function LoadParametersButtonPushed(app, event)

            table = readtable("data.xlsx");
            app.UITable.Data = table;
            app.UITable.ColumnName = table.Properties.VariableNames;

        end

        % Button pushed function: RunSimulationButton
        function RunSimulationButtonPushed(app, event)

            mdl = 'RoboticFish_NGC';
            open_system(mdl);

            % Save table in data
            data = app.UITable.Data;

            % Set parameters
            for i = 1:size(data, 2)
                paramName=app.UITable.ColumnName(i);
                paramValue = data{1, i}; % Nuovo valore del parametri
                if(i<2)
                    set_param('RoboticFish_NGC', string(paramName), string(paramValue))
                elseif(i<7)
                    set_param('RoboticFish_NGC/Target Position', string(paramName), string(paramValue));
                else
                    set_param('RoboticFish_NGC/Robotic-Fish Model & PropulsionSystem', string(paramName), string(paramValue));
                end
            end

            % Save model
            save_system(mdl);

            % Run simulation
            simOut=sim(mdl);

            % Squeeze variables
            simOut.X=squeeze(simOut.X);
            simOut.Y=squeeze(simOut.Y);
            simOut.Z=squeeze(simOut.Z);
            simOut.Roll=squeeze(simOut.Roll);
            simOut.Pitch=squeeze(simOut.Pitch);
            simOut.Yaw=squeeze(simOut.Yaw);
            simOut.u=squeeze(simOut.u);
            simOut.v=squeeze(simOut.v);
            simOut.w=squeeze(simOut.w);
            simOut.p=squeeze(simOut.p);
            simOut.q=squeeze(simOut.q);
            simOut.r=squeeze(simOut.r);

            % Save time
            time= simOut.tout;
            dt = simOut.SimulationMetadata.ModelInfo.SolverInfo.FixedStepSize;
            duration = simOut.SimulationMetadata.ModelInfo.StopTime;
```

```
% Save time
time= simOut.tout;
dt = simOut.SimulationMetadata.ModelInfo.SolverInfo.FixedStepSize;
duration = simOut.SimulationMetadata.ModelInfo.StopTime;

% Plot
data1=[simOut.X, simOut.Xr];
plot(app.UIAxes,time,data1);
legend(app.UIAxes,'X','Xr');

data2=[simOut.Y, simOut.Yr];
plot(app.UIAxes_2,time,data2);
legend(app.UIAxes_2,'Y','Yr');

data3=[simOut.Z, simOut.Zr];
plot(app.UIAxes_3,time,data3);
legend(app.UIAxes_3,'Z','Zr');

plot(app.UIAxes_4,time,simOut.Roll);

plot(app.UIAxes_5,time,simOut.Pitch);

plot(app.UIAxes_6,time,simOut.Yaw);

plot(app.UIAxes_7,time,simOut.u);

plot(app.UIAxes_8,time,simOut.v);

plot(app.UIAxes_9,time,simOut.w);

plot(app.UIAxes_10,time,simOut.p);

plot(app.UIAxes_11,time,simOut.q);

plot(app.UIAxes_12,time,simOut.r);

data4=[simOut.f_HT3, simOut.Thetastr_HT3, simOut.uHT1, simOut.uHT2, simOut.uVT1, simOut.uVT2, simOut.uVT3];
plot(app.UIAxes13 ,time, data4);
legend(app.UIAxes13,'f_HT3 [Hz]','Thetastr_HT3 [rad]','uHT1','uHT2','uVT1','uVT2','uVT3')

% Plot trajectory
data5=[simOut.Xr, simOut.X];
data6=[simOut.Yr, simOut.Y];
plot(app.UIAxes14,data5,data6);
WP_x = [0 10 15 20 20 10 10 0];
WP_y = [0 0 5 0 -10 -10 0 0];
hold(app.UIAxes14);
plot(app.UIAxes14,WP_x,WP_y,'go');
legend(app.UIAxes14,'Reference Trajectory','Effective Trajectory','Waypoints');
str = {'WP0','WP1','WP2','WP3','WP4','WP5','WP6','WP7'};
txt_x = [0.2 10.2 15.2 20.2 20.2 10.2 10.2 0.2];
txt_y = [0.2 0.2 5.2 0.2 -10.2 -10.2 -0.2 -0.2];
text(app.UIAxes14,txt_x,txt_y,str)

% Error computation WP0-WP1
WP0 = [0 0];
WP1 = [9.7 0];
timeWP0=1;
timeWP1=floor(10.33/dt);
Delta_t01 = timeWP1-timeWP0+1;
XrWP0WP1 = 1:Delta_t01;
YrWP0WP1 = 1:Delta_t01;
distance_WP0WP1 = 1:Delta_t01;
Lreal = 0;
Dreal = 1:Delta_t01;
Dref = 1:Delta_t01;
DWP = WP1-WP0;
Lref = sqrt(DWP(1).^2+DWP(2).^2);
psi = atan(DWP(2)/DWP(1));
```

