Projet KARSTEX

ROB 3A

Février 2024

Contents

1 Introduction

1.1 Abstract

Over a period of 60 time slots, third-year engineering students specializing in autonomous robotics at the National School of Applied Sciences and Technologies (ENSTA Bretagne) developed a solution for exploring karstic environments, underground cavities formed by the dissolution of limestone rock by slightly acidic water. This report provides a detailed overview of the chosen technical solution, the sensors used, and the mechanical, electronic, and software architecture adopted for the "Karstex" robot.

1.2 Context

Understanding karstic caves is crucial for regions like southern France, including the city of Nîmes, where these caves hold significant freshwater reserves. However, traditional exploration methods, such as drilling or sending divers, are both expensive and risky due to the complex underground terrain and dangers like stalactites, stalagmites, and narrow passages. Therefore, there's a pressing need to accurately map these caves to guide safe extraction efforts efficiently, minimizing costs and risks.

To address this challenge, an innovative initiative is underway to develop an autonomous robotic solution capable of navigating these environments and mapping caves without endangering human divers. By harnessing advanced robotics and sensor technologies, this project aims to create a versatile robot that can maneuver through the caves, mapping their features, and gathering crucial data without putting human lives at risk.

This approach not only improves safety but also ensures comprehensive exploration and mapping of karstic caves, thereby facilitating sustainable water resource management in the region. By eliminating the need for human divers to venture into hazardous conditions, this innovative approach enhances safety while promoting efficient and effective exploration of these valuable natural resources.

1.3 Organisation

You can see in Figure 1, the organisation we have chosen during this project.

RESPONSABLES	GUIDAGE VISUEL	CONTRÔLE	GUIDAGE	Versionnement (GIT)
Titouan Belier	Hugo Hofmann	Louis-Nam Gros	Louis Roullier	Martin Galliot
Théo Massa	Kevin Ren	Mathieu Pitaud	Joachim Zafrana	Guillaume Garde
Mathys Séry	Victor Bellot	Tristan Lefloch	Virgile Pelle	
	Gabriel Betton			
	Bastian Garagnon	Etienne Roussel	Louis Ravain	

ÉLECTRONIQUE	VISION	ECHOSONDEURS	IMU - PRESSION	MECANIQUE
Taddeo Guérin	Catherine Rizk	Martin Pilon	Gabriel Betton	Mathys Séry
	Guillaume Garde	Emilie Ledoussal	Catherine Rizk	Rania Ziane
	Adam Goux-Gâteaux	Marguerite Miallier	Victor Bellot	Marguerite Miallier
	Gabriel Betton			Emilie Ledoussal
	Victor Bellot			Etienne Roussel
				Martin Pilon
				Mathieu Pitaud

SIMULATION (gazebo)	EQUATIONS D'ETATS - Hydrodynamique	KALMAN	BALLAST
Théo Massa	Gwendal Crequer	Tristan Lefloch	Hippolyte Leroy
Clara Gondot	Johan Gonzalez	Louis Ravain	Samy Bourzoufi
Apolline De Vaulchier	Martin Pilon	Louis Gillard	Jules Le Gouallec
Ludovic Mustière	Léo Bernard	Titouan BELIER	
Etienne Roussel			

Figure 1: Project's organisation

2 Mechanical architecture

Students : Mathys SÉRY, Rania ZIANE, Marguerite MIALLIER, Emilie LEDOUSSAL, Mathieu PITAUD, Martin PILON, Étienne ROUSSEL

2.1 Preliminary work

Before starting any mechanical design, it is essential to understand the environment in which our robot operates, as well as its constraints and functions: what should our robot absolutely contain? What should it do and how should it do it?

Firstly, consider the environment in which the robot moves. It must operate:

- In an underwater environment. This implies taking waterproofing into account in the design. Waterproofing will be ensured by Blue Robotics and Blue Trail materials (for connectors). The Karstix must also be able to remain submerged for an extended period without being damaged or weakened by water and potential corrosion. Therefore, the use of corrosion-resistant materials such as plastic should be favored.

- In a restricted and irregular environment. Karsts have rocky walls with randomly varying sections throughout their length. Consequently, a robot that takes up the least amount of space possible, or at least the narrowest possible robot, should be considered. Additionally, rocks can snag cables if they are not protected and/or exposed around the robot. Therefore, an architecture should be planned to also store and protect data, command, or power cables.

Next, let's address the constraints and functions of the system:

- The Karstix must move "agilely" within its environment. To achieve this, two concepts have been chosen: the addition of ballast and a central and controllable articulation. These concepts form the basis of our vehicle and are combined with rear propulsion generated by a T200 motor from Blue Robotics. The central articulation will be actuated by a waterproof servo motor.

- The goal of Kartix is to autonomously explore unknown karsts. In its exploration equipment, there are Blue Robotics echosounders and a RealSense camera for stereovision. Since RealSense is not waterproof, a way must be found to install it in the robot while allowing it to see what is happening in front. As karsts are dark, bright spots (from Blue Robotics) must also be added to the front of the robot to illuminate the path. Regarding echosounders, they must also be placed at the front to detect potential collisions. Each echosounder has an emission cone that must be taken into account for their integration into the system.

Note: Initially, we were planning to use a mechanical scanning sonar, a Ping 360 from Blue Robotics. However, its scanning speed is too slow for our intended use. It was therefore agreed with the "echosounder" group that we would use 3 echo sounders arranged regularly (spaced 120° apart) to reproduce this scanning effect.

We have a preliminary architecture taking shape: a underwater robot consisting of two distinct parts, which we will call « Corps Avant » and « Corps Arrière ». These two parts are connected by an intermediate assembly allowing the robot to bend at its center using a servo motor. At the front of the robot (on the "Front Body"/ « Corps Avant »), our sensors (echosounders and camera inside) should be placed, and attached to the "Rear Body"/« Corps Arrière », our propulsion should be located. For stability reasons, it has been agreed with the Ballast group that we would have 2 ballasts: 1 for each body, as far apart as possible and in the axis of the robot.

From this initial description of the architecture, we have been able to start proposing concepts

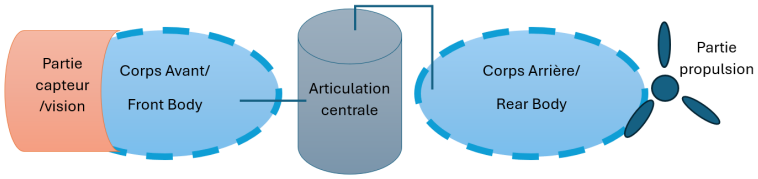


Figure 2: Minimalist Architecture of Karstix

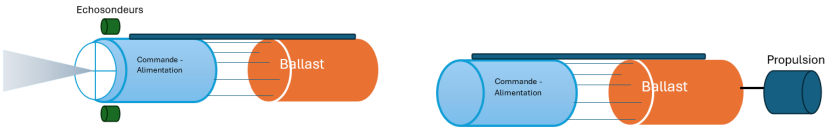
for parts and assembly for the 3D design of the robot, with a view to potential real-world manufacturing.

2.2 Robot Skeleton

The robot is composed of two main "bodies" to which lights, sensors, motors, etc., are added. Each "body" consists of a ballast and a cylinder containing a part of the "control & power" system. Between these two parts, there is a small space where cables connect them for control. To harmonize the structure and ensure waterproofing, we have opted for the use of Blue Robotics tubes.

There are various tube formats (multiple "width-length" combinations possible). For the "control & power" part, in collaboration with the "electronics" group, we have decided to use 4-inch tubes (approximately 10-11 cm in diameter) and 30 cm in length. This "4" - 300 mm" format allows us to accommodate enough equipment while restricting the overall size of the robot: the Karstix must carry all the necessary equipment while maintaining a compact profile (important for navigating in karsts).

Concerning the ballasts, their size and evolution have been subject to regular discussions between our group and the group in charge of their development. The goal was to follow their progress for optimal integration into the system. Changes in ballast architecture may result in design changes for the overall robot architecture, especially for inter-cylinder attachment parts.



(a) Composition of the Front Body (b) Composition of the Rear Body

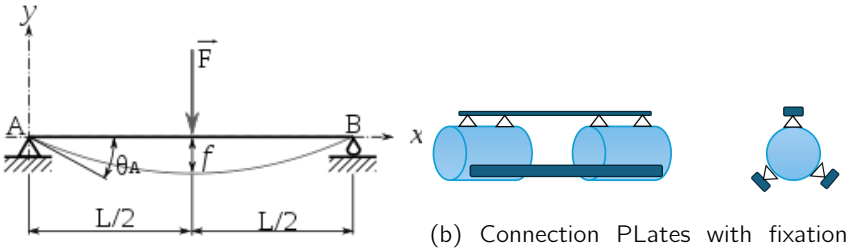
Figure 3: Organisation of each Body

2.3 Connection/Coherence within the Main Bodies

2.3.1 Connection Plates

The two cylinders of each body are held together by long PVC plates. Three plates surround each body and run the entire length of the body.

Due to their length, flexural phenomena may occur at the center of each plate if they are only fixed at the ends (and if they are too thin). For this reason, and to ensure a good attachment to the cylinders, we have 4 fixation points for each plate (2 fixation points per cylinder).



(a) Flexion (b) Connection PLates with fixation points

Figure 4: Position of the connection plates with fixation points

2.3.2 Cylinder Attachments

The design of fixation parts, capable of fitting on Blue Robotics cylinders and serving as a support for additional components on the robot, was based on existing pieces (réf. :Watertight Enclosure Clamps).

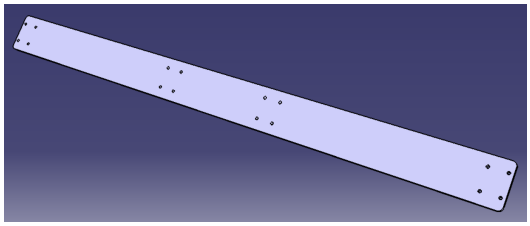


Figure 5: Connection Plate

The idea was to start from this fixation in two "half-pieces" that we shape to serve as support for the 3 plates as well as lights and echo-sounders.

- First Proposal: A robust-looking piece capable of accommodating 4 fixation plates. It focuses only on the attachment of the cylinders to each other (no consideration for echo-sounders and lights).
  This piece served as a "trial run": corrections and remarks were provided by Lab-STIC resource center members regarding its size, machinability, and how it attaches to the cylinder.

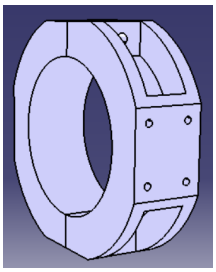
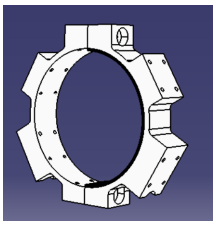
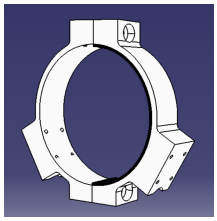


Figure 6: First Proposal

- Second Proposal: Taking into account the feedback, we arrived at 2 pieces (one being a derivative of the other). We reduced the number of plates to 3, the attachment to the cylinder is done by a piece of rubber glued under the part, and we have a version that can accommodate additional supports.



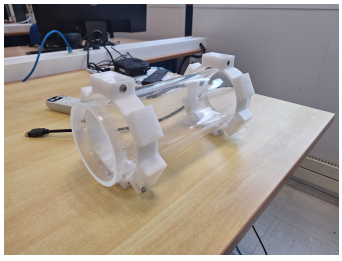
(a) With additional supports



(b) Without additional supports

Figure 7: Second Proposal

Four pieces (two rings) were machined from POM-C to verify the real-world result:



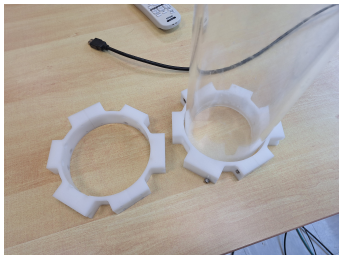


Figure 8: Real fixations

2.4 Servomotor attachment

To articulate the rear and front parts of the robot, we have opted for a servo motor allowing rotation around the z-axis.

The issue lies in the fragility of the servo motor due to the forces and hydrodynamic torques generated on the system. Therefore, we have considered a rigid-elastic linkage system. The chosen solution involves using two springs coupled to a connecting rod to ensure the rotation of the two parts of the robot, and a third spring beneath the servo motor to provide flexibility and balance to the system.

Below is a detailed plan of the chosen system.

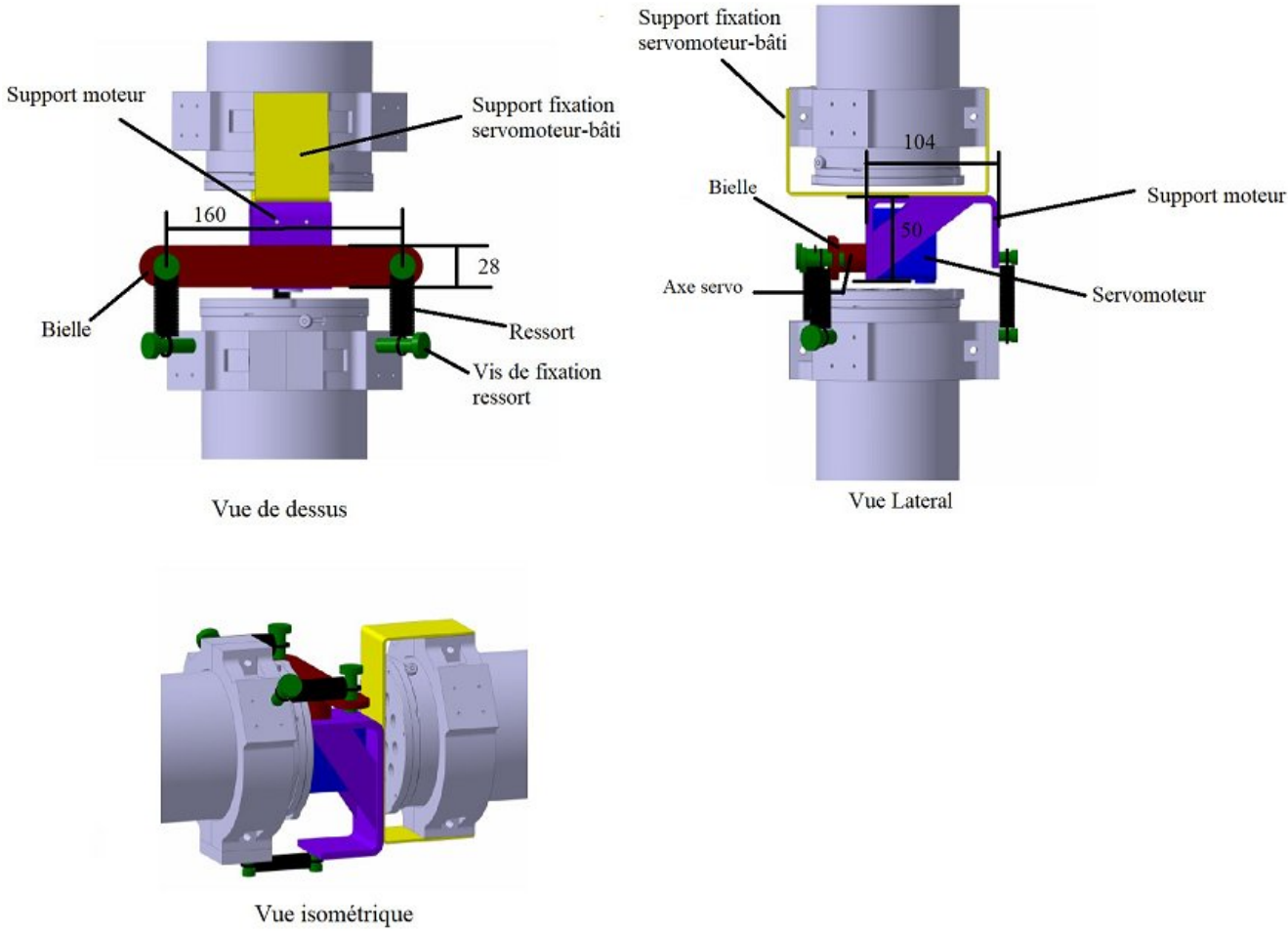


Figure 9: servomotor attachment system plan

2.5 T200 thruster fixings

To attach the T200 motor to the overall structure : First, we remove the conical cap from the T200 thruster to be able to attach the white plate instead. Then, we screw three threaded rods into the rear part. Finally, we fix the plate onto the three rods using bolts. The final assembly is shown in the figures below.

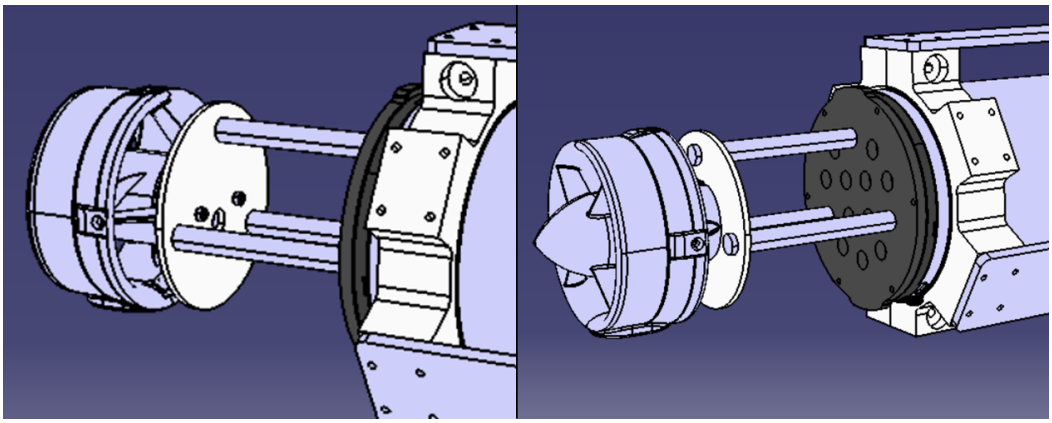


Figure 10: T200 thruster fixings on Computer aided design software

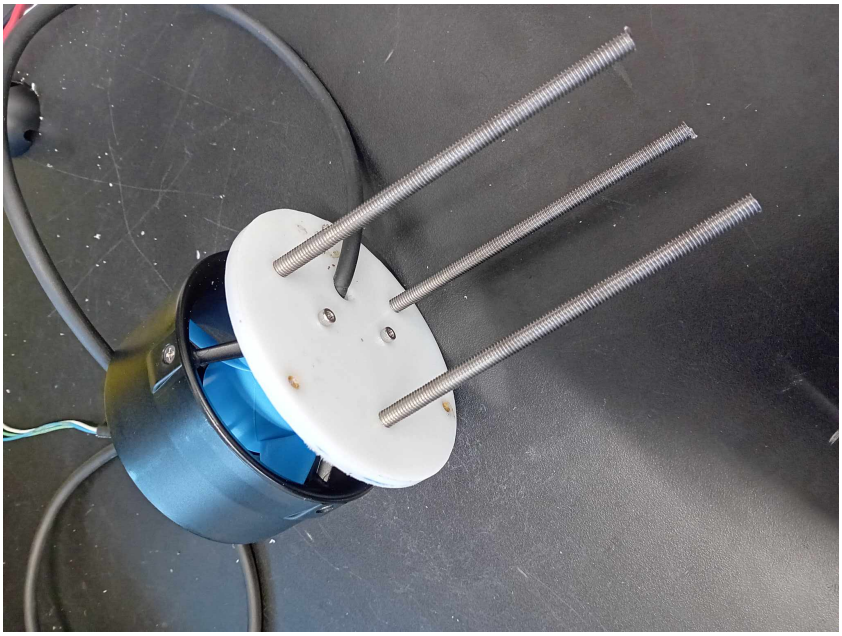


Figure 11: T200 thruster fixings

One of the problems with this design is the potential obstruction of water flow caused by the flat piece attached to the thruster. As a result, an alternative, more hydrodynamic design was considered. However, this proved too difficult to manufacture and consumed excessive quantities of materials.