```
206             for i=1:Delta_t01
207                 Dx = simOut.X(i+1)-simOut.X(i);
208                 Dy = simOut.Y(i+1)-simOut.Y(i);
209                 Dreal(i) = sqrt(Dx.^2+Dy.^2);
210                 Lreal = Lreal + Dreal(i);
211             end
212
213             Dref(1) = Dreal(1)*Lref/Lreal;
214             XrWP0WP1(1) = WP0(1)+cos(psi)*Dref(1);
215             YrWP0WP1(1) = WP0(2)+sin(psi)*Dref(1);
216             distance_WP1WP2(1) = sqrt((simOut.X(timeWP0)-XrWP0WP1(1))^2+(simOut.Y(timeWP0)-YrWP0WP1(1))^2);
217             for i=2:Delta_t01
218                 Dref(i) = Dreal(i)*Lref/Lreal;
219                 XrWP0WP1(i) = XrWP0WP1(i-1)+cos(psi)*Dref(i);
220                 YrWP0WP1(i) = YrWP0WP1(i-1)+sin(psi)*Dref(i);
221                 distance_WP1WP2(i) = sqrt((simOut.X(timeWP0+i)-XrWP0WP1(i))^2+(simOut.Y(timeWP0+i)-YrWP0WP1(i))^2);
222             end
223
224             app.AverageDistanceWP0WP1EditField.Value = mean(distance_WP1WP2);
225             app.StandardDeviationDistanceWP0WP1EditField.Value = sqrt(sum(distance_WP1WP2.^2)/Delta_t01);
226
227             % Plot WP0-WP1
228             data01x=[XrWP0WP1.', simOut.X(timeWP0:timeWP1)];
229             data01y=[YrWP0WP1.', simOut.Y(timeWP0:timeWP1)];
230             plot(app.UIAxes15,data01x,data01y);
231             hold(app.UIAxes15);
232             WPx=[WP0(1) WP1(1)];
233             WPy=[WP0(2) WP1(2)];
234             plot(app.UIAxes15,WPx,WPy,'go');
235             legend(app.UIAxes15,'Reference Trajectory','Effective Trajectory','Waypoints');
236             str01 = {'WP0','WP1'};
237             txt_x01 = [0 9.7];
238             txt_y01 = [0 0];
239             text(app.UIAxes15,txt_x01,txt_y01,str01)
240
241             % Error computation WP1-WP2
242             WP1 = [9.7 0];
243             WP2 = [14.77 4.7];
244             timeWP1=floor(10.33/dt);
245             timeWP2=floor(20.08/dt);
246             Delta_t12 = timeWP2-timeWP1+1;
247             XrWP1WP2 = 1:Delta_t12;
248             YrWP1WP2 = 1:Delta_t12;
249             distance_WP1WP2 = 1:Delta_t12;
250             Lreal = 0;
251             Dreal = 1:Delta_t12;
252             Dref = 1:Delta_t12;
253             DWP = WP2-WP1;
254             Lref = sqrt(DWP(1).^2+DWP(2).^2);
255             psi = atan(DWP(2)/DWP(1));
256
257             for i=1:Delta_t12
258                 Dx = simOut.X(i+1)-simOut.X(i);
259                 Dy = simOut.Y(i+1)-simOut.Y(i);
260                 Dreal(i) = sqrt(Dx.^2+Dy.^2);
261                 Lreal = Lreal + Dreal(i);
262             end
263
264             Dref(1) = Dreal(1)*Lref/Lreal;
265             XrWP1WP2(1) = WP1(1)+cos(psi)*Dref(1);
266             YrWP1WP2(1) = WP1(2)+sin(psi)*Dref(1);
267             distance_WP1WP2(1) = sqrt((simOut.X(timeWP1)-XrWP1WP2(1))^2+(simOut.Y(timeWP1)-YrWP1WP2(1))^2);
268             for i=2:Delta_t12
269                 Dref(i) = Dreal(i)*Lref/Lreal;
270                 XrWP1WP2(i) = XrWP1WP2(i-1)+cos(psi)*Dref(i);
271                 YrWP1WP2(i) = YrWP1WP2(i-1)+sin(psi)*Dref(i);
272                 distance_WP1WP2(i) = sqrt((simOut.X(timeWP1+i)-XrWP1WP2(i))^2+(simOut.Y(timeWP1+i)-YrWP1WP2(i))^2);
273             end
274
275             app.AverageDistanceWP1WP2EditField.Value = mean(distance_WP1WP2);
276             app.StandardDeviationDistanceWP1WP2EditField.Value = sqrt(sum(distance_WP1WP2.^2)/Delta_t12);
```

```
% Plot WP1-WP2
data12x=[XrWP1WP2.', simOut.X(timeWP1:timeWP2)];
data12y=[YrWP1WP2.', simOut.Y(timeWP1:timeWP2)];
plot(app.UIAxes16,data12x,data12y);
hold(app.UIAxes16);
WPx=[WP1(1) WP2(1)];
WPy=[WP1(2) WP2(2)];
plot(app.UIAxes16,WPx,WPy,'go');
legend(app.UIAxes16,'Reference Trajectory','Effective Trajectory','Waypoints');
str12 = {'WP1','WP2'};
txt_x12 = [9.7 14.77];
txt_y12 = [0 4.7];
text(app.UIAxes16,txt_x12,txt_y12,str12)

% Error computation WP2-WP3
WP2 = [14.77 4.7];
WP3 = [19.85 0.3];
timeWP2=floor(20.08/dt);
timeWP3=floor(34.20/dt);
Delta_t23 = timeWP3-timeWP2+1;
XrWP2WP3 = 1:Delta_t23;
YrWP2WP3 = 1:Delta_t23;
distance_WP2WP3 = 1:Delta_t23;
Lreal = 0;
Dreal = 1:Delta_t23;
Dref = 1:Delta_t23;
DWP = WP3-WP2;
Lref = sqrt(DWP(1).^2+DWP(2).^2);
psi = atan(DWP(2)/DWP(1));

for i=1:Delta_t23
    Dx = simOut.X(i+1)-simOut.X(i);
    Dy = simOut.Y(i+1)-simOut.Y(i);
    Dreal(i) = sqrt(Dx.^2+Dy.^2);
    Lreal = Lreal + Dreal(i);
end

Dref(1) = Dreal(1)*Lref/Lreal;
XrWP2WP3(1) = WP2(1)+cos(psi)*Dref(1);
YrWP2WP3(1) = WP2(2)+sin(psi)*Dref(1);
distance_WP2WP3(1) = sqrt((simOut.X(timeWP2)-XrWP2WP3(1))^2+(simOut.Y(timeWP2)-YrWP2WP3(1))^2);
for i=2:Delta_t23
    Dref(i) = Dreal(i)*Lref/Lreal;
    XrWP2WP3(i) = XrWP2WP3(i-1)+cos(psi)*Dref(i);
    YrWP2WP3(i) = YrWP2WP3(i-1)+sin(psi)*Dref(i);
    distance_WP2WP3(i) = sqrt((simOut.X(timeWP2+i)-XrWP2WP3(i))^2+(simOut.Y(timeWP2+i)-YrWP2WP3(i))^2);
end
app.AverageDistanceWP2WP3EditField.Value = mean(distance_WP2WP3);
app.StandardDeviationDistanceWP2WP3EditField.Value = sqrt(sum(distance_WP2WP3.^2)/Delta_t23);

% Plot WP2-WP3
data23x=[XrWP2WP3.', simOut.X(timeWP2:timeWP3)];
data23y=[YrWP2WP3.', simOut.Y(timeWP2:timeWP3)];
plot(app.UIAxes17,data23x,data23y);
hold(app.UIAxes17);
WPx=[WP2(1) WP3(1)];
WPy=[WP2(2) WP3(2)];
plot(app.UIAxes17,WPx,WPy,'go');
legend(app.UIAxes17,'Reference Trajectory','Effective Trajectory','Waypoints');
str23 = {'WP2','WP3'};
txt_x23 = [14.77 19.85];
txt_y23 = [4.7 0.3];
text(app.UIAxes17,txt_x23,txt_y23,str23)

% Error computation WP3-WP4
WP3 = [19.85 0.3];
WP4 = [20.04 -9.7];
timeWP3=floor(34.20/dt);
timeWP4=floor(44.37/dt);
Delta_t34 = timeWP4-timeWP3+1;
XrWP3WP4 = 1:Delta_t34;
YrWP3WP4 = 1:Delta_t34;
distance_WP3WP4 = 1:Delta_t34;
Lreal = 0;
Dreal = 1:Delta_t34;
Dref = 1:Delta_t34;
DWP = WP4-WP3;
Lref = sqrt(DWP(1).^2+DWP(2).^2);
psi = atan(DWP(2)/DWP(1));
```

```
            for i=1:Delta_t34
                Dx = simOut.X(i+1)-simOut.X(i);
                Dy = simOut.Y(i+1)-simOut.Y(i);
                Dreal(i) = sqrt(Dx.^2+Dy.^2);
                Lreal = Lreal + Dreal(i);
            end