2.6 Echo sounders fixings

To design these parts, we worked with the "Echo sounders" team.

Echo sounders are attached to the body by a combination of three different parts that interlock with each other :

- the first one is screwed to the body
- the second one glides in the first one. It can be oriented in two different ways depending on whether we want the sonar to be at 45° or towards the front of the robot
- the sonar is screwed in the third one, that glides in the second one.

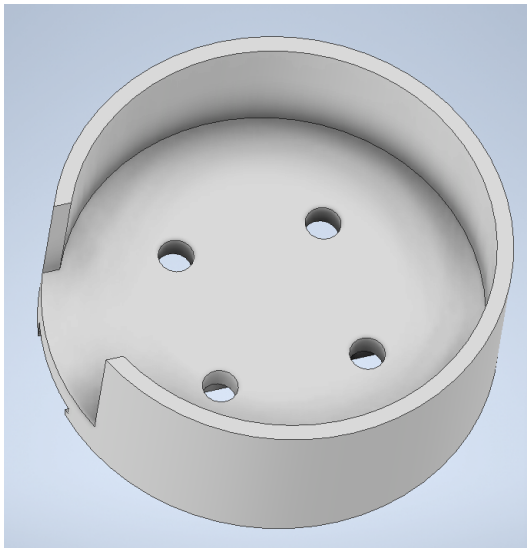


Figure 12: Echo sounders support

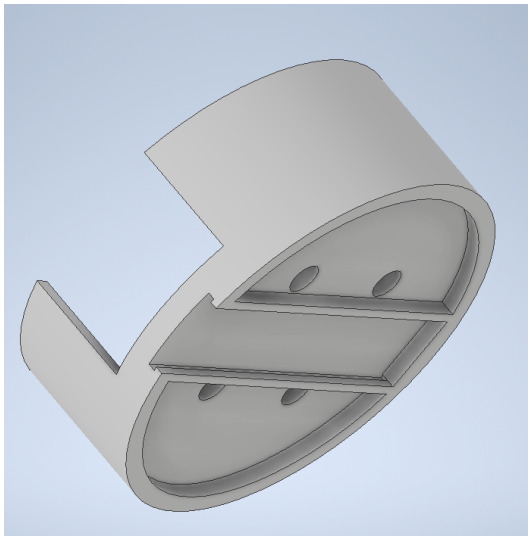


Figure 13: Echo sounders support with rail

The height and the width of the second part could be reduced, to reduce the weight and the global diameter of the robot.

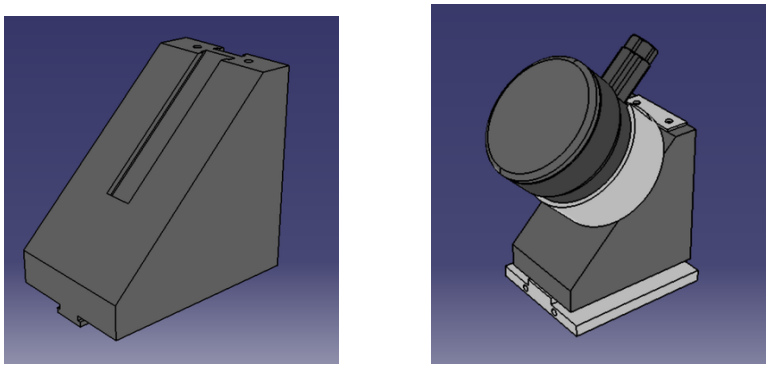


Figure 14: Echosounder fixation assembly

2.7 Lights fixations

The lights we used are the Lumen Subsea Light from Bluerobotics, we designed a support which can probably be folded from a piece of plastic using a thermoforming machine.

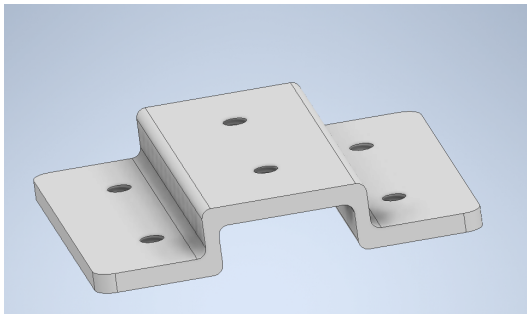
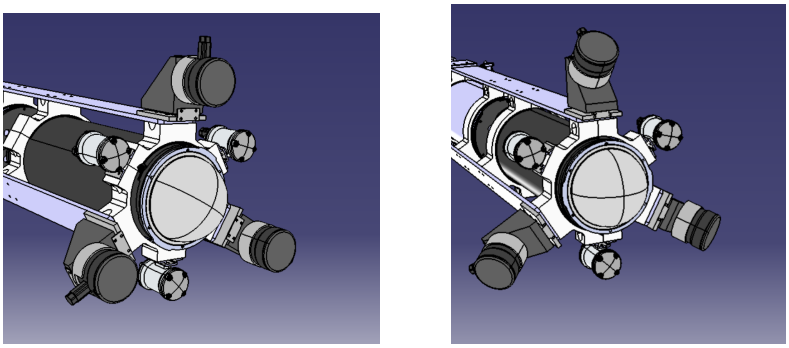


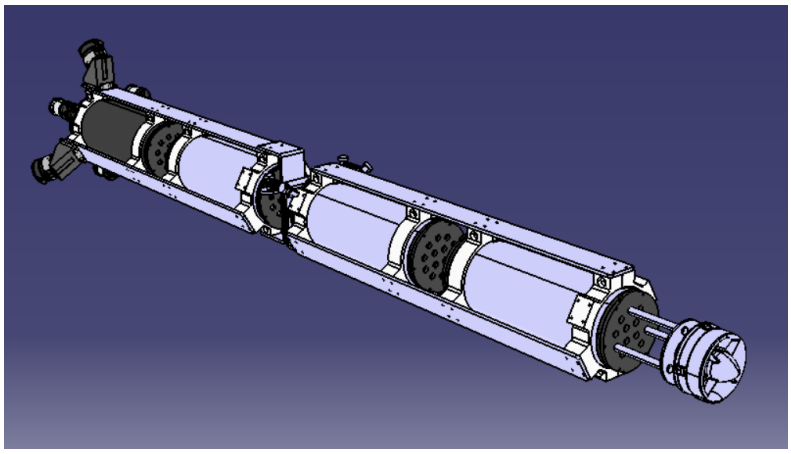
Figure 15: Support for the lights

2.8 Global Final results



(a) 0° (b) 45°

Figure 16: Echo-sounders configurations



3 Electronics

Student : Taddeo GUERIN

3.1 Architecture électronique

As our robot has to move autonomously in underwater karsts, it needs a number of sensors and actuators to ensure its autonomy. All these components are shown in the figure below:

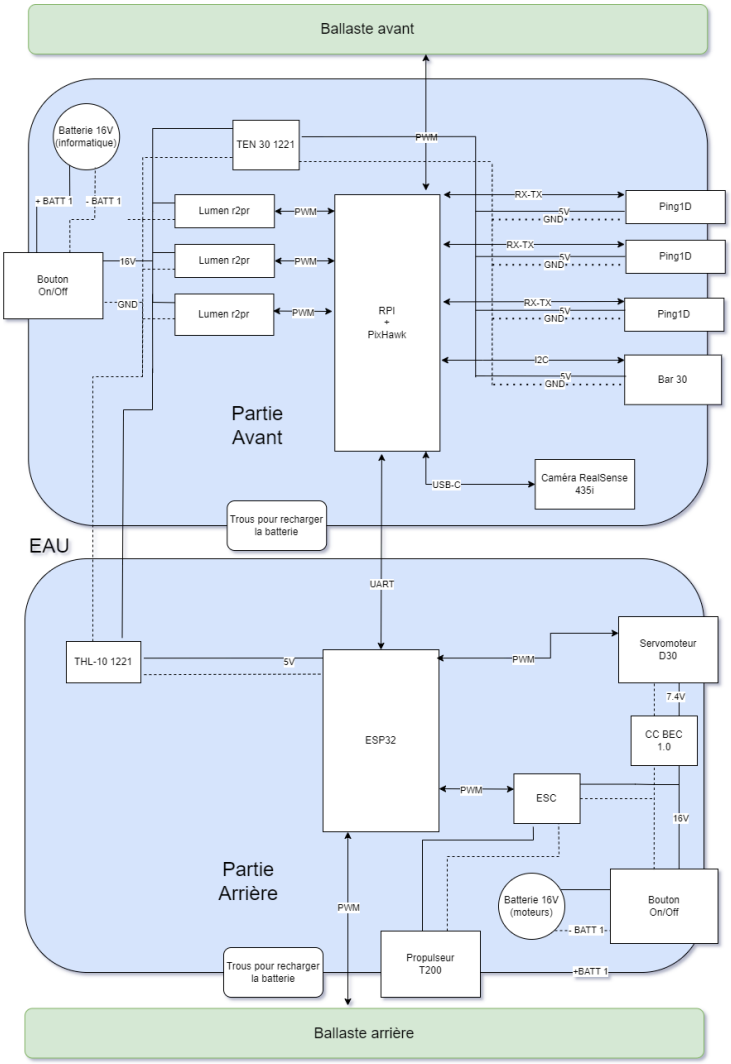


Figure 17: Electronic architecture

The robot being composed of two distinct parts (determined by the **mechanical group**), we divided the various components according to their utility in the robot. Two 4S batteries power the entire system, one being reserved for the actuators and the other for the sensors and controllers/microcontrollers. This division is useful if one of the actuators were to stall and thus use a large amount of current. Indeed, if all components were connected to the same battery and the actuators drew too much current, it could impair the power supply to the

controllers, which could output aberrant values, and thus affect the robot's trajectory. Our system addresses this issue.

3.2 Front Part

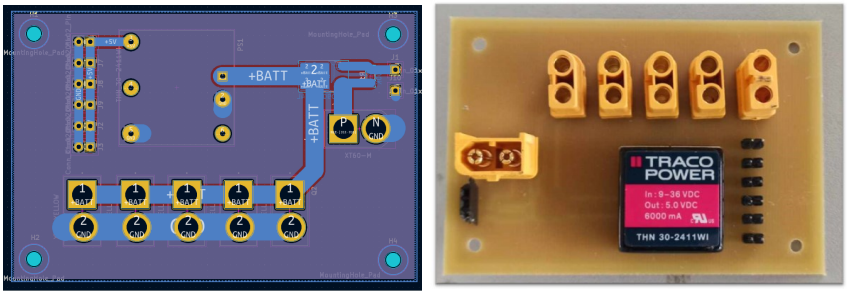
The front part contains the electronic components enabling our AUV to receive and process the information it receives in the water. The Raspberry Pi 4 is used here to perform complex real-time calculations, enabling the processing of data from various sensors. It runs perception, navigation, or trajectory planning algorithms to enable the robot to perform autonomous missions. We also chose it because of its flexibility and ease of programming.

The Pixhawk is primarily used here for its IMU. This device, composed of a gyroscope, an accelerometer, and a magnetometer, is used to measure the linear and angular accelerations experienced by the vehicle. These data are used to estimate the vehicle's attitude (its tilt, orientation) and to compensate for external disturbances. Finally, combined with the other sensors that we will describe just after, the Pixhawk can estimate the vehicle's position, orientation, and velocity in space.

The echosounders, BlueRobotics' Ping2 sonars, are detection and localization devices that use sound waves to measure the distance between the underwater robot and surrounding obstacles. Positioned around our AUV, they help center it in the underwater cavity and navigate it. In addition, we use a pressure sensor (the BAR 30) to measure the surrounding pressure and thus the robot's depth.

Finally, the camera and lights are used for filming and performing visual servoing in karsts, regardless of the ambient lighting. We use the RealSense 435i camera, which is a 3D depth camera designed to capture both high-resolution color images and real-time depth data.

The other components, such as the On/Off Button and the TEN 30 1221 current regulator, have been mounted on a PCB specially designed for the robot. They can be seen in the figure below:



(a) Front PCB(editor) (b) Front mounted PCB

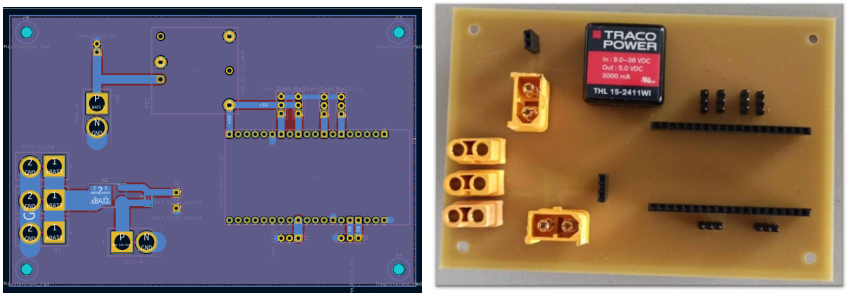
Figure 18: Front PCB

3.3 Rear Part

The rear part is dedicated to the robot's actuators. It therefore contains an ESP32, which with its numerous I/O interfaces, can generate precise control signals to drive these actuators according to the application's needs. This microcontroller is connected to the front part for

power and to receive information from the Raspberry Pi. It is the Raspberry Pi that gives it the instructions to transmit to the thruster and servo motor. The ESP32 also acts as a safety intermediary between the RPI and the actuators, as it is less susceptible to bugs than the other controller, and we can add a filter to it to prevent aberrant values from reaching the motors.ons ajouter un filtre dessus permettant d'empêcher les valeurs aberrantes d'accéder jusqu'aux moteurs.

The other components mainly serve to regulate the different voltages present in this part of the robot, and some are placed on a printed circuit board created directly by a milling machine at ENSTA Bretagne.



(a) Rear PCB(editor) (b) Rear Mounted PCB

Figure 19: Rear PCB

3.4 Final assembly

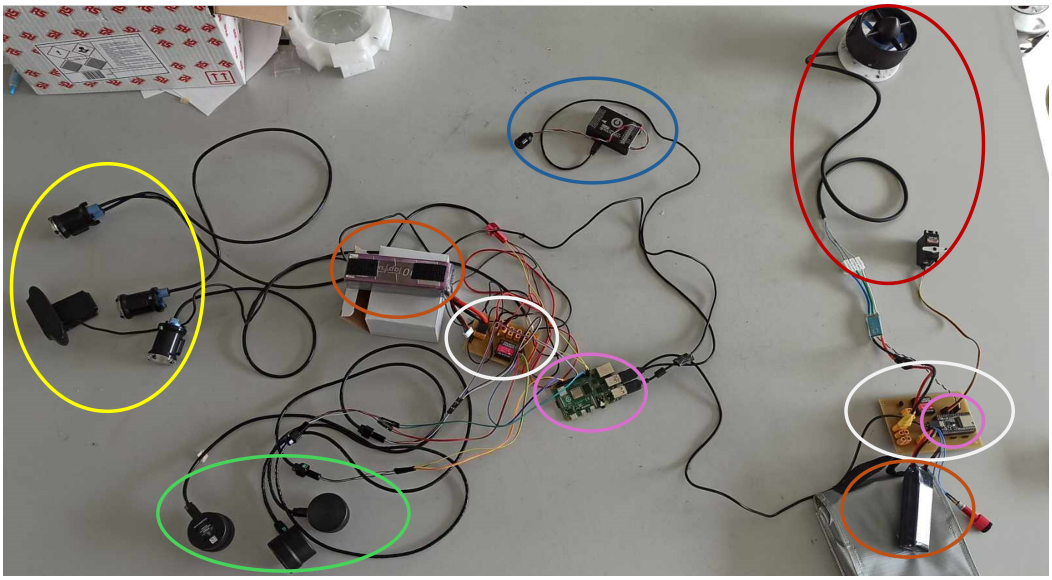


Figure 20: Karst assembly

Yellow for the camera and lumens, Green for the echosounders, Blue for the Pixhawk and pressure sensor, Red for the actuators, Orange for the batteries, White for the PCBs and Pink for the controllers.

4 Ballast

Students : Le Gouallec Jules, Bourzoufi Samy, Leroy Hippolyte

4.1 Rappel

The purpose of the ballast group is to provide the Karstex Robot a ballast system allowing it to control its depth in Z, but also to balance itself on the pitch angle. To balance the robot, we'll place one ballast at the front and one at the rear of the robot. A ballast is a system that can be seen as a pump, i.e. it pumps water in to make itself heavier and pumps water out to make itself lighter. In this way, the robot will have a mass slightly less than its volume, allowing it to sink when it pumps enough water and to rise when it rejects enough. Water levels in each of the ballast tanks will not necessarily be equal at all times, so this dual ballast system ensures zero pitching in the event of the robot's intial imbalance.

What we're more used to seeing is AUVs controlling their depth with thrusters, which can be power-hungry. It's mainly for this and balancing that we chose ballasts.

Initially, the idea was to create the system ourselves, taking inspiration from work already done by others, however rare, such as that of Thomas Le Mézo. However, for lack of time and for safety's sake, we also decided to order some off-the-shelf, despite the limited supply. Here's the ballast from Engel-modellbau, which we have in duplicate.

4.2 Off-the-shell solution

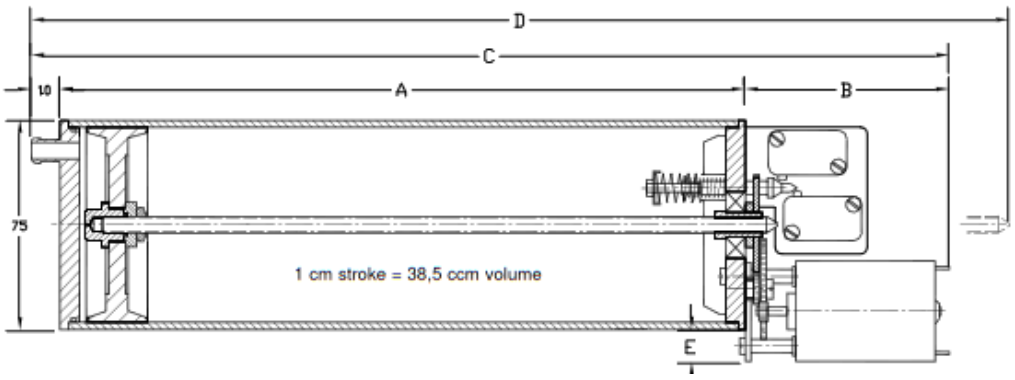


Figure 21: Ballast cross-section diagram

This ballast works as follows: The DC motor drives a worm screw through reducers to move it in or out of the tube, to which is attached a piston to make a syringe system, which pumps water in or out via the orifice on the left of the figure. This orifice allows us to attach a plastic tube to the ballast and pull it out of the robot. The ballast is therefore located in the dry zone inside the robot. Given that distance A in figure [] is 10cm, we need to allow for a total space of 22cm when the auger is extended.

The Engel ballast works quite simply. Controlled with the UNIpro switch, it is possible to carry out a proportional control of the ballast position. The measurement of the position of the piston is done using a Hall effect sensor and magnets sticked on a gear wheel. The configuration of the ballast is done as follows: choose a voltage interval by successively putting two different voltages min and max to the switch successively (the minimum voltage corresponds to the piston pushed to its maximum position and the maximum voltage to its minimum position), choose a mode control, for example in 80/20. Once configured, it is then necessary to provide a voltage in the chosen interval to impose a position on the piston. The input voltage can be transmitted through a PWM signal generated by an Arduino. The UNIpro switch is actually made to prevent the system go over a certain pressure level : a mechanical pressure sensor will send a signal to the switch and the piston will get pushed to its maximal position to empty the ballast. This is what we want to avoid in our case as the ballast can't handle too much pressure (2 bars). These ballasts will help us testing our command law in a pool without worrying about pressure problem.

4.3 Developed solution

In order to provide our own solution we have chosen to base our architecture on a worm screw actuated by a DC motor with odometers. Screw-nut connection eliminates the need to supply energy to hold the ballast in a fixed position. Moreover, the screw pitch, in the order of a millimetre, multiplies the torque supplied by the motor, enabling it to withstand the high pressures exerted by the water. When the motor rotates, it drives a tapped gear, which in turn moves the worm screw. We dimensioned the tube to contain a volume of 200 mL, for a diameter of 75 mm, giving us a useful length of 45 mm. We wanted the ballast to go to a depth of 50 m, which implies a 5 bar differential pressure. For a piston diameter of 75 mm, it represents a force of 2210N to counter.
The minimal torque of the motor is calculated as following :

$$\begin{aligned} dW_1 &= \frac{C_{motor}}{R_1} \cdot dx_1 \\ dW_2 &= F_{water} \cdot dx_2 \\ dx_1 &= d\theta_1 \cdot R1 \\ dx_2 &= d\theta_1 \cdot \frac{R_1}{R_2} \cdot \frac{s_t}{2 \cdot \pi} \end{aligned}$$

(s_t is the screw thread)
As a minimum, the input torque must be equal to the output torque: $dW_1 = dW_2$, which leads to :

$$C_{motor} = F_{water} \cdot \frac{R_1}{R_2} \cdot \frac{s_t}{2 \cdot \pi}$$

With a motor torque of 12kg.cm and a reduction ratio of 1, a force of 3760N is provided which is enough for our application.

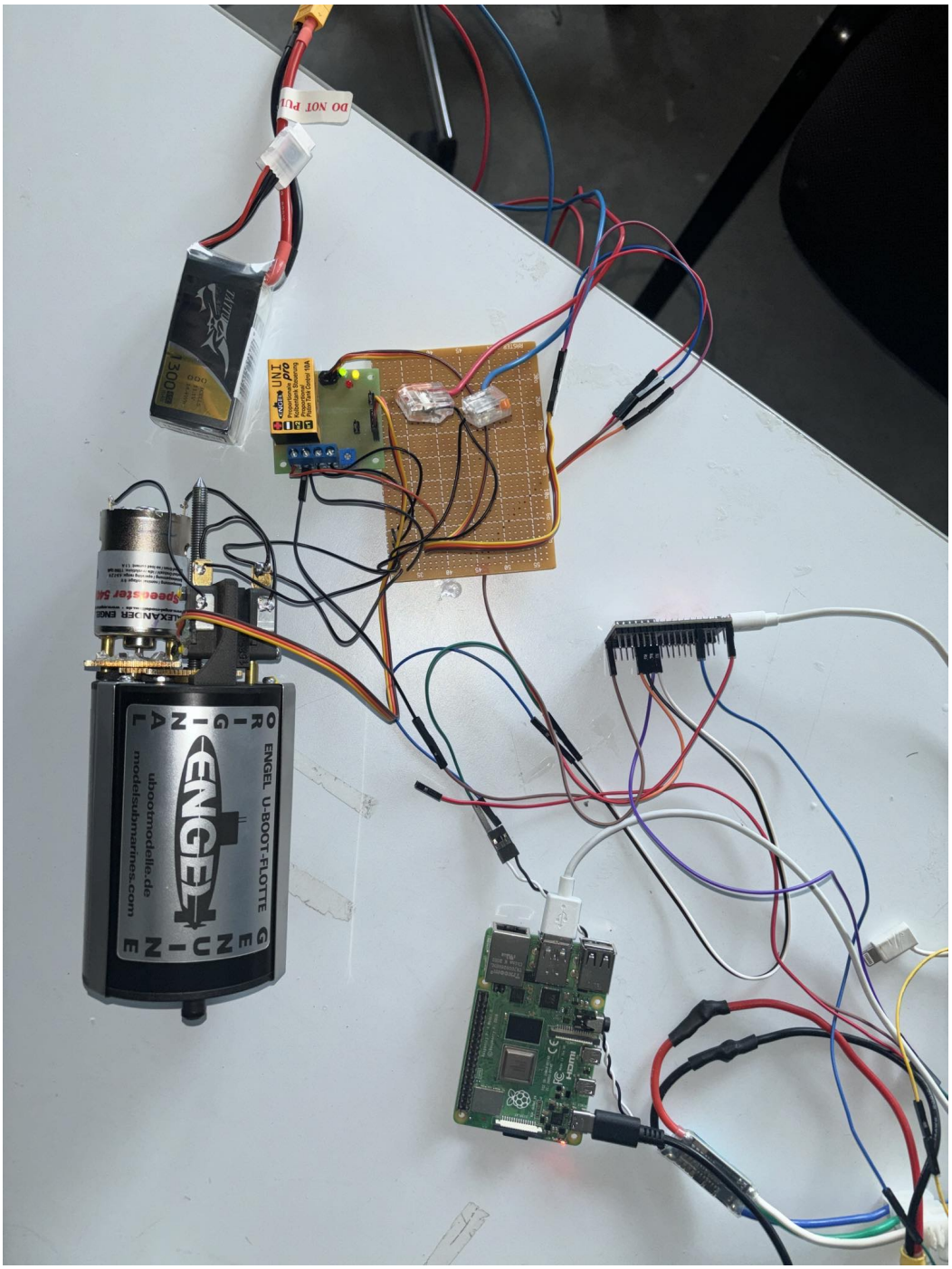


Figure 22: Ballast control circuit

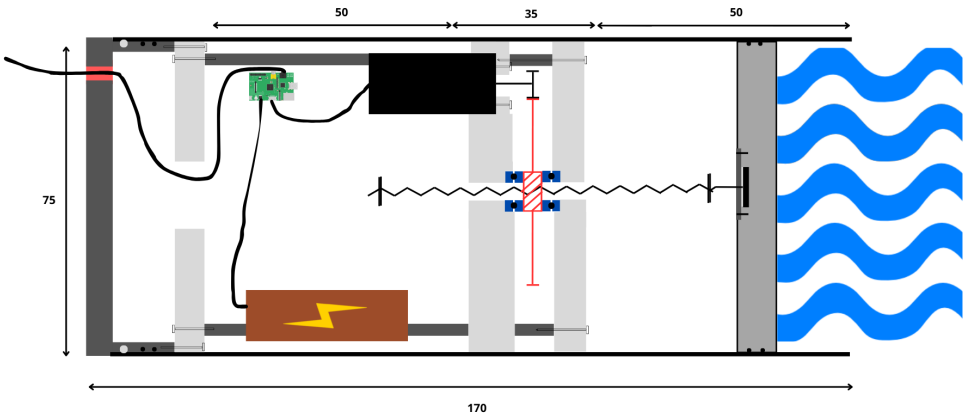


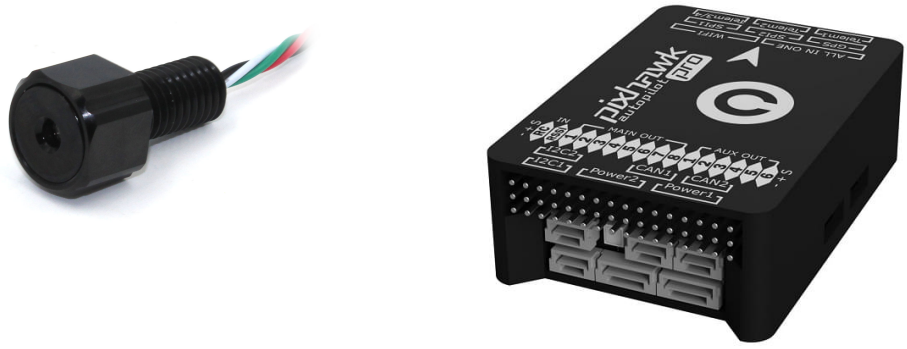
Figure 23: Diagram of the developed ballast

5 Inertial Measurement Unit and Depth Sensor

Students : BELLOT Victor, BETTON Gabriel, RIZK Catherine

5.1 Pixhawk 3 Pro Autopilot

To manage the Blue Robotics *BAR30* pressure sensor, we will be using the *Pixhawk* 3 Pro autopilot with its integrated IMU.



(a) Blue Robotics *BAR30* pressure sensor

(b) *Pixhawk* 3 Pro autopilot

Figure 24: IMU and Depth Sensor gears

The *BAR30* can measure up to 30 bar (300 m depth) and communicates over I^2C. It also includes a temperature sensor accurate to $\pm 4\,^{\circ}C$.

The *Pixhawk* autopilot communicates with the main processing unit (a Raspberry 4 in our case) through USB. It sends and receives **MAVLink** messages to change the autopilot state and to claim the measurements we need. Here are the types of the messages we are looking for :

- ATTITUDE (roll, pitch, yaw, rollspeed, pitchspeed, yawspeed)
- SCALED_PRESSURE2 (press_abs, press_diff)

We calibrated the integrated IMU using *QGroundControl* software. This involved placing the IMU in various positions (along different axes) and letting it acquire data in these positions. The following picture shows how well the process went:

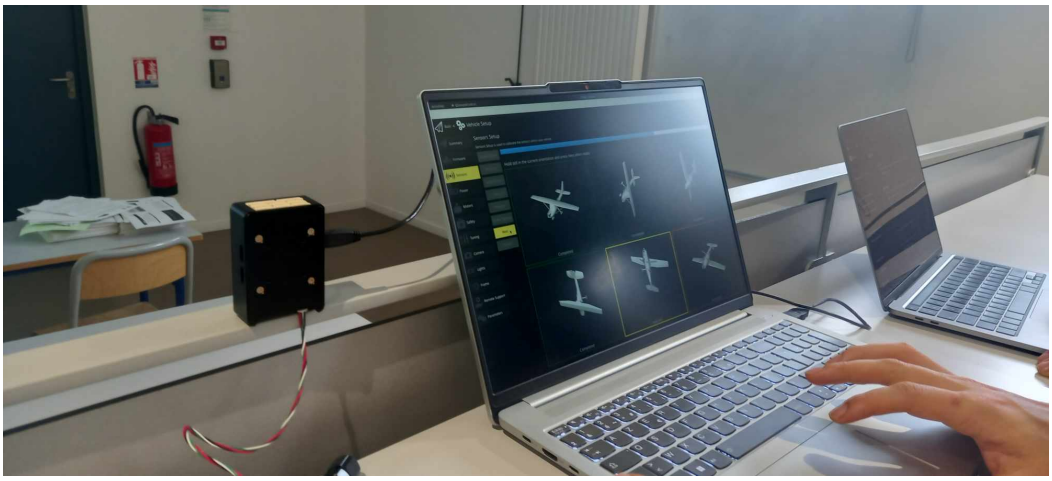


Figure 25: The IMU calibration process

5.2 ROS2 interface

To read the **MAVLink** messages, we first attempt to use the *MAVROS* ROS2 package. We hadn't succeeded in using it properly so we are finally using the *pymavlink* Python library. This enable us to create a ROS2 node publishing two sensor topics : */imu* and */pressure*. We've also written a launch file to facilitate integration of the package into the common ROS. It allows you to specify the USB port to which the *Pixhawk* is connected using the "usb_port :=<>" argument.

Our ROS package is untitled **inertel_toolz_tkt_bb** (referring to the last names of its three authors), and the IMU/pressure node can be launched using the following command :

```
ros2 launch inertel_toolz_tkt_bb sensors.launch
```

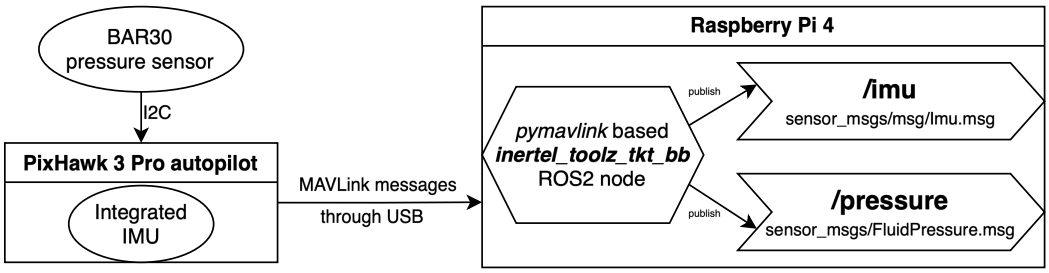


Figure 26: Hardware and Software structure

6 Echosounders

Students : Martin PILON, Emilie LEDOUSSAL, Marguerite MIALLIER

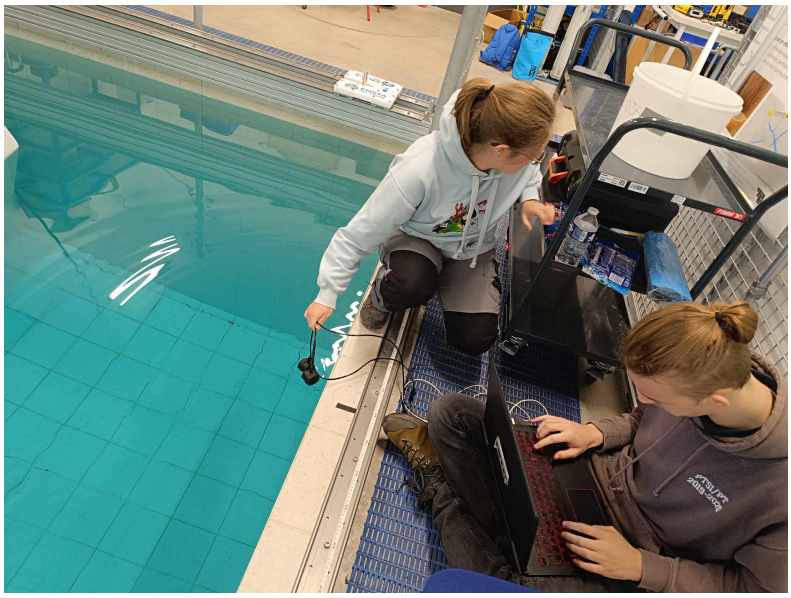


Figure 27: Testing the echosounders in the pool at ENSTA Bretagne

6.1 Choosing echo sounders

Several solutions were considered for implementing obstacle detection using an echo sounder. One of them was the use of sector sonar or scanning sonar, but in addition to being difficult to attach to the front of the robot due to the porthole allowing camera vision, this would not have been relevant because it would have made it possible to know the information only in the space immediately surrounding the front of the robot, therefore preventing possible obstacles from being detected in advance. We therefore opted for the use of several 1D sonars, positioned at the front of the robot at an angle of 45°. The three of them are separated by 120°.

The sonar Ping2 from Blue Robotics has been chosen for this project, because it is easy to use in Python with the `brping` library developed by Blue Robotics itself. The library can be found at : https://pypi.org/project/bluerobotics-ping/.

6.2 Getting the echo sounders data

At first the Python ROS node has been developed for only one echo sounder. Tests have been led in the pool, to verify that the distance values obtained by the node were the same as those showed by the Ping Viewer software, developed by the manufacturer.

Then, the node has been adapted to the use of three sonars. As the three of them publish at the same frequency, they cannot be used at the same time due to the interference. They

are therefore used one by one : when the first one has received the signal back, the second one emits, waits for the responds, and finally the third one does the same. This process take around 0.68 seconds.

6.3 ROS2 interface

Data of each sonar is published on one topic as a `Float32` message. The publishers publish at a period of 0.1s. If the data hasn't been updated since the last publication, the last value will be republished. The low frequency of update should not be a problem since the robot moves slowly.

7 Hydrodynamic model

Student : Gwendal Crequer

The goal of this part is to estimate drag coefficients and/or friction actions depending on the flow and the configuration of the structure. This work has first been done with a Fluid-structure interaction solver, but an simpler approach will be presented for the succession.

7.1 COMSOL modeling

The goal of this method is to find a general law, in laminar flow, and to enhance parameters which could be fitted to the real *karstex* robot. The simulation environment is presented in figure 28.

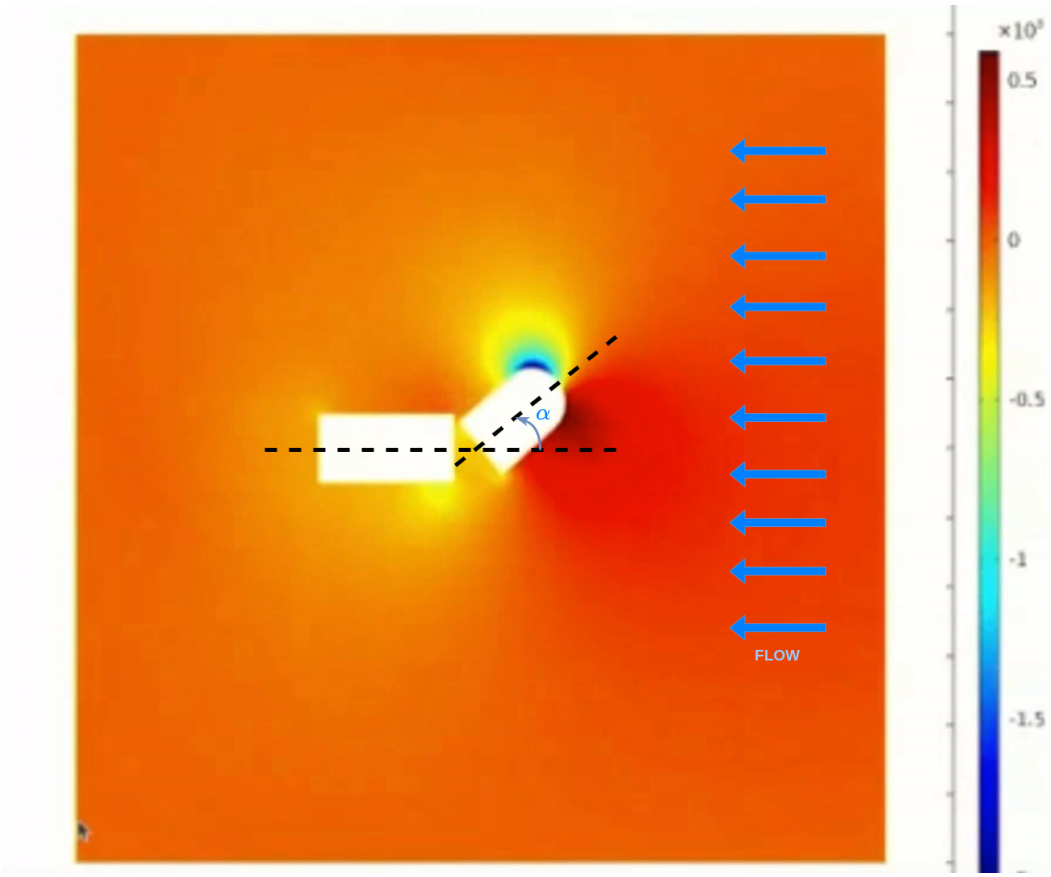


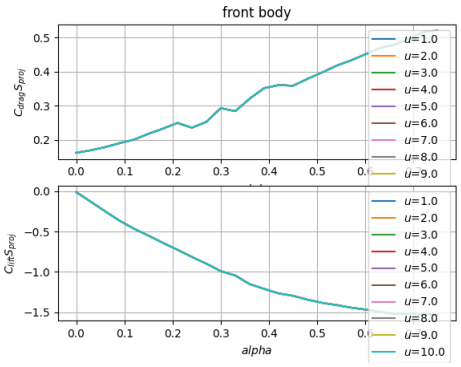
Figure 28: Pressure field around the shape

The angle between the 2 bodies changes but the flow is only in the opposite direction of the thrust (i.e. hypothesis : no drift). The two shapes are considered with a revolution symmetry, so the model is simplified to a plane problem which would need experimental validation. Because of the hypothesis of no-drift effect, the behavior of the robot is supposedly the same in all orientations while the flow is opposed to the thrust force.