            Dref(1) = Dreal(1)*Lref/Lreal;
            XrWP3WP4(1) = WP3(1)+cos(psi)*Dref(1);
            YrWP3WP4(1) = WP3(2)+sin(psi)*Dref(1);
            distance_WP3WP4(1) = sqrt((simOut.X(timeWP3)-XrWP3WP4(1))^2+(simOut.Y(timeWP3)-YrWP3WP4(1))^2);
            for i=2:Delta_t34
                Dref(i) = Dreal(i)*Lref/Lreal;
                XrWP3WP4(i) = XrWP3WP4(i-1)+cos(psi)*Dref(i);
                YrWP3WP4(i) = YrWP3WP4(i-1)+sin(psi)*Dref(i);
                distance_WP3WP4(i) = sqrt((simOut.X(timeWP3+i)-XrWP3WP4(i))^2+(simOut.Y(timeWP3+i)-YrWP3WP4(i))^2);
            end

            app.AverageDistanceWP3WP4EditField.Value = mean(distance_WP3WP4);
            app.StandardDeviationDistanceWP3WP4EditField.Value = sqrt(sum(distance_WP3WP4.^2)/Delta_t34);

            % Plot WP3-WP4
            data34x=[XrWP3WP4.', simOut.X(timeWP3:timeWP4)];
            data34y=[YrWP3WP4.', simOut.Y(timeWP3:timeWP4)];
            plot(app.UIAxes18,data34x,data34y);
            hold(app.UIAxes18);
            WPx=[WP3(1) WP4(1)];
            WPy=[WP3(2) WP4(2)];
            plot(app.UIAxes18,WPx,WPy,'go');
            legend(app.UIAxes18,'Reference Trajectory','Effective Trajectory','Waypoints');
            str34 = {'WP3','WP4'};
            txt_x34 = [19.85 20.04];
            txt_y34 = [0.3 -9.7];
            text(app.UIAxes18,txt_x34,txt_y34,str34)

            % Error computation WP4-WP5
            WP4 = [20.04 -9.7];
            WP5 = [10.30 -10.13];
            timeWP4=floor(44.37/dt);
            timeWP5=floor(59.37/dt);
            Delta_t45 = timeWP5-timeWP4+1;
            XrWP4WP5 = 1:Delta_t45;
            YrWP4WP5 = 1:Delta_t45;
            distance_WP4WP5 = 1:Delta_t45;
            Lreal = 0;
            Dreal = 1:Delta_t45;
            Dref = 1:Delta_t45;
            DWP = WP5-WP4;
            Lref = sqrt(DWP(1).^2+DWP(2).^2);
            psi = atan(DWP(2)/DWP(1));

            for i=1:Delta_t45
                Dx = simOut.X(i+1)-simOut.X(i);
                Dy = simOut.Y(i+1)-simOut.Y(i);
                Dreal(i) = sqrt(Dx.^2+Dy.^2);
                Lreal = Lreal + Dreal(i);
            end

            Dref(1) = Dreal(1)*Lref/Lreal;
            XrWP4WP5(1) = WP4(1)-cos(psi)*Dref(1);
            YrWP4WP5(1) = WP4(2)-sin(psi)*Dref(1);
            distance_WP4WP5(1) = sqrt((simOut.X(timeWP4)-XrWP4WP5(1))^2+(simOut.Y(timeWP4)-YrWP4WP5(1))^2);
            for i=2:Delta_t45
                Dref(i) = Dreal(i)*Lref/Lreal;
                XrWP4WP5(i) = XrWP4WP5(i-1)-cos(psi)*Dref(i);
                YrWP4WP5(i) = YrWP4WP5(i-1)-sin(psi)*Dref(i);
                distance_WP4WP5(i) = sqrt((simOut.X(timeWP4+i)-XrWP4WP5(i))^2+(simOut.Y(timeWP4+i)-YrWP4WP5(i))^2);
            end

            app.AverageDistanceWP4WP5EditField.Value = mean(distance_WP4WP5);
            app.StandardDeviationDistanceWP4WP5EditField.Value = sqrt(sum(distance_WP4WP5.^2)/Delta_t45);

            % Plot WP4-WP5
            data45x=[XrWP4WP5.', simOut.X(timeWP4:timeWP5)];
            data45y=[YrWP4WP5.', simOut.Y(timeWP4:timeWP5)];
            plot(app.UIAxes19,data45x,data45y);
            hold(app.UIAxes19);
            WPx=[WP4(1) WP5(1)];
            WPy=[WP4(2) WP5(2)];
            plot(app.UIAxes19,WPx,WPy,'go');
            legend(app.UIAxes19,'Reference Trajectory','Effective Trajectory','Waypoints');
            str45 = {'WP4','WP5'};
            txt_x45 = [20.04 10.30];
            txt_y45 = [-9.7 -10.13];
            text(app.UIAxes19,txt_x45,txt_y45,str45)
```

```
% Error computation WP5-WP6
WP5 = [10.30 -10.13];
WP6 = [9.96 -0.30];
timeWP5=floor(59.37/dt);
timeWP6=floor(73.09/dt);
Delta_t56 = timeWP6-timeWP5+1;
XrWP5WP6 = 1:Delta_t56;
YrWP5WP6 = 1:Delta_t56;
distance_WP5WP6 = 1:Delta_t56;
Lreal = 0;
Dreal = 1:Delta_t56;
Dref = 1:Delta_t56;
DWP = WP6-WP5;
Lref = sqrt(DWP(1).^2+DWP(2).^2);
psi = atan(DWP(2)/DWP(1));

for i=1:Delta_t56
    Dx = simOut.X(i+1)-simOut.X(i);
    Dy = simOut.Y(i+1)-simOut.Y(i);
    Dreal(i) = sqrt(Dx.^2+Dy.^2);
    Lreal = Lreal + Dreal(i);
end