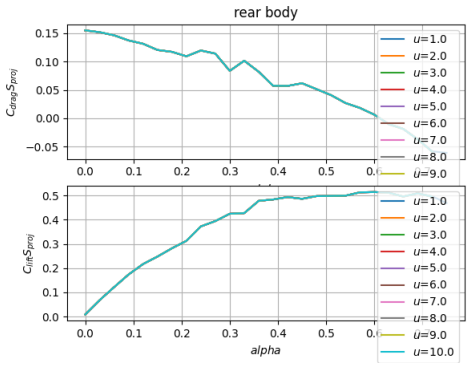
The situation is presented for various α values in figure **??**. We can notice that the over pression between the the bodies progressively becomes a depression, then, a suction phenomenon appears.

7.2 Parameters estimation

The integral of pressure around each body permits to estimate drag and lift forces. Because $\vec{F}_{drag} = -\frac{1}{2}\rho.S_{proj}C_{drag}||\vec{u}||\vec{u}$, we can easily estimate $S_{proj}C_{drag}$ for various angles and various speeds, as presented in Figure 29.



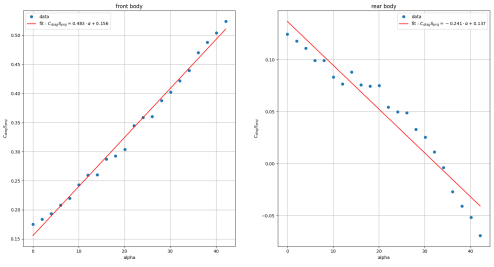
$C_{drag}S_{proj}$ and $C_{lift}S_{proj}$ for front body, depending on speed



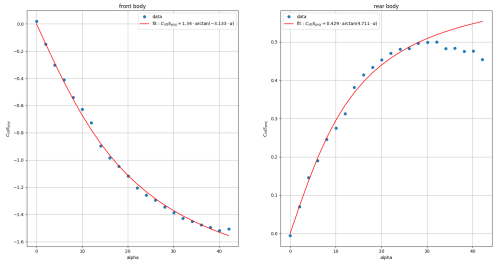
$C_{drag}S_{proj}$ and $C_{lift}S_{proj}$ for rear body, depending on speed

Figure 29: (Maybe) because of the laminar hypothesis used in the solver, there is no drag and lift dependency on the flow intensity, notice once again the suction effect for $\alpha > 0.6rd$

Considering symmetry of the behaviour for $\alpha > 0$ and $\alpha < 0$, heuristics with 2 parameters are suggested figure 30.



$C_{drag}S_{proj}$ is approximated by an affine relation



$C_{lift}S_{proj}$ is approximated by a arctan relation

Figure 30: Heuristics and parameters to determine

7.3 Critics about the results

There are 2 main issues which makes this estimation unusable for now:

- All of these operations are considered with no drift, but 3D simulations of the robot lead to an important drift, as lift coefficients figure 30 enhance too. Then, that might imply to switch model and to log a parametric 3D surface/heuristic for the same result and the curve fitting with real data. That is computationally heavy,
- Lift effect for high α value is not supposed to be higher than drag effect. It could be due to an illogical projection of forces in the wrong frame (frame of the front body while the dynamical model is only working in the frame of the rear body.

7.4 Easier alternative

2 alternative methods might be used to get a robust hydrodynamic models:

- Assemble the robot and log a lookup table of all the forces and torques on the articulation with experimental measurement (maybe find an relation with the speed of the flow). Torque values can help for finding the application point of the force. This method is close to the real environment and can be highly accurate, but needs an assembled robot and a method to measure the pose of the robot, and the water flow.
- Simplifying each body by an ellipsoid, theoretical results exist for such primitive, smooth shapes. The main drawback, if each ellipsoid is computed independently, is that the rear ellipsoid drag coefficient might be highly affected. Then, a solution is to consider that the input flow for the rear body is a combination of the main flow the mean flow at the side of the front body, near the articulation (linear combination ?).

8 Équations d'état

Students : Johan Berrier-Gonzalez, Gwendal Crequer

8.1 Description

The aim of this part is to define the equations of state of the robot. This will be useful for the rest of the project, as it will enable the robot's behaviour to be represented and therefore enable control and guidance laws to be tested, etc.

Initially, we provided a simplified version of the model to give the other members of the project an initial working tool.

In a second phase, the idea was to propose a more realistic method for the model using different approaches to dynamics (PFD, Kinetic Energy Theorem, etc.).

8.2 Simplified 2D study of the Karst

In the first version of the model, we decoupled the robot's behaviour to represent the robot without the ballast and therefore without the articulated part.

Explanation of equations of state

In the first part of the study we obtain one of the bodies, so we end up with the following model :

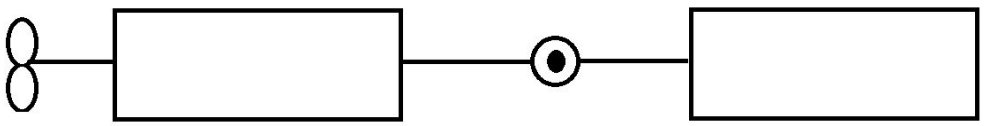


Figure 31: Karst System

We have a single thruster at the rear to keep the robot moving.

We therefore have the velocity vector which has a component along x. Thanks to the use of ballasts, we can manage its translation along the z axis (this part will be studied in the control section) and prevent any rotation along the x axis.

We can therefore study the system in the (x,y) plane without taking the z axis into account.

I remind you of the model

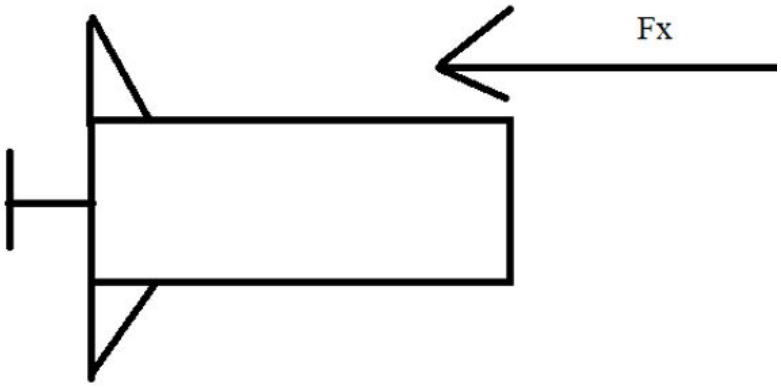


Figure 32: action eau sur robot

$$\dot{Vr} = a_r - \omega_r \wedge V_r$$

$$\omega_r = 0$$

so

$$\dot{Vr_x} = a_{rx}$$

Vr_x: composante seulement suivant x

$$F_x * V = d/dt(1/2 * m * V^2) = m * v * \dot{V} = m * a_{rx} = (m + m_a) * a_{rx}$$

$$m_a = k_a * \rho_{eau} * V_{rob} = m$$

$$F_x = K_p * u_0 - 1/2 * C_x * S_x * \rho_{eau} * V^2$$

$$ar_x = F_x/2 * m = (K_p * U_0^2/2 * m) - (C_x * S_x * \rho_e au * V^2/4 * m$$

with

$$p_1 = K_p * /2 * m$$

$$p_2 = C_x * S_x * \rho_{eau}/4 * m$$

The result is

$$\dot{V} = p_1 * U_0^2 - p_2 * V^2$$

with U_0 : commande du propulseur

Now let's take a look at the real systems

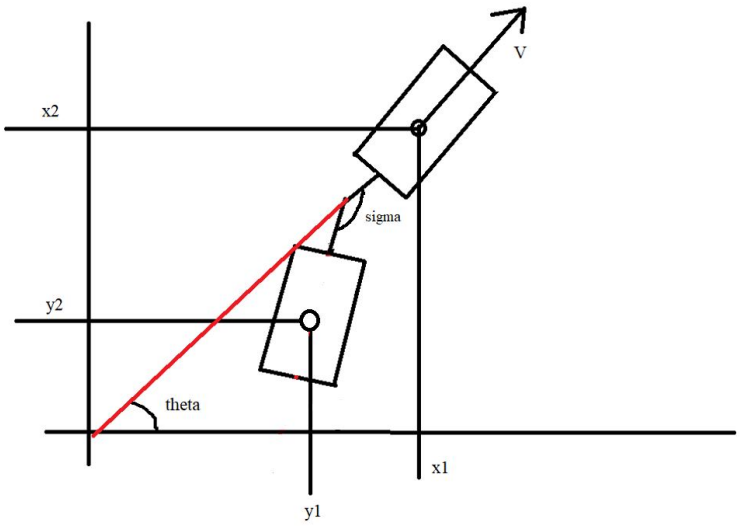


Figure 33: 2D System

$$\begin{cases} \dot{x1} = v\cos(\theta) \\ \dot{x2} = v\sin(\theta) \\ \dot{\theta} = ? \\ \dot{x4} = v\sin(x3 - x4) \\ \dot{v} = p_1^2 - p_2^2 \end{cases}$$

θ : angle servomoteur

Now let's take a closer look at x4, because its expression is incomplete

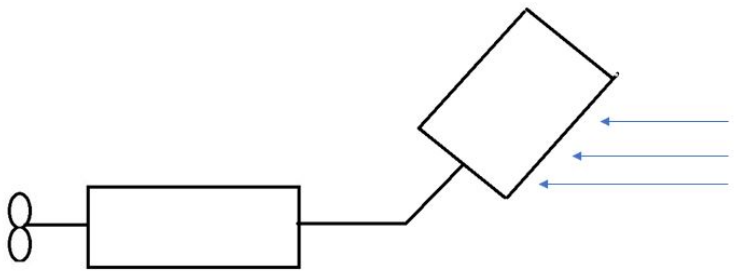


Figure 34: Systeme 2D

x4 represents the angle between body 1 and 2.

Over time this angle will vary to follow the orientation of the head. When the head is tilted, the pressure of the water on the head increases, creating a torque and causing the rear section to rotate.

We therefore have :

$$\begin{cases} \dot{x1} = v\cos(\theta) \\ \dot{x2} = v\sin(\theta) \\ \dot{\theta} = ? \\ \dot{x4} = v\sin(x3 - x4) - k * x4 \\ \dot{v} = p_1^2 - p_2^2 \end{cases}$$

with

$$K = M * L = \frac{P * L}{S} = \frac{p_0 g h \cos(\theta)}{S} \tag{1}$$

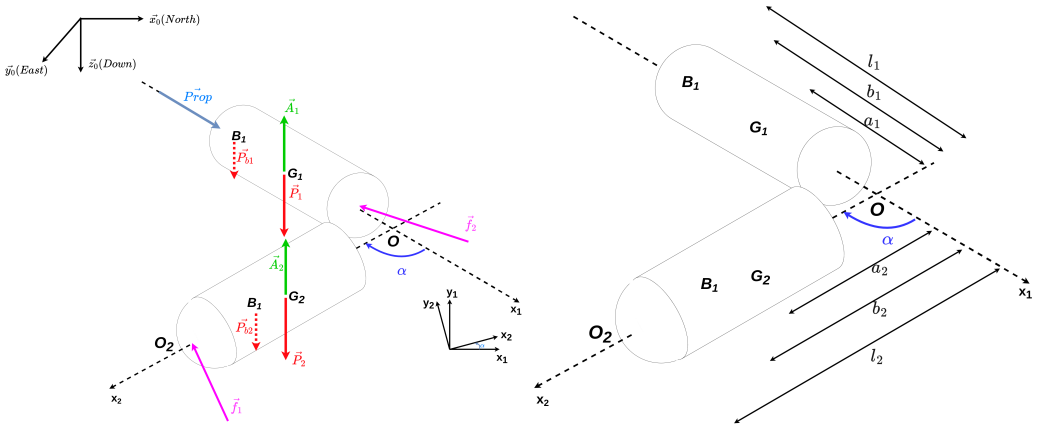
8.3 3D model and refinement

This model is based on the Newton's law of Motion, through mechanization equations. Then, the main work is about defining forces and their intensity, application points, localizing the center of mass and taking knowledge of the limits.

8.3.1 3D model and parameters

Actions and Geometry are defined in the figure 35. Actions considered are :

- the weight of each body, at their center of mass : $\overrightarrow{P_i} = mg\overrightarrow{z_0}\Big|_{G_i,\mathcal{R}_1}$,
- the weight of each ballast, at their center of mass: $\overrightarrow{P_{bi}} = m_i g\overrightarrow{z_0}\Big|_{B_i,\mathcal{R}_1}$,
- the buoyant force of each body, at their centroid, $\overrightarrow{A_i} = -mg\overrightarrow{z_0}\Big|_{G_i,\mathcal{R}_1}$,
- the thrust of the propeller, at the back of the rear body, $\overrightarrow{T} = T\overrightarrow{x_1}\Big|_{T_1,\mathcal{R}_1}$,
- the viscous friction forces, expressed at an arbitrary point, defined as the extremum of each body : $\overrightarrow{F_i} = -k(state)\dfrac{\overrightarrow{V_{O_i \in body_i}}}{\left\|\overrightarrow{V_{O_i \in body_i}}\right\|}\Big|_{O_i,\mathcal{R}_1}$



Forces considered in the 3D model — Geometrical parameters of the model

Figure 35: Scheme of the 3D model

Some aspects have been considered :

- A small centerboard could be added on *body 1* to balance the torque of the thruster.
- The application point of the drag is realistic in a case of a small drift. However, it is not the case here, it might be interesting to apply the drag force at the centroid (to avoid excessive calculus) or at the point of the body where the speed is maximal,
- lift force is not considered in this model, supposing that most viscous effects create drag,
- There is no added mass in this model, that is not justifiable,
- the drag coefficient is a constant, because the hydro-dynamical model has been considered as non-relevant. It is destined to change with a more accurate modeling of hydrodynamical properties, empirically or with a model including parameters of the final shape.

8.3.2 Center of mass and reduction of the problem

The dynamical equations are applied at the center of mass, which is moving depending of the angle between *body 1* and *body 2*. All inertia matrices (cylinders for bodies, point for ballasts until their shape are decided) are so moved to the center of mass.

Torques due to all the forces are computed at the center of mass. It will be possible to add pure torques easily. All forces are expressed in the body frame.

8.3.3 Mechanization equations and state model

The proposed state is :

$$\begin{cases} \mathbf{X} & \text{Position in } \mathcal{R}_0 \\ \mathbf{R} & \text{Rotation } \mathcal{R}_0 \to \mathcal{R}_1 \\ \mathbf{v} & \text{Speed in } \mathcal{R}_1 \\ \omega & \text{Angular speed in } \mathcal{R}_1 \end{cases} \quad \text{The state equations is so :} \quad \begin{cases} \dot{\mathbf{X}} = \mathbf{R}\mathbf{v} \\ \dot{\mathbf{R}} = \mathbf{R}\,(\omega\wedge) \\ \dot{\mathbf{v}} = \frac{1}{m_{tot}}\,\Sigma\overrightarrow{\mathbf{F}}\Big|_{\mathcal{R}_1} - \omega\times\mathbf{v} \\ \dot{\omega} = \mathbf{I}_G^{-1}\left(\Sigma\overrightarrow{\mathbf{C}}\Big|_{\mathcal{R}_1} - \omega\times\mathbf{I}_G\omega\right) \end{cases}$$

and can be solve iteratively with *Runge-Kutta* methods for $\mathbf{X}$, $\mathbf{v}$ and ω, an exponential method can be suggested for $\mathbf{R}$. The state model is also adapted for euler angles.

Beware : dynamical effects due to the control of the servomotor are not considered, and the center of mass is moving in its own frame.

8.4 results

The dynamical model is coherent with expectation for no inclination between the 2 bodies, or with no propulsion (achieve equilibrium point). Gwendal Crequer and Leo Bernard worked at explaining a phenomenon which creates an intense rotation for very small inclinations : it might be due to the application point of the drag force, and its evolution due to the hydrodynamical model is not implemented.

8.5 liens avec les autres groupes

Ces equations 2D ont été fournis dans les groupes suivants :

Figure 36: Lien avec les autre groupes

9 Control

Students : Tristan LE FLOCH, Louis-Nam GROS, BERNARD Leo, Mathieu PITAUD, Louis GILLARD

9.1 Control Architecture

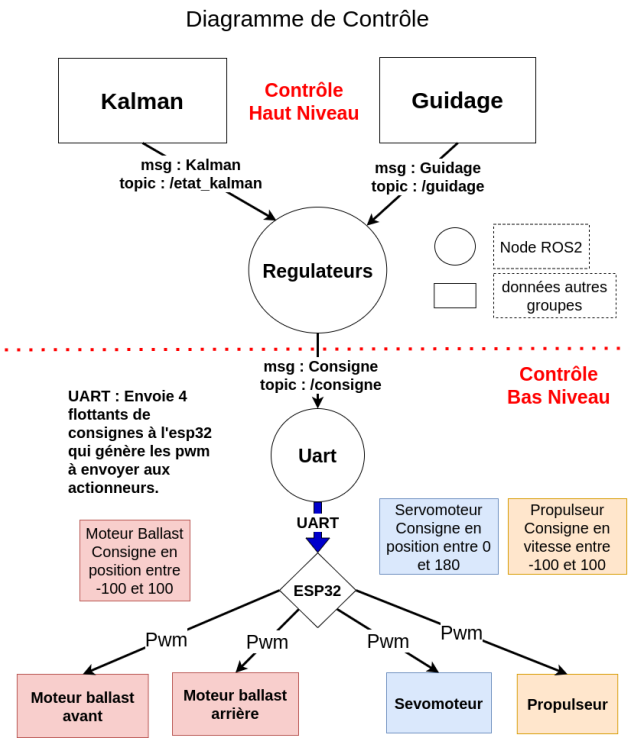


Figure 37: Control Architecture

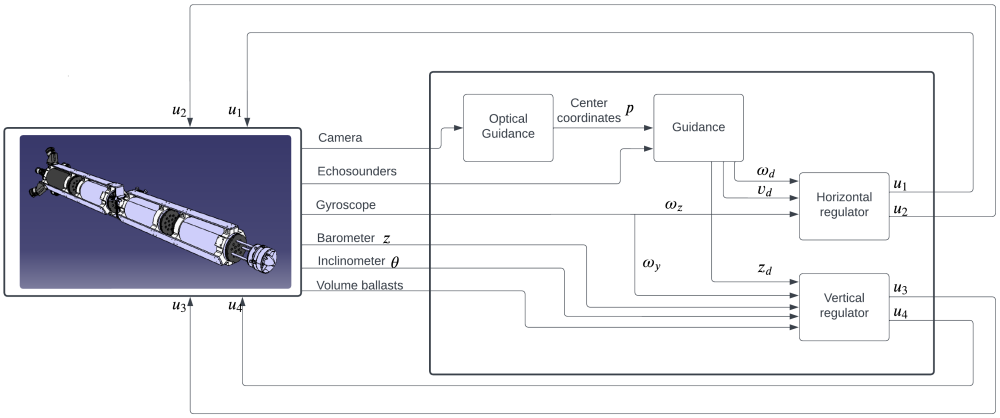


Figure 38: Navigation Guidance Control Architecture

9.2 Vertical Regulator

Work with groups Ballast, Guidance, Simulation and Kalman

9.2.1 Context and definitions

We call A the rear module and B the front module.