Dref(1) = Dreal(1)*Lref/Lreal;
XrWP5WP6(1) = WP5(1)-cos(psi)*Dref(1);
YrWP5WP6(1) = WP5(2)-sin(psi)*Dref(1);
distance_WP5WP6(1) = sqrt((simOut.X(timeWP5)-XrWP5WP6(1))^2+(simOut.Y(timeWP5)-YrWP5WP6(1))^2);
for i=2:Delta_t56
    Dref(i) = Dreal(i)*Lref/Lreal;
    XrWP5WP6(i) = XrWP5WP6(i-1)-cos(psi)*Dref(i);
    YrWP5WP6(i) = YrWP5WP6(i-1)-sin(psi)*Dref(i);
    distance_WP5WP6(i) = sqrt((simOut.X(timeWP5+i)-XrWP5WP6(i))^2+(simOut.Y(timeWP5+i)-YrWP5WP6(i))^2);
end

app.AverageDistanceWP5WP6EditField.Value = mean(distance_WP5WP6);
app.StandardDeviationDistanceWP5WP6EditField.Value = sqrt(sum(distance_WP5WP6.^2)/Delta_t56);

% Plot WP5-WP6
data56x=[XrWP5WP6.', simOut.X(timeWP5:timeWP6)];
data56y=[YrWP5WP6.', simOut.Y(timeWP5:timeWP6)];
plot(app.UIAxes20,data56x,data56y);
hold(app.UIAxes20);
WPx=[WP5(1) WP6(1)];
WPy=[WP5(2) WP6(2)];
plot(app.UIAxes20,WPx,WPy,'go');
legend(app.UIAxes20,'Reference Trajectory','Effective Trajectory','Waypoints');
str56 = {'WP5','WP6'};
txt_x56 = [10.30 9.96];
txt_y56 = [-10.13 -0.30];
text(app.UIAxes20,txt_x56,txt_y56,str56)

% Error computation WP6-WP7
WP6 = [9.96 -0.30];
WP7 = [0.30 0.01];
timeWP6=floor(73.09/dt);
timeWP7=floor(88.39/dt);
Delta_t67 = timeWP7-timeWP6+1;
XrWP6WP7 = 1:Delta_t67;
YrWP6WP7 = 1:Delta_t67;
distance_WP6WP7 = 1:Delta_t67;
Lreal = 0;
Dreal = 1:Delta_t67;
Dref = 1:Delta_t67;
DWP = WP7-WP6;
Lref = sqrt(DWP(1).^2+DWP(2).^2);
psi = atan(DWP(2)/DWP(1));

for i=1:Delta_t67
    Dx = simOut.X(i+1)-simOut.X(i);
    Dy = simOut.Y(i+1)-simOut.Y(i);
    Dreal(i) = sqrt(Dx.^2+Dy.^2);
    Lreal = Lreal + Dreal(i);
end
```

```
            Dref(1) = Dreal(1)*Lref/Lreal;
            XrWP6WP7(1) = WP6(1)-cos(psi)*Dref(1);
            YrWP6WP7(1) = WP6(2)-sin(psi)*Dref(1);
            distance_WP6WP7(1) = sqrt((simOut.X(timeWP6)-XrWP6WP7(1))^2+(simOut.Y(timeWP6)-YrWP6WP7(1))^2);
            for i=2:Delta_t67
                Dref(i) = Dreal(i)*Lref/Lreal;
                XrWP6WP7(i) = XrWP6WP7(i-1)-cos(psi)*Dref(i);
                YrWP6WP7(i) = YrWP6WP7(i-1)-sin(psi)*Dref(i);
                distance_WP6WP7(i) = sqrt((simOut.X(timeWP6+i)-XrWP6WP7(i))^2+(simOut.Y(timeWP6+i)-YrWP6WP7(i))^2);
            end

            app.AverageDistanceWP6WP7EditField.Value = mean(distance_WP6WP7);
            app.StandardDeviationDistanceWP6WP7EditField.Value = sqrt(sum(distance_WP6WP7.^2)/Delta_t67);

            % Plot WP6-WP7
            data67x=[XrWP6WP7.', simOut.X(timeWP6:timeWP7)];
            data67y=[YrWP6WP7.', simOut.Y(timeWP6:timeWP7)];
            plot(app.UIAxes21,data67x,data67y);
            hold(app.UIAxes21);
            WPx=[WP6(1) WP7(1)];
            WPy=[WP6(2) WP7(2)];
            plot(app.UIAxes21,WPx,WPy,'go');
            legend(app.UIAxes21,'Reference Trajectory','Effective Trajectory','Waypoints');
            str67 = {'WP6','WP7'};
            txt_x67 = [9.96 0.30];
            txt_y67 = [-0.30 0.01];
            text(app.UIAxes21,txt_x67,txt_y67,str67)
```

```
% Plot fin kinematics
cd 'Fin Kinematics'
main
cd ..
```

```


nent initialization
 (Access = private)

reate UIFigure and components
ction createComponents(app)

 % Create UIFigure and hide until all components are created
 app.UIFigure = uifigure('Visible', 'off');
 app.UIFigure.Position = [100 100 820 522];
 app.UIFigure.Name = 'MATLAB App';

 % Create TabGroup
 app.TabGroup = uitabgroup(app.UIFigure);
 app.TabGroup.Position = [1 1 923 609];

 % Create MainTab
 app.MainTab = uitab(app.TabGroup);
 app.MainTab.Title = 'Main';

 % Create UITable
 app.UITable = uitable(app.MainTab);
 app.UITable.ColumnName = {'Column 1'; 'Column 2'; 'Column 3'; 'Column 4'};
 app.UITable.RowName = {};
 app.UITable.Position = [85 391 754 75];
```

```
% Create LoadParametersButton
app.LoadParametersButton = uibutton(app.MainTab, 'push');
app.LoadParametersButton.ButtonPushedFcn = createCallbackFcn(app, @LoadParametersButtonPushed, true);
app.LoadParametersButton.IconAlignment = 'center';
app.LoadParametersButton.Position = [408 293 107 23];
app.LoadParametersButton.Text = 'Load Parameters';

% Create ParametersLabel
app.ParametersLabel = uilabel(app.MainTab);
app.ParametersLabel.HorizontalAlignment = 'center';
app.ParametersLabel.Position = [85 500 67 22];
app.ParametersLabel.Text = 'Parameters';

% Create RunSimulationButton
app.RunSimulationButton = uibutton(app.MainTab, 'push');
app.RunSimulationButton.ButtonPushedFcn = createCallbackFcn(app, @RunSimulationButtonPushed, true);
app.RunSimulationButton.IconAlignment = 'center';
app.RunSimulationButton.Position = [412 210 100 23];
app.RunSimulationButton.Text = 'Run Simulation';

% Create PositionTab
app.PositionTab = uitab(app.TabGroup);
app.PositionTab.Title = 'Position';

% Create UIAxes_3
app.UIAxes_3 = uiaxes(app.PositionTab);
title(app.UIAxes_3, 'Z ')
xlabel(app.UIAxes_3, 'Time [s]')
ylabel(app.UIAxes_3, 'Z [m]')
zlabel(app.UIAxes_3, 'Z')
app.UIAxes_3.XGrid = 'on';
app.UIAxes_3.YGrid = 'on';
app.UIAxes_3.ZGrid = 'on';
app.UIAxes_3.Position = [269 7 385 263];
```

```
% Create UIAxes_2
app.UIAxes_2 = uiaxes(app.PositionTab);
title(app.UIAxes_2, 'Y')
xlabel(app.UIAxes_2, 'Time [s]')
ylabel(app.UIAxes_2, 'Y [m]')
zlabel(app.UIAxes_2, 'Z')
app.UIAxes_2.XGrid = 'on';
app.UIAxes_2.YGrid = 'on';
app.UIAxes_2.ZGrid = 'on';
app.UIAxes_2.Position = [480 285 385 270];

% Create UIAxes
app.UIAxes = uiaxes(app.PositionTab);
title(app.UIAxes, 'X')
xlabel(app.UIAxes, 'Time [s]')
ylabel(app.UIAxes, 'X [m]')
zlabel(app.UIAxes, 'Z')
app.UIAxes.XGrid = 'on';
app.UIAxes.YGrid = 'on';
app.UIAxes.ZGrid = 'on';
app.UIAxes.Position = [45 290 385 265];

% Create OrientationTab
app.OrientationTab = uitab(app.TabGroup);
app.OrientationTab.Title = 'Orientation';

% Create UIAxes_6
app.UIAxes_6 = uiaxes(app.OrientationTab);
title(app.UIAxes_6, 'Yaw')
xlabel(app.UIAxes_6, 'Time [s]')
ylabel(app.UIAxes_6, 'Yaw [rad]')
zlabel(app.UIAxes_6, 'Z')
app.UIAxes_6.XGrid = 'on';
app.UIAxes_6.YGrid = 'on';
app.UIAxes_6.ZGrid = 'on';
```

```
app.UIAxes_6.Position = [269 7 385 263];