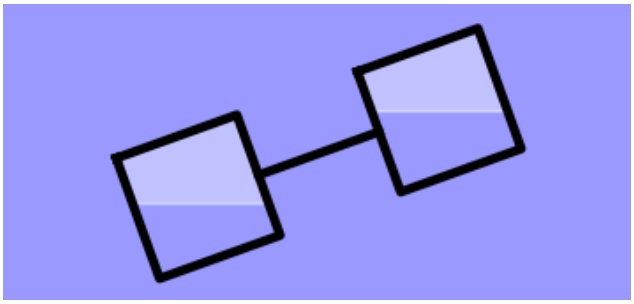


Figure 39: Model of the double ballast

We make the assumption that it is the volume that is modified, not the mass, when the ballasts fill or empty.
Let's state some constants and variables:

- Constants:
  - m_{A0} and m_{B0}: masses of modules A and B
  - $m_0 = m_{A0} + m_{B0}$: total mass of the robot
  - V_{A0} and V_{B0}: volumes max of modules A and B (ie when there is no water in the ballasts)
  - L: the length between the two barycenters of the ballasts, which are both a length $L/2$ from the total barycenter of the robot. It means we consider the robot symetric for the masses.
- Variables:
  - z: the depth
  - v_z: the speed along z-axis
  - θ: the pitch
  - $\dot{\theta}$: the angular velocity for the pitch
  - V_{Aw} and V_{Bw}: volumes of water in ballasts A and B
  - u_3 and u_4: the commands sent to the ballasts A and B (variations of volumes)

Then, we define the state vector and the evolution equation as:

$$\mathbf{x} = \begin{pmatrix} z \\ v_z \\ \theta \\ \dot{\theta} \\ V_{Aw} \\ V_{Bw} \end{pmatrix} \quad ; \quad \dot{\mathbf{x}} = \begin{pmatrix} v_z \\ g*(-1+(V_{A0}+V_{B0}-(V_{Aw}+V_{Bw}))*\rho_0/m_0) \\ \dot{\theta} \\ \frac{Lgcos(\theta)}{2J_{YY}}(m_{B0}-m_{A0}+(V_{A0}-V_{B0}-V_{Aw}+V_{Bw})*\rho_0) \\ u_3 \\ u_4 \end{pmatrix}$$

Proof using the Newton's second law of motion :

$$\sum F_z = -m_0 g + \rho_0 g V_A + \rho_0 g V_B = m\dot{v_z} \tag{2}$$

With the volume of the ballast A : $V_A = V_{A0} - V_{Aw}$, and for B : $V_B = V_{B0} - V_{Bw}$, it comes :

$$\dot{V_z} = g(-1+\rho_0(V_{A0}+V_{B0}-(V_{Aw}+V_{Bw}))/m_0) \tag{3}$$

And also :

$$\sum M_Y = \frac{Lcos(\theta)}{2}(-m_{A0}g+\rho_0 g V_A) - \frac{Lcos(\theta)}{2}(-m_{B0}g+\rho_0 g V_B) = J_{YY}\dot{\dot{\theta}} \tag{4}$$

Which leads to the following equation :

$$\dot{\dot{\theta}} = \frac{Lgcos(\theta)}{2J_{YY}}(m_{B0}-m_{A0}+\rho_0(V_A-V_B)) \tag{5}$$

In order to process a precise control using feedback linearization, we need to know the state precisely. We know θ and $\dot{\theta}$ with high-precision thanks to the inertial unit, and we also know the volumes V_{Aw} and V_{Bw} from the electronics in the ballasts. But there is noise on the pressure measurements, so we need an accurate estimation of the real depth and speed. For this purpose we use a Kalman Filter.

9.2.2 Kalman Filter

We try to estimate z and v_z using the measurements of a pressure sensor.

The Kalman system can be described by the following model:

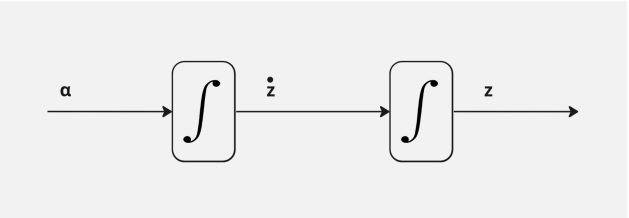


Figure 40: Kalman model

The state vector to be estimated is therefore: $\mathbf{x} = \begin{pmatrix} z \\ v_z \end{pmatrix}$

with the evolution function:

$$\dot{\mathbf{x}} = f_c(\mathbf{x}, a) = \begin{pmatrix} v_z \\ a \end{pmatrix}$$

Then, we can discretize the equation and apply the Euler method:

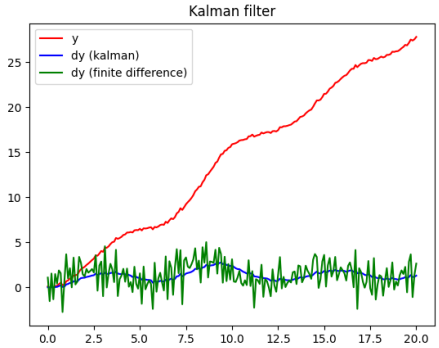


Figure 41: Simulated evolution of y and $\dot{y}$ using Kalman approach and finite differences, with respect to time

$$\mathbf{x}_{k+1} = \mathbf{A}_k \mathbf{x}_k + \mathbf{u}_k + \alpha_k$$

with $\mathbf{A}_k = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}$; $\mathbf{u}_k = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

As we don't know the acceleration, we introduce an uncertainty, in which we consider the acceleration due to the weight and the buoyancy but also the possible perturbations that could act on the robot. For this purpose, we will define a_{max}, the maximum acceleration that the system could be subject to. Then, we have:

$$\Gamma_{\alpha_k} = dt \cdot \Gamma_\alpha = dt \begin{pmatrix} 0 & 0 \\ 0 & a_{max}^2 \end{pmatrix}$$

We measure the pressure, from which we deduce a depth, so:

$$\mathbf{y} = g(\mathbf{x}) + \beta = C\mathbf{x} + \beta$$

with

$$C = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

so:

$$\mathbf{y}_k = C\mathbf{x}_k + \beta_k$$

and

$$\Gamma_{\beta_k} = \frac{1}{dt} * depth\_accuracy^2$$

For the simulation we used $a_{max} = 10 m.s^{-2}$ and $depth\_accuracy = 20cm$

After simulating the robot with arbitrary commands for the ballasts, we get the following results: y is the estimated depth of the robot and dy the estimated vertical speed. We can clearly see that the Kalman approach is satisfactory and gives a much better curve than the finite difference method because the noise in the observation throws off the finite differences.

9.2.3 Feedback Linearization

$$\begin{cases} \dot{\mathbf{x}} = f(\mathbf{x}) + g1(\mathbf{x}) * u_3 + g_2(\mathbf{x}) * u_4 \\ \mathbf{y} = h(\mathbf{x}) \end{cases}$$

with:

$$f(\mathbf{x}) = \begin{pmatrix} v_z \\ g*(-1+(V_{A0}+V_{B0}-(V_{Aw}+V_{Bw}))*\rho_0/m_0) \\ \dot{\theta} \\ 1/J_{YY}*L/2*cos(\theta)*g*(m_{B0}-m_{A0}+(V_{A0}-V_{B0}-V_{Aw}+V_{Bw})*\rho_0) \\ 0 \\ 0 \end{pmatrix} \quad ;$$

$$g1(\mathbf{x}) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad ; \quad g2(\mathbf{x}) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad ; \quad h(\mathbf{x}) = \begin{pmatrix} z \\ \theta \end{pmatrix}$$

We define, using Lie derivatives, $v = \begin{pmatrix} z^{(3)} \\ \theta^{(3)} \end{pmatrix} = A(\mathbf{x}) \cdot \mathbf{x} + b(\mathbf{x})$

with $A(\mathbf{x}) = \begin{pmatrix} \mathcal{L}_{g_1}\mathcal{L}_f^2 h_1(x) & \mathcal{L}_{g_2}\mathcal{L}_f^2 h_1(x) \\ \mathcal{L}_{g_1}\mathcal{L}_f^2 h_2(x) & \mathcal{L}_{g_2}\mathcal{L}_f^2 h_2(x) \end{pmatrix}$ and $b(\mathbf{x}) = \begin{pmatrix} \mathcal{L}_f^3 h_1(x) \\ \mathcal{L}_f^3 h_2(x) \end{pmatrix}$

Finally, we define the control law by choosing $v = (w-y) + 3(\dot{w}-\dot{y}) + 3(\ddot{w}-\ddot{y}) + w^{(3)}$
with w our desired state for y and $\dot{w}, \ddot{w}, w^{(3)}$ its derivatives
and we have:

$$u = \begin{pmatrix} u_3 \\ u_4 \end{pmatrix} = A(x)^{-1}(v - b(x))$$

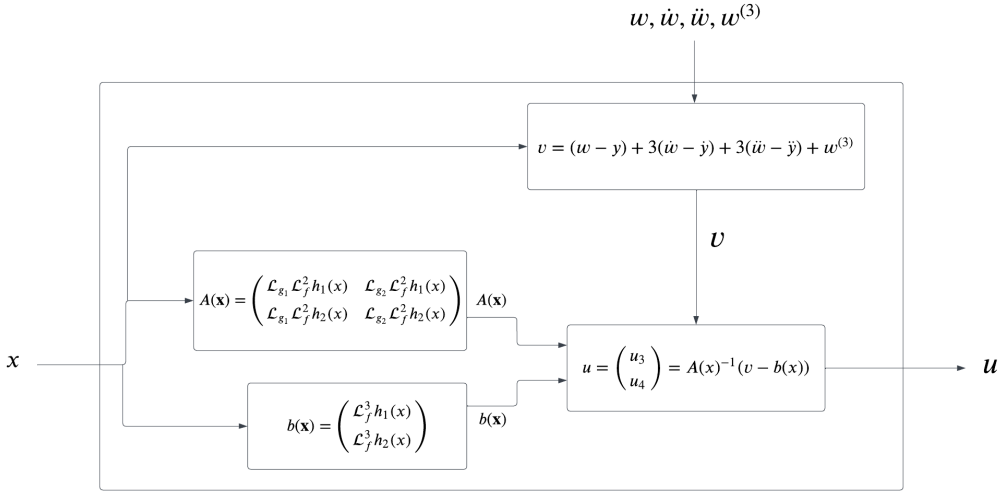


Figure 42: Feedback Linearization process

9.2.4 Simulation Python

Simulation in Python environment of the vertical regulator on a simple double ballast system

9.3 Horizontal Regulator

Work with groups State Equations, Guidance, Simulation

9.3.1 Context and definitions

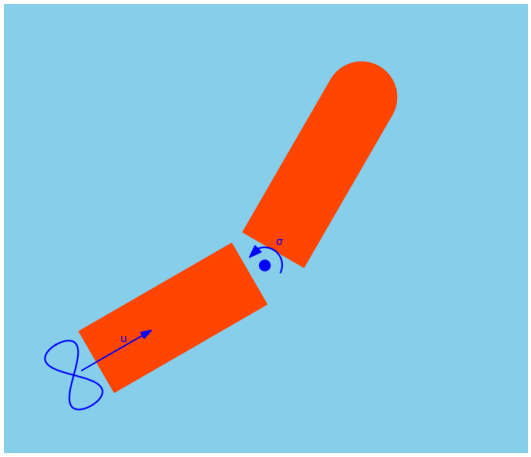


Figure 43: Model of the Robot on the Horizontal plane

Let's first define some variables and constants:

- Variables:
  - (x, y) : the position of the robot in the horizontal plane (position of the servomotor)
  - v : the forward speed, in the direction of thrust
  - ψ : the yaw
  - σ : the angle of the servomotor
  - u_1 : the input of the thruster
  - u_2 : the input of the servomotor
- Constants:
  - k_p : the motor constant that links the command to the induced force
  - k_1 and k_2 (and K): are fluid friction constants dependent on the robot geometry $(1/2 * \rho * S * Cx)$

Then, we define the state vector and the horizontal evolution equation as:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ v \\ \psi \end{pmatrix} \quad ; \quad \dot{\mathbf{x}} = \begin{pmatrix} v \cdot cos(\psi) \\ v \cdot sin(\psi) \\ k_p \cdot u_1^2 - k_1 \cdot v^2 - k_2 \cdot v^2 \cdot cos(\sigma) \\ k_2 \cdot v \cdot sin(\sigma) \end{pmatrix}$$

And in the same time we have $\sigma = u_2$
σ cannot be considered as a state variable as it is defined cinematically and not dynamically...

Explanation :
Using the Newton's second law of motion we have :

$$\sum F_{xy} = F_u + fr \tag{6}$$

With :
F_u the force induced by the motor $F_u = k_p u^2$
and fr the fluid frictions force $fr = Kv^2$

For this control, we consider that a fraction of the fluid friction force fr is applied depending only on the forward speed v of the robot and that the other part depends bothe on v and on the angle of the servomotor σ. The first part is always applied in the opposite direction of the thruster and the friction fraction is applied perpendicularly to the front module.

If we project the forces on the advancement direction and on the perpendicular direction, we have :

$$\begin{aligned} &F_u - k_1 v^2 - k_2 v^2 cos(\sigma) \quad \text{in the direction of advancement} \\ &\text{and} \\ &k_2 v^2 sin(\sigma) \quad \text{in the perpendicular direction} \end{aligned} \tag{7}$$

The second force generates a torque :

$$L k_2 v^2 cos(\sigma) \quad \text{with } L \text{ the leverage} \tag{8}$$

Which leads to the following equation :

$$\dot{\psi} = k_2 \cdot v \cdot sin(\sigma) \tag{9}$$

In order to keep having a behaviour stable and close to the one considered here (among other reasons), we limit the angle of the servomotor between $-\pi/6$ and $\pi/6$. *(let's note that v would be more relevant here but we considered v so far so we'll keep v)*

9.3.2 Horizontal controller

To facilitate the **guidance** of the robot inside the karst, we want to provide a controller that transforms the behaviour of the robot in the horizontal plane as if it was a dubins car.
To achieve this, we have to transform the actual inputs of the robots into a desired forward velocity v_{target} and a desired speed of rotation ω_{target}.

To control the speed, we use a controller based on the error $v_1 = v_{target} - v_{actual}$ with v_{actual} estimated at all times using the equations of motion (which is relevant as the navigation

is stable). The angle σ is also known at all times through the feedback of the servomotor. We calculate the input u_1 with the following formula :

$$u_1 = \sqrt{v1 + k_1 \cdot v^2 + k_2 \cdot v^2 \cdot cos(\sigma)}/k_p \tag{10}$$

This controller may appear a bit complex but it showed better performances than a classic proportional.

To control the speed of rotation, we use a different controller based on the special status of σ (both an input and a geometrical parameter involved in the dynamic evolution). We calculate the input u_2 as so :

$$\begin{cases} u_2 = arcsin(\omega_{target}/(k_2 \cdot v) & if v > |\omega_{target}/(k_2 \cdot v)| \\ u_2 = \sigma + \delta_\sigma & else \end{cases}$$

with δ_σ a small variation of angle in the direction we want to turn
Whereas this controller may seem not linear and unstable (because of the arcsin), in fact :

- the condition is just here to ensure that the forward velocity is enough to turn at the target speed ($\omega target$) while having σ bounded.
  In reality it constitutes just a starting condition to avoid any instability at low speed.
- with σ in $[-\pi/6, \pi/6]$ and a realistic speed (<10m/s), the arcsin is almost linear
- the only constraint that we have on v and ω is $\omega \leq (k_2 \cdot v)/2$ which not very constraining as we should be able to rotate at at least $\pi/3 rad/s$ with a forward velocity of $1 m/s$.

9.4 Hardware and Drivers

Work with groups Ballast and electronics All actuators must becontrolled by **PWM** signals. We chose to generate them using an micro-controller **ESP32**. The control commands are communicated to the **ESP32** via a **UART** serial connection from the onboard computer **Raspberry Pi**.

9.4.1 Raspberry Pi Configuration and ESP32 Configuration

The Raspberry Pi is configured with **Ubuntu 22.04 server**. **ROS2 Humble Base** is installed and set up on it. To establish the serial connection, the Python **Serial library** is installed. An **ROS2 WORKSPACE** is set up and source and follow the Control Architecture[figure 37].

9.4.2 Serial Link between Raspberry Pi and ESP32

To establish the serial connection, a ROS2 node was coded to retrieve the control commands calculated by the regulators in the control node and send them to the ESP32. Hence, we encode four floats in our connection. The ESP32 decodes these four floats at the beginning of its loop. A serial link using **UART** (Universal Asynchronous Receiver/Transmitter) is a form of serial communication commonly used in microcontrollers and computers. **UART** communication involves two lines for transmitting data **Tx** and receiving data **Rx**. It's asynchronous because there is no shared clock signal between the two devices, which means they must agree on the data rate (baud rate), data bits, stop bits, and parity beforehand to communicate properly. Our UART link Configuration:

- **Data Format:** Each float is typically represented as a 32-bit binary number in most systems. In UART communication, each of these bits would be sent sequentially along with start, stop, and optional parity bits.
- **Start and Stop Bits:** Each packet of data starts with a start bit (0) to signal the beginning of a new byte, followed by the data bits (in this case, the binary representation of a float), and one stop bits (1) to signal the end of the byte.
- **Baud Rate Synchronization:** The Raspberry Pi and the ESP32 are configured to the same baud rate (115200 bits per second) to understand each other.

Although the ESP32 can send data back to the Raspberry Pi, for the momemnt, it's not needed.

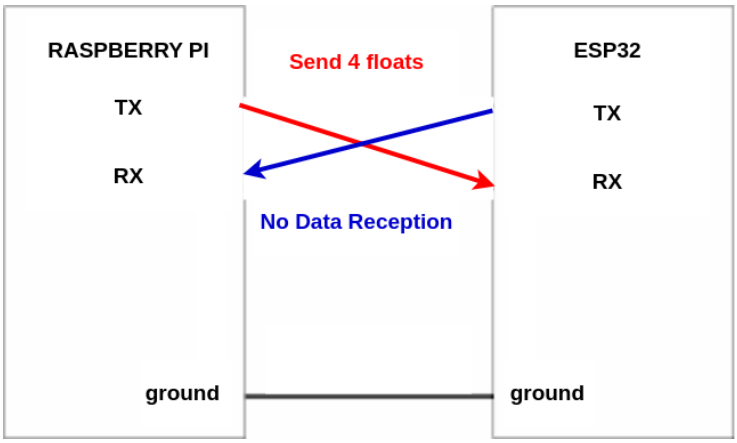


Figure 44: Uart Diagramm

9.4.3 Servo Driver

The **servo driver** retrieves a command value for angle between **0** and **180** degree. It remaps this to a pulse range between 1000 and 2000ms. The resolution is 16 bits, and the period is 20ms.

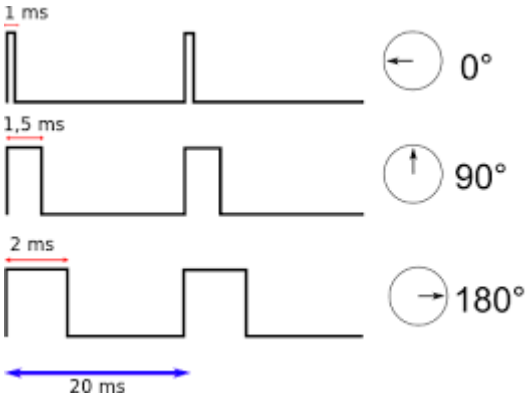


Figure 45: servomotor PWM

9.4.4 T200 Motor Driver

The **T200 Motor Driver** retrieves a command value for speed between **-100** and **100** (speed percentage). It remaps this to a pulse range between 1000 and 2000ms. The resolution is 16 bits, and the period is 20ms.

9.4.5 Motor Ballast Driver

Similar to the **T200 motor driver**, the **Motor Ballast Driver** accepts a command value for speed within the range of **-100** to **100**, representing the speed percentage. The pulse width modulation (PWM) signal have the same range with the T200 motor driver.

This approach ensures that both drivers can be integrated seamlessly into the control system, providing a uniform method for manipulating the speeds of different motor types within the system.

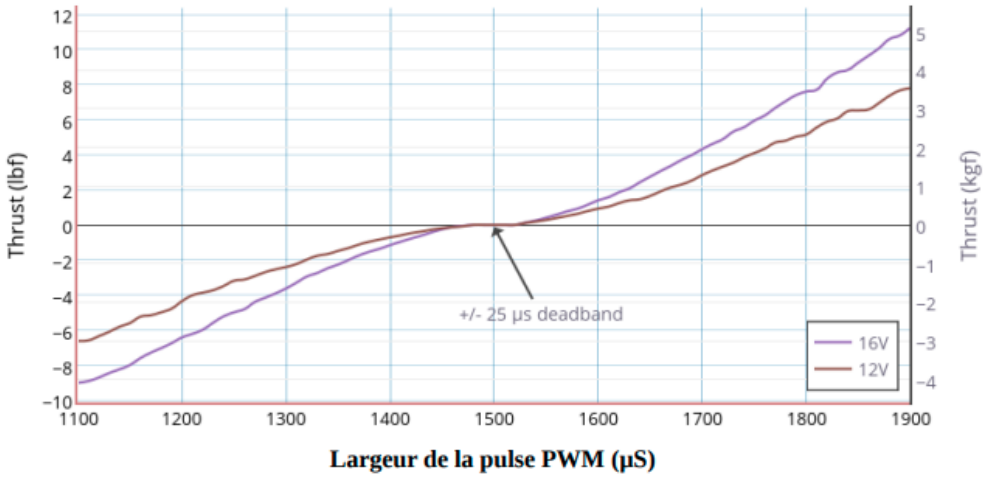


Figure 46: T200 PWM Control Diagramm

9.5 Speed performance approximation

In this part we try to give order of magnitude of the robot max forward speed. Let's consider a 2D plane. If the robot is moving straight with no angles between the two parts, we can approximate its dynamics saying it is subject to two forces : thrust given by the propeller and drag resistance.

$$ma_x = Thrust - \frac{\rho S C_x}{2} v^2 \tag{11}$$

with $\rho = 1000\ kg.m^{-3}$, S the surface area in front of fluid movement, here it is a circle : $S = \pi r^2$. And C_x the Drag coefficient which is around 0.5 for a circle (surface that faces the fluid movement).

We have reached the max speed when the drag force compensate the thrust given by the propeller, so when $a = 0$, which leads to :

$$Thrust - \frac{\rho S C_x}{2} v_{max}^2 = 0 \implies v_{max} = \sqrt{\frac{2 * Thrust}{\rho S C_x}} \tag{12}$$

We use the T200 data sheet to have the thrust as a function of PWM sent, and we can plot the max speed estimated for each PWM sent :

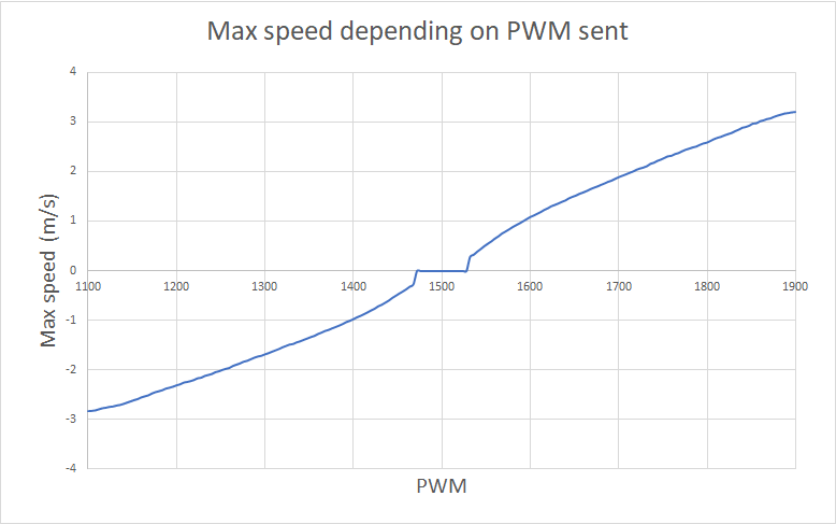


Figure 47: Max reachable speed depending on PWM sent

9.6 Pilot test

We conducted a test to control all the actuators by sending them commands via the Raspberry Pi. Here is the link to the test video https://www.youtube.com/watch?v=3rpy5u1xWQo.

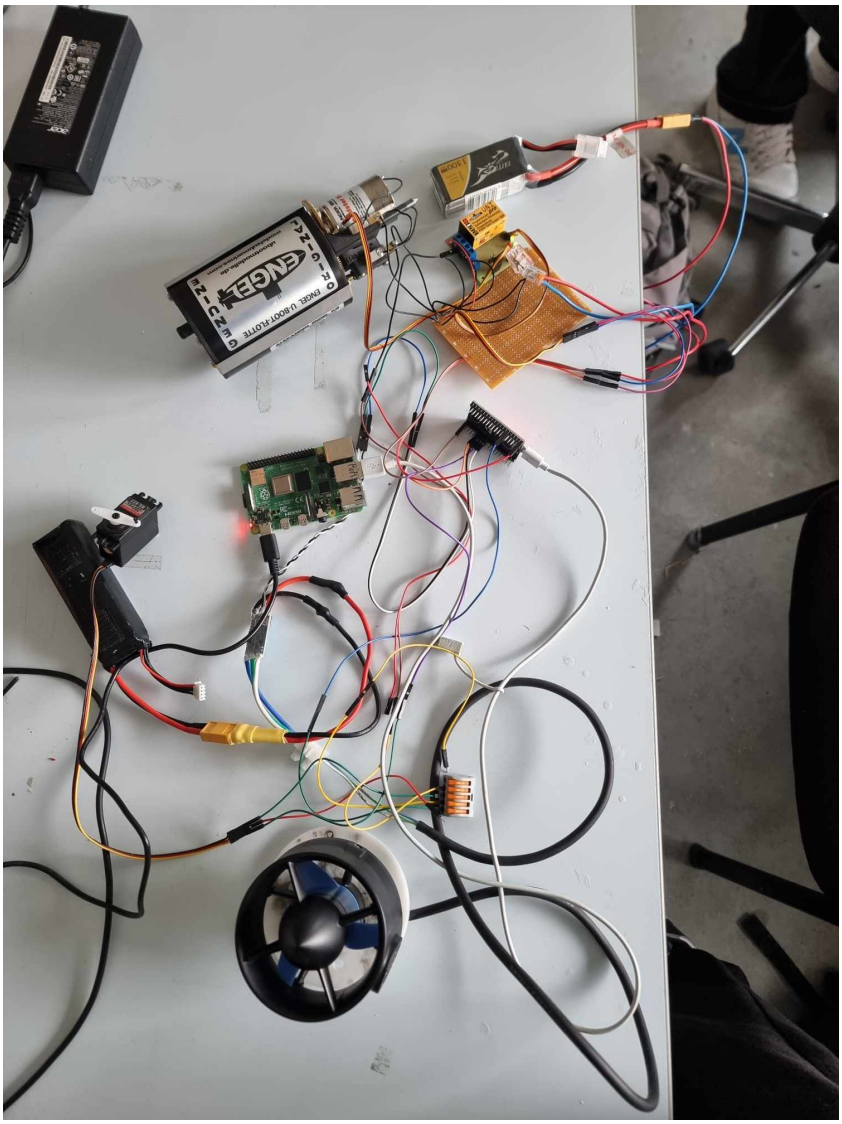


Figure 48: Pilot test

10 Optical guidance

Students : Gabriel Betton, Hugo Hofmann, Kevin Ren

10.1 Objective and system integration

While the robot is equipped with sonars that will prove very useful to detect obstacles and follow the tunnel, it is still wiser to use as much information as we have, and with this in mind using the camera placed at the front of the robot is particularly interesting. The camera model and its position on the robot have been meticulously discussed with the **3D reconstruction** group. In this section we will present two methods for visual guidance :

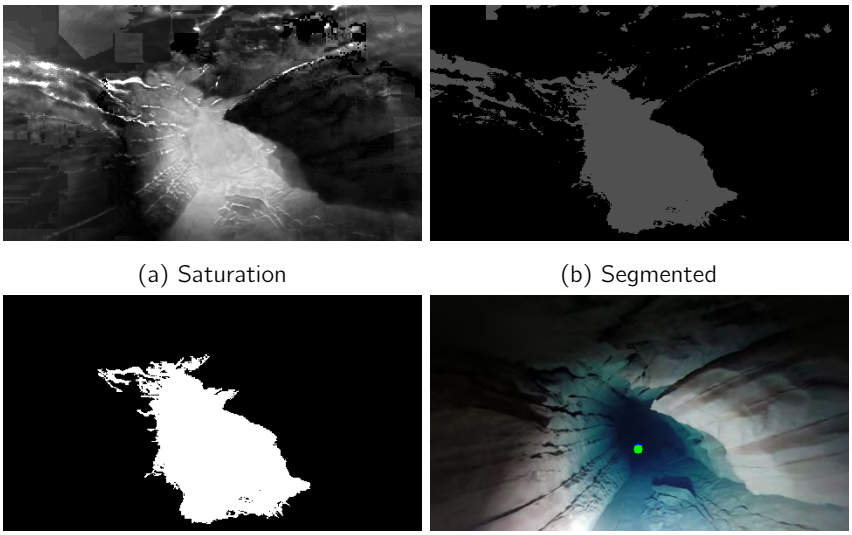
- using image segmentation
- using an optical flow method

These two approaches share the same goal, that is to compute the coordinates of the tunnel center, which will then be used by the **Guidance** team to keep the robot centered along the karst.

10.2 Image segmentation

To obtain the center of the karst using image segmentation, we follow theses steps for each frame :

- Preprocessing : the image is converted to HSV (Hue Saturation Value), the best results were obtained by converting the image into a gray scale of its saturation (Figure 49a). Then the image is vectorized and sampled to keep 1/1000 of its values, the goal is to lighten the processing time of the next operation.
- Segmentation : a bayesian gaussian mixture model is trained on the vectorized image to split the image in clusters, the weights of the previous frame are used as the starting point to ensure more continuous results (Figure 49b).
- Postprocessing : the segmented image is converted to blobs to unify the result, then the largest one is kept (Figure 49c) to compute its centroid corresponding to the desired center of the karst (Figure 49d).



(a) Saturation (b) Segmented

(c) Largest blob (d) Result

After various optimization the results are consistent and the algorithm takes around 50ms per frame (tested on a Ryzen 5 7530U, 6 cores, 2.0 Ghz). However, this method lacks reliability, for certain geometry of karst or ligthnings, the results can be less precise.

10.3 Optical Flow

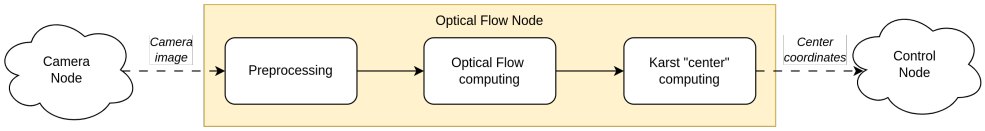


Figure 50: Optical flow node structure

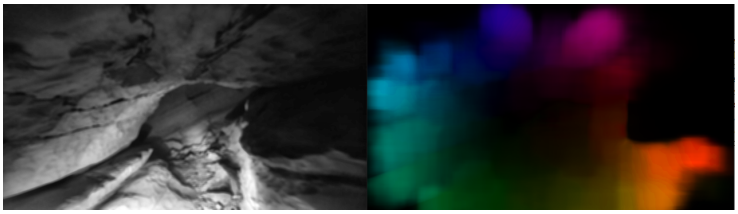


Figure 51: Computed optical flow (right) from a karst video (left)

Now that the optical flow has been computed, we need to find a method to use the information to compute the center of the karst as accurately and reliably as possible, which will then be sent to the **guidance team**'s node. Several methods were developed and tested, of which we will present here the two main ones.

10.3.1 Convolution method

Using the optical flow image, we would like to compute an estimation of the center of the karst. Fortunately, videos tend to show that generally speaking the karst does keep a cylindrical

shape, which yields a characteristic circular flow, as seen in Figure 51. This means that we can use this general shape to determine where (or at least, in what direction) the center of the tunnel is, defining a reference profile (or kernel) as seen in Figure 52. The chosen kernel represents what we expect the camera to see when the robot is centered in the tunnel and looking straight ahead. Using this, we can then compute an estimation of the center of the karst using a 2D convolution on the image. The location of the maximum value within the result of the convolution should correspond to the center (see Eq 13).

(a) x component of the reference kernel (b) Reference kernel

Figure 52: Karst reference kernel

$$\boxed{(x_0, y_0) = arg\max_{x\in[0,w],y\in[o,h],\epsilon_i\in\mathcal{D}_i}(T(x,y) * P(x,y,\epsilon_1,...,\epsilon_n))} \quad (13)$$

Results can be seen in Figure 53 and here: https://youtu.be/E9AmsiBS_N0

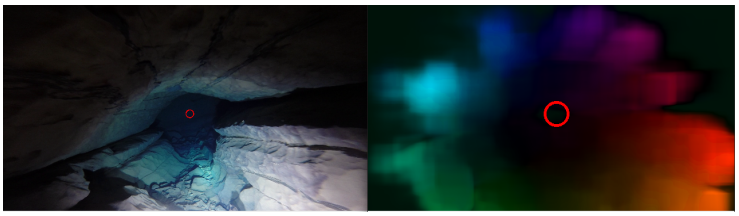


Figure 53: Computed center (red circle) obtained with the convolution method

10.3.2 Depth Map

The aim of this part is to give a method for estimating the distance of a point in an image via optical flow and the approach speed of the camera which are needed by the **guidance team**. Let M be the point representing the AUV that can be controlled like a Dubins car and P a point in the environment. This point P is visible with the robot's camera. The robot advances at a speed v which is tangent to its orientation (Dubins). By relativity, we can consider that it is the point P which advances towards the robot. We then index by n the different positions of point P at time n. We then obtain the following figure:

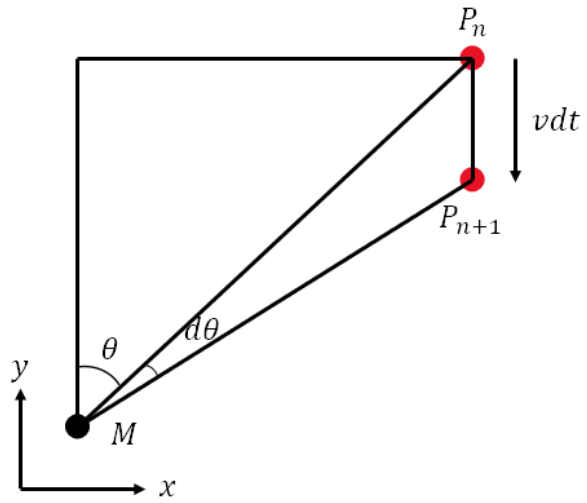


Figure 54: Relative displacement of point P compared to M

Let $\overrightarrow{MP_n} = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$, $\theta_{n+1} = \theta_n + d\theta$ and $d = \|\overrightarrow{MP_n}\|$. Then $x_n = x_{n+1}$ and $y_n = y_{n+1} + v\,dt$.
We have :

$$\tan(\theta_n + d\theta) = \frac{x_n}{y_n - vdt} \tag{14}$$

$$vdt = y_n - \frac{x_n}{\tan(\theta_{n+1})} \tag{15}$$

$$= d\left(\cos(\theta_n) - \frac{\sin(\theta_n)}{\tan(\theta_{n+1})}\right) \tag{16}$$

From here we are able to estimate the AUV's velocity with sonars if other solutions do not allow it by considering that P is the point targeted by the sonar:

$$\boxed{v = \frac{1}{dt}\left(y_n - \frac{x_n}{tan(\theta + d\theta)}\right)} \tag{17}$$

Where $||\overrightarrow{MP_n}||$ is given by the sonar, θ represents the angle between the direction of the robot and that of the sonar in the horizontal plane, which is known. The change in angle, $d\theta$, is inferred from the optical flow by subtracting $\theta_n = f(u_n)$ from $\theta_{n+1} = f(u_{n+1})$, where u_n denotes the horizontal component of the pixel coordinate of point P (see Equation (19)).

Finally the distance d between the robot and a random point P on the image is given by :

$$\boxed{d = \frac{vdt}{\cos(\theta_n) - \frac{\sin(\theta_n)}{\tan(\theta_{n+1})}}} \tag{18}$$

Let $\theta_n > 0$. The point P is moving towards us, hence to the right, so $\theta_{n+1} > \theta$. We can restrict ourselves to $\theta < \frac{\pi}{2}$ since the field of views (FOVs) of typical cameras are less than 180°. Under these conditions, we can show that $\cos(\theta_n) - \frac{\sin(\theta_n)}{\tan(\theta_{n+1})} > 0$ and $\frac{\sin(\theta_n)}{\tan(\theta_{n+1})} > 0$. Thus, $\cos(\theta_n) - \frac{\sin(\theta_n)}{\tan(\theta_{n+1})}$ increases with θ_{n+1}. Consequently, d decreases with θ_{n+1}, which is logical since a point that is closer moves faster. The same reasoning can be applied for $\theta < 0$.

Let's now demonstrate the relationship between an angle θ and u. Let f be the focal distance, i.e., the distance between the optical center and the image plane for the pinhole model. We know there exists a constant $k \in \mathbb{R}$ such that $u - c_x = kf\tan(\theta)$ where c_x corresponds to the image center in pixels along the x-axis. Given the camera's field of view (FOV) and the width l in pixels of the image, we have $\frac{l-c_x}{\tan\left(\frac{\text{FOV}}{2}\right)} = kf$.