% Create UIAxes_5
app.UIAxes_5 = uiaxes(app.OrientationTab);
title(app.UIAxes_5, 'Pitch')
xlabel(app.UIAxes_5, 'Time [s]')
ylabel(app.UIAxes_5, 'Pitch [rad]')
zlabel(app.UIAxes_5, 'Z')
app.UIAxes_5.XGrid = 'on';
app.UIAxes_5.YGrid = 'on';
app.UIAxes_5.ZGrid = 'on';
app.UIAxes_5.Position = [480 285 385 270];

% Create UIAxes_4
app.UIAxes_4 = uiaxes(app.OrientationTab);
title(app.UIAxes_4, 'Roll')
xlabel(app.UIAxes_4, 'Time [s]')
ylabel(app.UIAxes_4, 'Roll [rad]')
zlabel(app.UIAxes_4, 'Z')
app.UIAxes_4.XGrid = 'on';
app.UIAxes_4.YGrid = 'on';
app.UIAxes_4.ZGrid = 'on';
app.UIAxes_4.Position = [45 290 385 265];

% Create LinearVelocityTab
app.LinearVelocityTab = uitab(app.TabGroup);
app.LinearVelocityTab.Title = 'Linear Velocity';

% Create UIAxes_9
app.UIAxes_9 = uiaxes(app.LinearVelocityTab);
title(app.UIAxes_9, 'w')
xlabel(app.UIAxes_9, 'Time [s]')
ylabel(app.UIAxes_9, 'w [m/s]')
zlabel(app.UIAxes_9, 'Z')
app.UIAxes_9.XGrid = 'on';
app.UIAxes_9.YGrid = 'on';
app.UIAxes_9.ZGrid = 'on';
app.UIAxes_9.Position = [269 7 385 263];

% Create UIAxes_8
app.UIAxes_8 = uiaxes(app.LinearVelocityTab);
title(app.UIAxes_8, 'v')
xlabel(app.UIAxes_8, 'Time [s]')
ylabel(app.UIAxes_8, 'v [m/s]')
zlabel(app.UIAxes_8, 'Z')
app.UIAxes_8.XGrid = 'on';
app.UIAxes_8.YGrid = 'on';
app.UIAxes_8.ZGrid = 'on';
app.UIAxes_8.Position = [480 285 385 270];

% Create UIAxes_7
app.UIAxes_7 = uiaxes(app.LinearVelocityTab);
title(app.UIAxes_7, 'u')
xlabel(app.UIAxes_7, 'Time [s]')
ylabel(app.UIAxes_7, 'u [m/s]')
zlabel(app.UIAxes_7, 'Z')
app.UIAxes_7.XGrid = 'on';
app.UIAxes_7.YGrid = 'on';
app.UIAxes_7.ZGrid = 'on';
app.UIAxes_7.Position = [45 290 385 265];

% Create AngularVelocityTab
app.AngularVelocityTab = uitab(app.TabGroup);
app.AngularVelocityTab.Title = 'Angular Velocity';

% Create UIAxes_12
app.UIAxes_12 = uiaxes(app.AngularVelocityTab);
title(app.UIAxes_12, 'r')
xlabel(app.UIAxes_12, 'Time [s]')
ylabel(app.UIAxes_12, 'r [rad/s]')
```

```
zlabel(app.UIAxes_12, 'Z')
app.UIAxes_12.XGrid = 'on';
app.UIAxes_12.YGrid = 'on';
app.UIAxes_12.ZGrid = 'on';
app.UIAxes_12.Position = [269 7 385 263];

% Create UIAxes_11
app.UIAxes_11 = uiaxes(app.AngularVelocityTab);
title(app.UIAxes_11, 'q')
xlabel(app.UIAxes_11, 'Time [s]')
ylabel(app.UIAxes_11, 'q [rad/s]')
zlabel(app.UIAxes_11, 'Z')
app.UIAxes_11.XGrid = 'on';
app.UIAxes_11.YGrid = 'on';
app.UIAxes_11.ZGrid = 'on';
app.UIAxes_11.Position = [480 285 385 270];

% Create UIAxes_10
app.UIAxes_10 = uiaxes(app.AngularVelocityTab);
title(app.UIAxes_10, 'p')
xlabel(app.UIAxes_10, 'Time [s]')
ylabel(app.UIAxes_10, 'p [rad/s]')
zlabel(app.UIAxes_10, 'Z')
app.UIAxes_10.XGrid = 'on';
app.UIAxes_10.YGrid = 'on';
app.UIAxes_10.ZGrid = 'on';
app.UIAxes_10.Position = [45 290 385 265];

% Create AllActuatorsTab
app.AllActuatorsTab = uitab(app.TabGroup);
app.AllActuatorsTab.Title = 'All Actuators';
```

```
% Create AllActuatorsTab
app.AllActuatorsTab = uitab(app.TabGroup);
app.AllActuatorsTab.Title = 'All Actuators';