Finally :

$$\boxed{\theta = \arctan\left(\tan\left(\frac{FOV}{2}\right)\frac{2(u-c_x)}{l}\right)} \tag{19}$$

By denoting $(u(M), v(M))$ as the coordinates of M in the image frame, we have:

$$u(M_{n+1}) = u(M) + \overrightarrow{flux}(u(M_n), v(M_n)) \cdot \vec{e_u}$$

Thus, we can determine θ_{n+1} from the optical flow with equations (18) and (19), which allows us to conclude that for every pixel in the image where the optical flow is non-zero, we can associate a distance d.

On figure 55 is presented above the image which was used for the optical flow. Below is plotted the curve representing the distance of each of the pixels of the black line, drawn in figure above, relative to the position of the camera. The distances have been capped. Which means that all distances greater than d_{max} will be set to d_{max}. In particular the areas where the distance calculated with the method described is infinite because the optical flow is zero have been limited. In order to distinguish these d_{max} values from the real ones, these outlier values were set to $0.9d_{max}$ instead. The result is rather satisfactory and seems to depict the appearance of the karst quite well. The figure 56 shows the depth map of the image, the lighter the color of a pixel, the farther away the pixel is. The result confirms the previous observation. However, a discontinuity appears approximately in the middle of the image separating the image in two. This discontinuity is also observed in figure 55. The cause of this phenomenon has not yet been found.

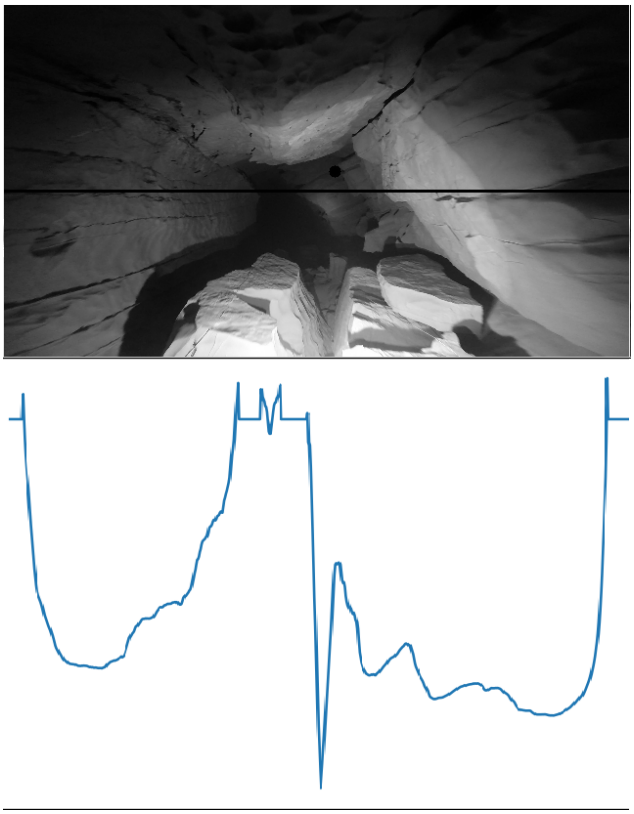


Figure 55: (a) Image used for optical flow (b) Normalized distance estimated for each pixel of the black line drawn on (a) using the method presented

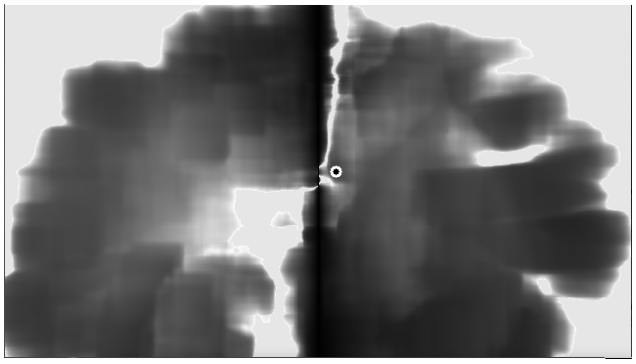


Figure 56: Estimated depth map; The lighter the color of a pixel, the farther away the pixel is

...

11 Guidance

Students : Louis Roullier, Louis Ravain, Joachim Zaffrana, Virgile Pelle, Bastian Garagnon

The goal of this part is to give optimal instructions to the robot in order to make it navigate the karst. Our code outputs are a desired speed (along the longitudinal axis of the robot), a desired rotation speed (around its z-axis) and a desired altitude. Subsequently, these parameters are relayed to the node managed by the **control team**, which in turn translates them into actionable instructions for the actuators. This phase operates within a simulated environment, provided by the **simulation team**. Therefore, we don't need a precise model for the robot. Hence, we've elected to approximate our model to that of a Dubins car, enabling the evaluation of various guidance models. This approach is viable due to the efforts of our colleagues in the **state equations team**, who have established a seamless transition between the Dubins car model and the Karstex model.

Our navigation strategy depends on two essential pieces of information: :

- the center of the karst, given by the **vision guidance team**, who utilize a camera to pinpoint the central location of the karst landscape accurately.
- the distances given by the sonar, thanks to the work done by the **echosounder team**. These measurements help us gauge distances to obstacles and features within the karst environment.

By incorporating these inputs into our navigation framework, we equip our system with the necessary guidance to traverse the karst terrain effectively.

11.1 Use of camera

In conjunction with the **optical guidance team**, we tried to formulate instructions utilizing data from camera sensors. Within the simulation environment GAZEBO, our colleagues of the **simulation team** managed to acquire images. Initially, the exploitation of these images was difficult because the environment was too dark. To overcome this hurdle, a light was installed on the robot, enabling the capture of usable images. An example of an exploitable image is given on the following figure :

Figure 57: Image provided by the simulation environment

This advancement allowed the **optical guidance team** to calculate the coordinates of the pixel of the center of the karst, in the simulated environnment. With this foundation, we proceed to issue instructions to the robot:

- Utilizing a conventional proportional controller, we compute the instruction as the difference between our current linear speed (v) and the desired speed (v_d), with v_d assumed constant in our scenario.

- Firstly, we acquire the center of the image through a ROS2 topic provided by the **simulation team**. Subsequently, the **optical guidance team** employs image processing techniques to determine a barycenter representing the center of the karst : the location the robot should aim for within the image to continue exploring the karst. This process involves calculating the optical flow between successive frames, allowing us to detect significant changes in the scene and potential obstacles near the robot. The objective is to guide the robot towards the position indicated by the ideal pixel. To achieve this, we set a maximum angular velocity (ω_{max}) and introduce a coefficient (C_ω) defined by the following formula:

$$\boxed{C_\omega = \frac{imagecenter_x - barycenter_x}{imageshape_x}} \tag{20}$$

  To ensure precise angular speed control, we utilize the horizontal dimension of the image as the shape parameter and the x-coordinate for the numerator. The instruction for the angular speed required to reach the goal is achieved by multiplying the maximum angular velocity (ω_{max}) by the coefficient (C_ω). This approach facilitates dynamic adjustment of the robot's orientation, enabling it to accurately navigate towards the designated target within the image. Therefore we have :

$$\boxed{\omega_{target,camera} = C_\omega * \omega_{max}} \tag{21}$$

- For altitude control, we employ a similar process as before to determine the objective pixel within the image. To attain the desired altitude, we directly assign the coefficient (C_z) as the instruction. However, in this case, we utilize the vertical coordinate of the picture for the shape parameter and the y-coordinate of the objective pixel for the numerator.

$$\boxed{C_z = \frac{imagecenter_y - barycenter_y}{imageshape_y}} \tag{22}$$

This approach ensures precise adjustment of the robot's altitude, facilitating smooth and accurate navigation within the karst environment. Therefore we have :

$$\boxed{z_{target,camera} = C_z} \tag{23}$$

11.2 Use of Sonars

In addition to the camera, our sensor suite includes three sonars. To ensure the relevance of the data supplied by these sonars, we collaborated closely with the **echosounder team**. Together, we strategized on how best to position them on the robot within the simulation environment. The resulting arrangement is illustrated in the following figure:

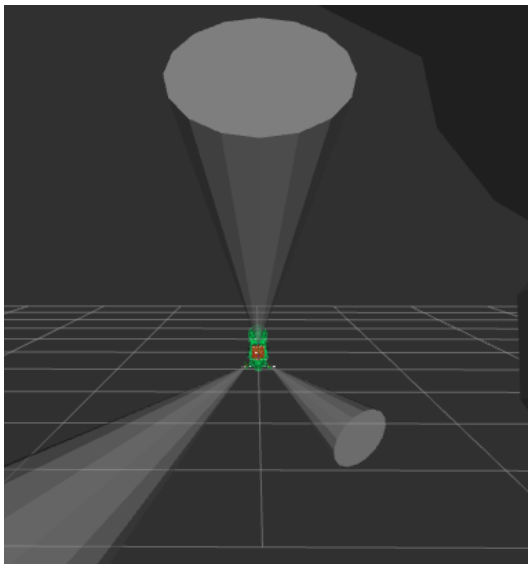


Figure 58: disposition of the echosounders on the robot

Following the setup, our focus shifted to devising appropriate instructions based on the information given by the sonars. Our objective was to consistently maximize the distance measured by all three sonars, indicating that we are near the center of the available pathway. To achieve optimal heading alignment, we employ the following formula:

$$\boxed{\omega_{target,sonar} = K1 * tanh(-3 * (sonar_1 \cdot \cos\left(-\frac{2\pi}{3}\right) + sonar_2 \cdot \cos\left(-\frac{\pi}{3}\right)))} \tag{24}$$

Moreover, for altitude control, we use the depth data given by the **simulation team** (accessible through the topic /depth). Additionally, we harness information from the sonars. Consequently, we can employ the following formula:

$$\boxed{z_{target,sonar} = depth + (sonar_1 * sin(7 * \pi/6) + sonar_2 * sin(11 * \pi/6) + sonar_3)} \quad (25)$$

11.3 Global guidance

Combining all the precedings informations, we now possess :

- Three inputs from the exploitation of the camera: a linear speed, an angular speed, and an altitude.
- Two inputs from the exploitation of the sonars: an angular speed and an altitude.

Our approach involves utilizing both sets of controllers concurrently. In essence, we compute the average of the angular speed and altitude provided by our two controllers, weighted by their reliability index ($i_{w,sonar/image}$ and $i_{z,sonar/image}$). This integrated approach ensures a comprehensive and balanced navigation strategy, leveraging the strengths of both sensor modalities to guide the robot effectively through the karst environment.

The final ω_{target} and z_{target} are given by these formula :

$$\begin{cases} z_{target} = \frac{z_{target,sonar} * i_{z,sonar} + z_{target,camera} * i_{z,image}}{i_{z,sonar} + i_{z,image}} \\ \omega_{target} = \frac{\omega_{target,sonar} * i_{w,image} + \omega_{target,camera} * i_{w,sonar}}{i_{w,sonar} + i_{w,image}} \end{cases}$$

11.4 ROS2 architecture

Now that we have our ω_{target} and z_{target}, we have to make them accessible to the other teams, especially the **control team**. The following graph shows how and where we take our inputs, and where we output the results of our node.

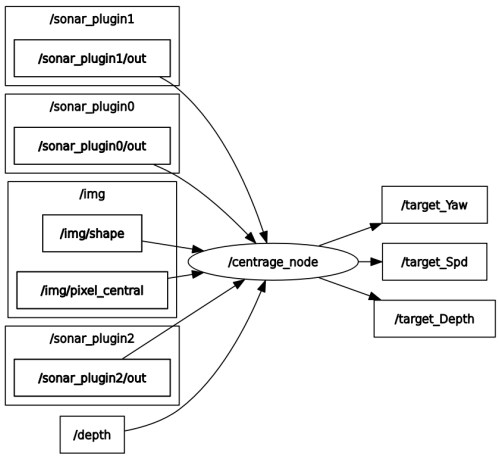


Figure 59: Graph of the ROS2 architecture

12 Simulation

Students : Clara Gondot, Théo Massa, Ludovic Mustière, Etienne Roussel et Apolline de Vaulchier

12.1 Simulation environment

In this type of robotic project, having a simulation that allows to do a first validation of our architecture is essential. Simulating the AUV and its environment allows us to try models, algorithms or the feasibility of the project before even having to try on a real submarine. Even more if we consider the context of this project, which is to explore karsts, natural cavities that are not easily accessible, especially around Brest.

In order to obtain a convincing simulation, we had several options. We could have done a simulation using a simple language like Python, but we thought that simulating every sensor and the interaction between the environment and the Karstex sensors was not going to be easy in this language. Another option was to use a visual engine like Unreal Engine, but we have no knowledge of how to use this kind of tools to recreate a virtual world. Finally we opted for the common robotic way, which is to use Gazebo, a simulator widely used in robotics. This simulator has the advantage of having many libraries that simplifies, in theory, the creation of a virtual environment and, of course, a virtual robot.

With this in mind, we began to work on this simulation. For this, we had to work step-by-step. The first one was to have the environment in which the AUV evolves implemented in the simulator. Thanks to a mesh file of a karst's cartography made during previous explorations, we were able to have a simulated representation of a real karst in our simulated world.

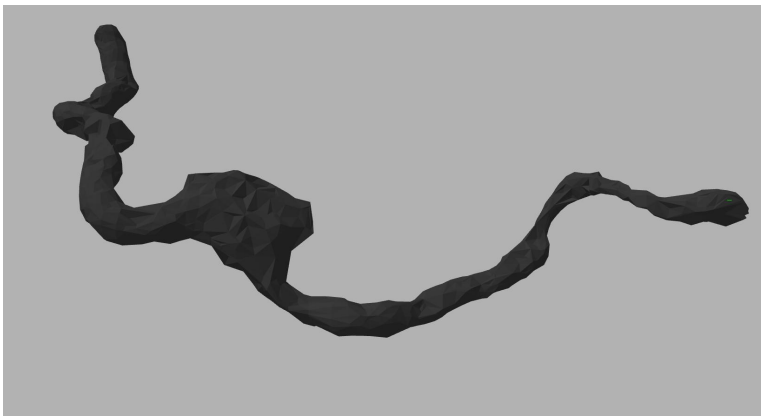


Figure 60: Fontanilles' karst 3D model

Once this was done, we had to create a virtual representation of the AUV's body. For this part, we worked in collaboration with the group that was in charge of modelling the robot and its mechanic conception, also called the **mechanics group**. In order to be the closest to the reality, the simulated AUV's dimensions have to be close to the real one's. In Gazebo, we have to use what is called an URDF (Unified Robotics Description Format) file to describe the

geometry of the robot. In this URDF file, we define a collection of parts and joints, describing the geometry of the blocks, the joints between them, their type, limits, etc. To simplify the processing cost of the simulator, the description relies on simple shapes to compute volume, collision or inertia. In our case, the Karstix is just two cylinders linked with a revolute joint in between.

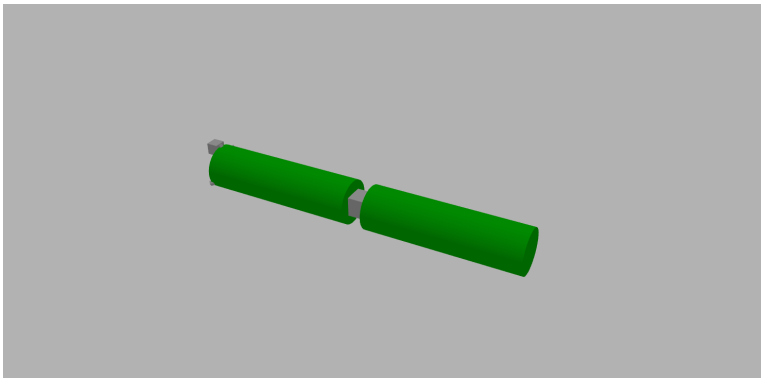


Figure 61: Simple version of the AUV

In theory, just this simple version should already be enough to simulate a mission. However, it is possible to add a visual layer to this simple but effective model. By using a stl file that the **mechanics group** has given to us, we gave the simulated AUV a realistic aspect, except for its color.

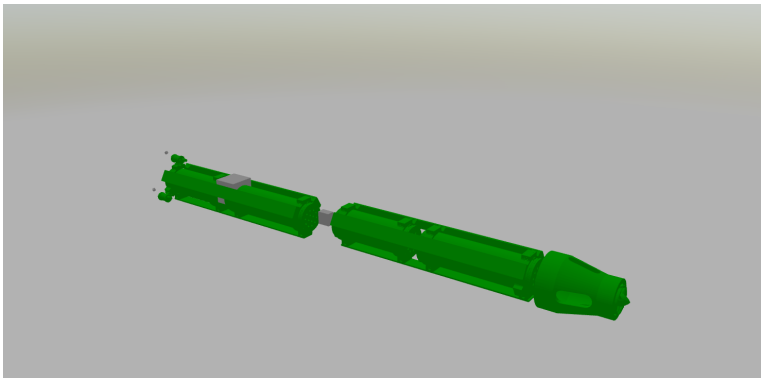


Figure 62: Visual version of the AUV

The next step is to implement the physics of the environment and the simulated AUV. This part was the most challenging one. Indeed, Gazebo already has a physical engine in order to simulate typical physical phenomenons (gravity, friction, actuators thrust etc). However, we want to control fully our simulation and it is not especially a good idea to rely on the software physics, considering we do not know fully what's behind. Furthermore, Gazebo is good at simulating robots that evolves on land or maybe in the air, but the libraries that allow to manipulate underwater robots are not always working and are quite obscure. We then decided to reimplement ourselves the physics of the AUV by creating a custom Gazebo Plugin that

will give us full control on the physics. Writing a custom Gazebo Plugin is a task worthy of the worst means of tortures, for which the documentation is really sparse or even non-existent. However, after losing tears and blood, we managed to get functional physics for the simulation.

12.2 State evolution model

Having a functional dynamic model for the simulated Karstix is the main point of the simulation. We have to make a compromise between the precision of the model (which augment the cost of the simulation), and simplifying assumptions for the implementation that still allow us to validate a model. The main challenge of this aspect is that it depends a lot of the results given by the **group in charge of state model**. We had to work with them often, in order to always have the last functional model. The problem is that we could have a full, working and accurate enough model only near the end of the project, so not many tests could have been done with this simulation.

One important aspect is that, in the simulation, we consider not related the state equations concerning the ballast and the rest. It means that the karstex is supposed to stay horizontal thanks to the pitch regulation so that the state model related to the linear movement of the robot is simplified.

See *Section 7: State equations* to get the model state used in the simulation concerning the movement in the plan.

See *Section 8.3: Vertical regulator* to get the model state used in the simulation concerning the ballast.

In accordance with the two sections above mentioned, the Gazebo Plugin is subscribed to 4 ROS topics, each corresponding to a given control input. The 4 inputs control the thrust of the motors, the rotation of the revolute joint linking the two parts of the Karstix and the volume of water inside the body of the front or back ballasts. Thanks to those control inputs and the state model, we are able to control the simulated model directly in speed, which allows to avoid an integration step.

12.3 Architecture of the simulation

The simulation uses the ROS (Robot Operating System) framework. We first created a package only focused on generating the simulated environment called *mnt_fontanilles* (mnt stands for "Modèle Numérique de Terrain", which is the French name for a Digital Elevation Model). This package contains the sdf file that describes the karst, the launch file that starts the simulation and the world file that contains the description of the world in which the AUV evolves. We chose to add a ground plane under the karst to avoid the AUV to fall indefinitely if it goes too low.