% Create UIAxes13
app.UIAxes13 = uiaxes(app.AllActuatorsTab);
title(app.UIAxes13, 'All Actuators')
xlabel(app.UIAxes13, 'Time [s]')
zlabel(app.UIAxes13, 'Z')
app.UIAxes13.XGrid = 'on';
app.UIAxes13.YGrid = 'on';
app.UIAxes13.ZGrid = 'on';
app.UIAxes13.Position = [89 76 746 457];

% Create TrajectoryTab
app.TrajectoryTab = uitab(app.TabGroup);
app.TrajectoryTab.Title = 'Trajectory';

% Create UIAxes14
app.UIAxes14 = uiaxes(app.TrajectoryTab);
title(app.UIAxes14, 'Effective trajectory with respect to reference trajectory')
xlabel(app.UIAxes14, 'X')
ylabel(app.UIAxes14, 'Y')
zlabel(app.UIAxes14, 'Z')
app.UIAxes14.XGrid = 'on';
app.UIAxes14.YGrid = 'on';
app.UIAxes14.ZGrid = 'on';
app.UIAxes14.Position = [117 54 690 479];

% Create ErrorWP0WP1Tab
app.ErrorWP0WP1Tab = uitab(app.TabGroup);
app.ErrorWP0WP1Tab.Title = 'Error WP0-WP1';
```

```
% Create UIAxes15
app.UIAxes15 = uiaxes(app.ErrorWP0WP1Tab);
title(app.UIAxes15, 'Effective trajectory w.r.t. reference trajectory')
xlabel(app.UIAxes15, 'X')
ylabel(app.UIAxes15, 'Y')
zlabel(app.UIAxes15, 'Z')
app.UIAxes15.Position = [269 54 596 493];

% Create AverageDistanceWP0WP1EditFieldLabel
app.AverageDistanceWP0WP1EditFieldLabel = uilabel(app.ErrorWP0WP1Tab);
app.AverageDistanceWP0WP1EditFieldLabel.HorizontalAlignment = 'right';
app.AverageDistanceWP0WP1EditFieldLabel.Position = [26 525 165 22];
app.AverageDistanceWP0WP1EditFieldLabel.Text = '  Average Distance WP0-WP1';

% Create AverageDistanceWP0WP1EditField
app.AverageDistanceWP0WP1EditField = uieditfield(app.ErrorWP0WP1Tab, 'numeric');
app.AverageDistanceWP0WP1EditField.Editable = 'off';
app.AverageDistanceWP0WP1EditField.Position = [32 500 169 22];

% Create StandardDeviationDistanceWP0WP1EditFieldLabel
app.StandardDeviationDistanceWP0WP1EditFieldLabel = uilabel(app.ErrorWP0WP1Tab);
app.StandardDeviationDistanceWP0WP1EditFieldLabel.HorizontalAlignment = 'right';
app.StandardDeviationDistanceWP0WP1EditFieldLabel.Position = [28 455 217 22];
app.StandardDeviationDistanceWP0WP1EditFieldLabel.Text = 'Standard Deviation Distance WP0-WP1';

% Create StandardDeviationDistanceWP0WP1EditField
app.StandardDeviationDistanceWP0WP1EditField = uieditfield(app.ErrorWP0WP1Tab, 'numeric');
app.StandardDeviationDistanceWP0WP1EditField.Editable = 'off';
app.StandardDeviationDistanceWP0WP1EditField.Position = [35 434 167 22];

% Show the figure after all components are created
app.UIFigure.Visible = 'on';
```

```
reation and deletion
 (Access = public)

onstruct app
ction app = App

 % Create UIFigure and components
 createComponents(app)

 % Register the app with App Designer
 registerApp(app, app.UIFigure)

 if nargout == 0
     clear app
 end


ode that executes before app deletion
ction delete(app)

 % Delete UIFigure when app is deleted
 delete(app.UIFigure)
```

Appendix 3 – Evaluation Questionnaire

TC4.1: App designer end-users

TP4.1.1 Experience in the use of the MATLAB application on its opening.

Evaluation:

1 (KO)	2	3	4	5 (OK)

TP4.1.1 Problems encountered in the procedure.

Enter in the box problems encountered in terms of app crashes, opening problems, manually entering parameters, clicking buttons on the panel.

Enter your problems…

TC4.1: App designer end-users

TP4.1.2. Experience in the use of the MATLAB application by starting the simulation with default parameters.

Evaluation:

1 (KO)	2	3	4	5 (OK)

TP4.1.2 Problems encountered in the procedure.

Enter in the box problems encountered in terms of app crashes, opening problems, manually entering parameters, clicking buttons on the panel.

Enter your problems…

TC4.1: App designer end-users

TP4.1.3. Experience in the use of the MATLAB application by starting the simulation with the parameters entered by the user.

Evaluation:

1 (KO)	2	3	4	5 (OK)

TP4.1.3 Problems encountered in the procedure.

Enter in the box problems encountered in terms of app crashes, opening problems, manually entering parameters, clicking buttons on the panel.

Enter your problems…