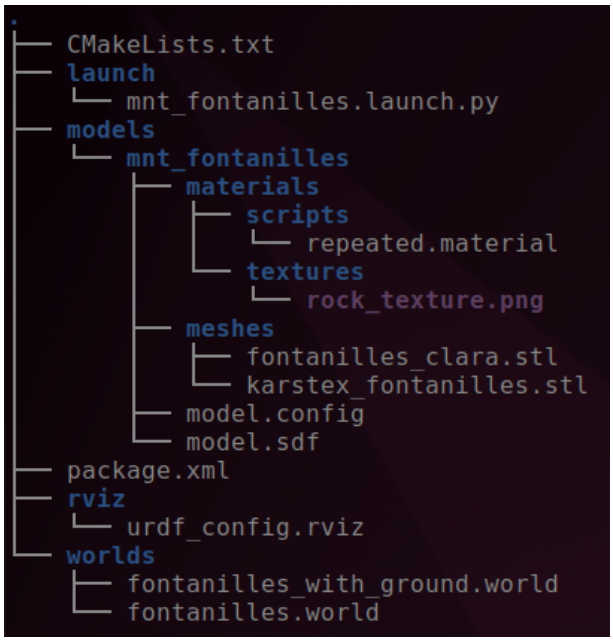


Figure 63: Package Architecture

The second package is called *karstix*. It contains the URDF file that describes the AUV, the launch file that starts the simulation and launches the package *mnt_fontanilles*. It also launches the Gazebo plugin that simulates the physics of the AUV. Inside the *karstix* package, we have the *urdf* file that contains the structure of the AUV and all the sensors implementation. It loads the meshes and textures of the AUV and the sensors and makes the link between the Gazebo environment and the ROS environment.

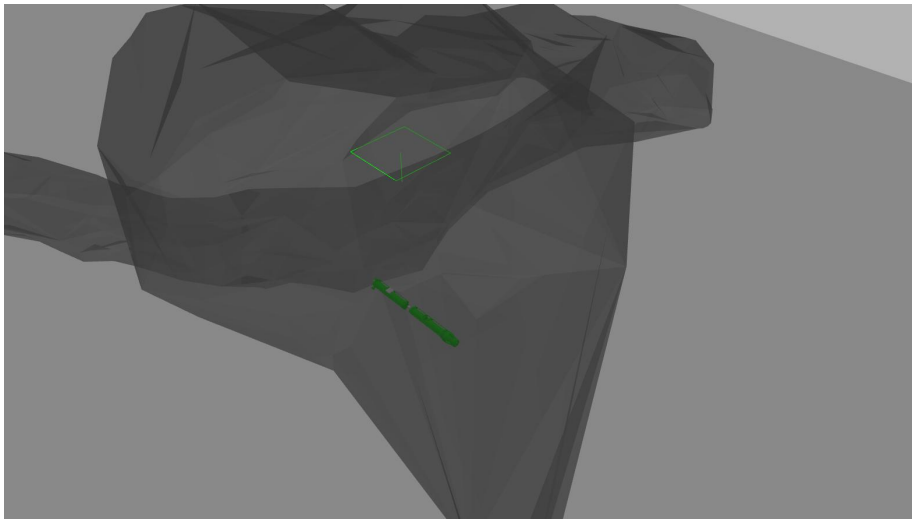


Figure 64: Fontanilles' karst 3D model and the AUV

12.4 Simulating sensors

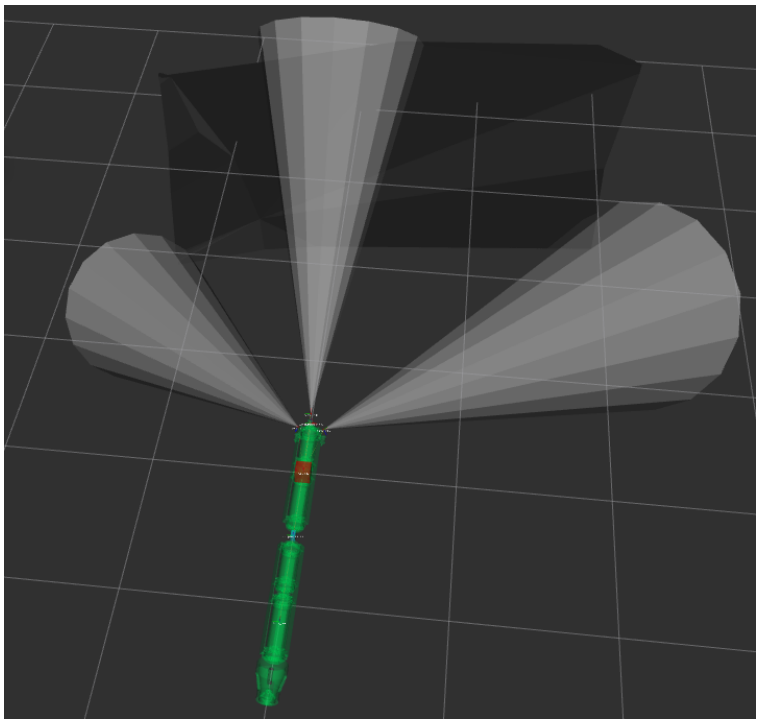
The sensors taken into account in the simulation are: the inertial measurement unit (IMU), the barometer, the camera and the sonars. They are coded in the URDF file associated with the AUV.

The inertial unit is simulated using the *libgazebo_ros_imu_sensor.so* plugin, and is located in the center of the AUV's front body. It publishes the angles, linear accelerations and angular speeds of the robot on the */sensors/imu/imu_data* topic, simulating 3-axes magnetometers, accelerometers and gyrometers. Gaussian noise was added for each measurement to simulate the noise of the real sensors.

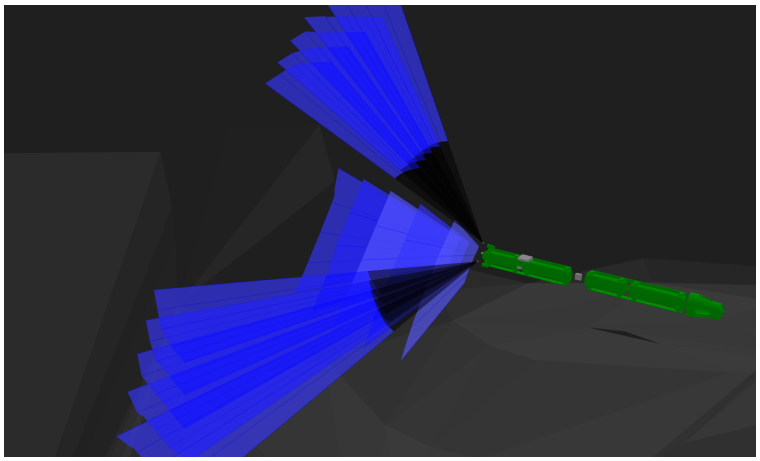
The barometer is not simulated with a plugin unlike the other sensors. We use it to obtain the depth of the AUV. Thus, the AUV height is retrieved directly in the Gazebo Plugin used to compute the state evolution (described in the last section), then Gaussian noise is added to reproduce the noise of a real sensor to finally publish it on the */sensors/pressure_sensor/depth*.

The camera is RGB-D for depth information. It is simulated using the gazebo plugin *libgazebo_ros_camera.so* and is located at the front, as on the real AUV. It publishes the disparity map and the corresponding point cloud on the */sensors/camera/image* and */sensors/camera/depth_map* topics. In reality, the disparity map and point cloud would be computed using computer vision first, and a lamp would be paired with the camera to light up the scene. In the simulation, we have chosen to illuminate the entire karst to take this lighting into account, but the texture and contrasts seen by the camera are still too low to use computer vision. We tried to setup the virtual camera with parameters coherent with the one that will be used on the real AUV, thanks the information given by **the group in charge of 3D reconstruction**.

There are three sonars. The choice of their position, orientation and range was discussed with the **group in charge of the echosounders** to match the actual AUV. The Gazebo plugin used is *libgazebo_ros_ray_sensor.so* and they publish the distance associated with each sensor on the */sensors/sonar/sonar1*, */sensors/sonar/sonar2* and */sensors/sonar/sonar3* topics.



(a) RVIZ



(b) Gazebo

Figure 65: Visualization of sonar and camera simulation on RVIZ (a) and Gazebo (b)

As can be seen from the figure 65, the sonars are positioned well forward with three different orientations. The RVIZ image 65a shows the point cloud (shaded in front of the AUV) obtained by the depth camera corresponding to the karst walls. The red block visible on the RVIZ image corresponds to the IMU. The AUV and the shadows of the karst walls can be seen thanks to the light added to the simultaion.

The final product of the simulation was made available to the other groups, especially the ones in charge of the control and guidance of the Karstix, along with how to retrieve every sensor's data and how to control the robot.

13 3D Reconstruction

Students : Rizk Catherine, Betton Gabriel, Goux-Gateau Adam, Garde Guillaume, Bellot Victor

13.1 The stereo camera

The purpose of the 3d reconstruction is to enable the spatial environment of the karst to be viewed once the exploration of the karst has been completed. To reconstruct this space in three dimensions, we needed to estimate the distance from the karst to each of the surrounding walls. The camera best suited to this type of reconstruction is the stereo camera, which enables us to obtain these distances and thus easily reconstruct a 3d environment. The choice of camera is an important one. As it has to be mounted on the robot, it must be able to operate without a graphics card in the system. Our choice is therefore the Intel RealSense D435, which has an integrated graphics card and is suitable for on-board use.

Figure 66: RealSense camera

Setting up the camera involved initially connecting it to a computer using a USB-C cable, with the ultimate goal of integrating it with a Raspberry Pi 4 to transform the camera into an embedded sensor. However, we encountered some difficulties during this process. Despite successfully fitting it into a Blue Robotics waterproof tube and connecting it to the Raspberry Pi, we faced software compatibility issues when running it on the Ubuntu configured in the embedded computer. Even with a Raspbian environment (this tutorial describes the main steps : Raspberry Pi 4 and Intel RealSense D435), we encountered the same problem.

Here is an image depicting the device installed within the tube:

13.2 Reconstruction of the environment

The principle of this reconstruction is based on the creation of a point cloud in PLY format, which can then be visualized using tools such as MeshLab. PLY, or Polygon File Format, is a format for storing an object in the form of a list of polygons. It can also store numerous properties such as color, transparency and texture.

Figure 67: Camera in the Blue Robotics waterproof tube

We have tried a number of different approaches, using different software and different methods, to generate this point cloud. Although there are many 3D reconstruction software packages available today, many of them are expensive or time-consuming to run. We therefore turned to the RealSense camera library, which features functions for generating this point cloud, which we then save as a PLY file. This PLY file is generated by our `camera_node.py` file. A Python module named `open3d` provides several useful functions to treat the `PointCloud2` data published.

Once the point cloud has been saved, all that remains is to display it. To do this, we used MeshLab software, which can open a PLY file and display it in 3D.

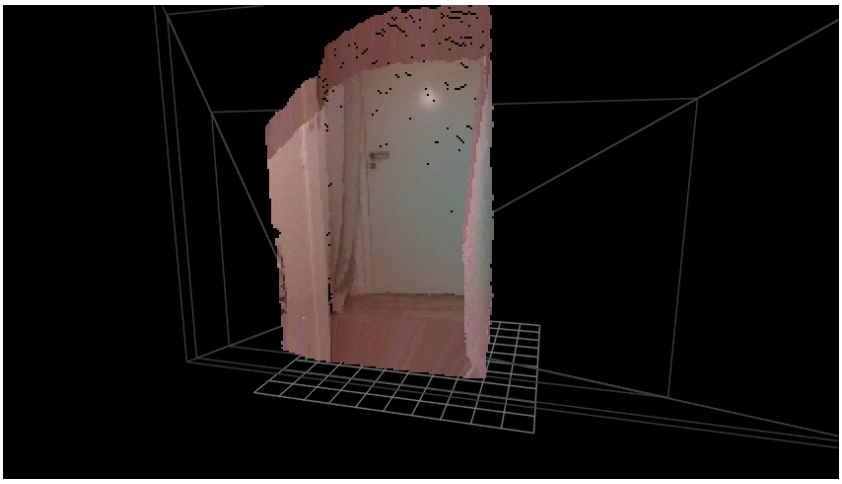


Figure 68: 3D reconstruction of a corridor performed by the camera

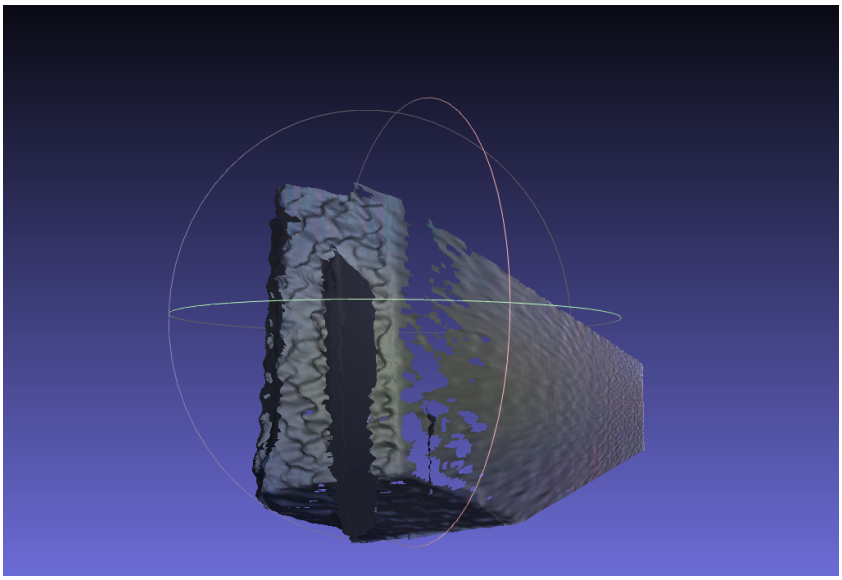


Figure 69: 3D reconstruction of the point cloud representing a corridor in MeshLab

In this way, we can generate point clouds at any time during our AUV's underwater exploration of the karst, so that we can view them after the exploration.

13.3 ROS2 architecture

As we are using ROS2, we required the Intel SDK to assist us in publishing and subscribing to topics. It can be found at : https://github.com/IntelRealSense/realsense-ros.

We followed the provided instructions in the Git to configure the camera to publish a `PointCloud2` message. To do so, we just need to launch the ROS2 node :

```
$ launch ros2 launch realsense2_camera rs_launch.py align_depth:=true
  pointcloud.enable:=true
```

We will not get into details for each possible topic, but we will provide a scheme to illustrate the architecture of our code :

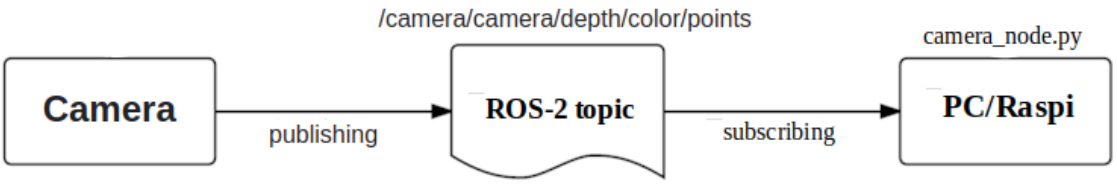


Figure 70: Basic publisher/subscriber architecture for the camera

There are a lot of different topics on which we can subscribe, but the most useful one is `/camera/camera/depth/color/points` : we obtain directly our `PointCloud2` from which we can extract all the information.

13.4 A quick look at an older but efficient way using ROS

For the sake of this project's coherence, the work on 3D vision was performed under ROS2. It must nonetheless be known that 3D reconstruction as well as visual simultaneous localization and mapping (SLAM) with ROS (that is, ROS 1 here) have been well explored. Here follows a list of useful packages that can perform such a task together:

- realsense2_camera
- imu_filter_madgwick
- rtabmap_ros
- robot_localization

The important point here is that the realsense library provides all the tools needed. A simple command can launch the process:

```
$ roslaunch realsense2_camera opensource_tracking.launch
```

It is then possible to play a rosbag and export the 3D point cloud associated with the mission (the RealSense D435i is equipped with its own IMU and GPU).

The difficult part here was to use ROS with our ROS2-equiped computers. For practical reasons, we chose to use Docker to containerize our application. The troubleshooting part was to enable the container to have access to the host's ports and to its displaying functions. We have found a way to do it properly, and we have managed to launch our command. RViz opens, and it is then possible to see the point cloud.

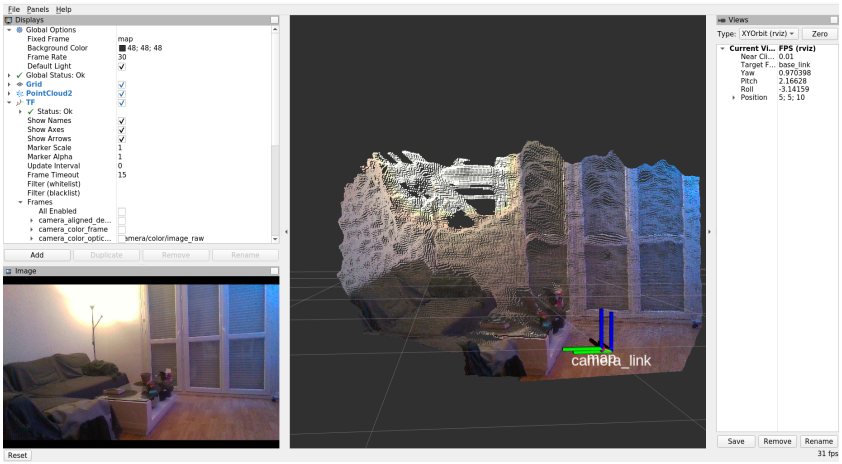


Figure 71: RViz displaying the point-cloud

It takes a little additional configuration to then be able to have a 3D reconstruction.

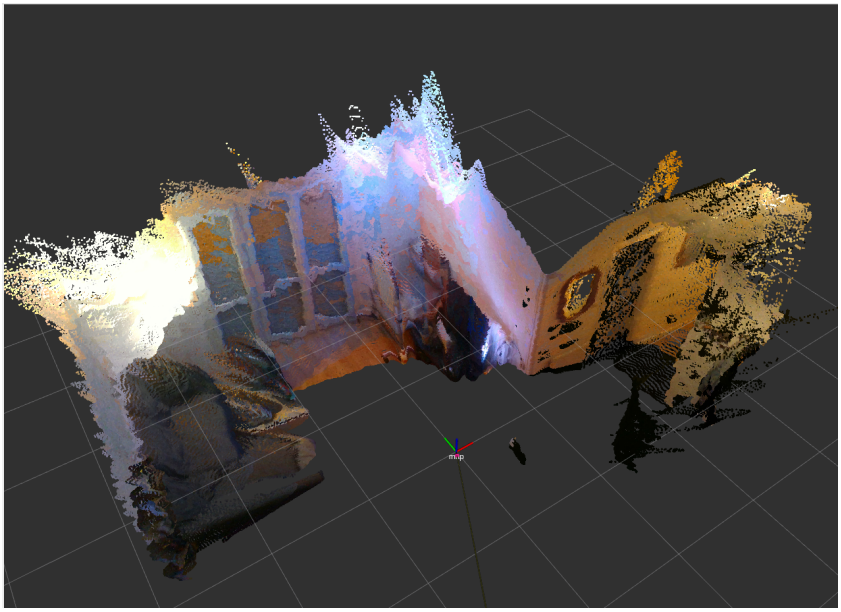


Figure 72: 3D reconstruction with rtabmap

The only thing left to do is figure out a way to save and export the 3D map of the scene with Docker. The main constraint is that it is impossible to send a second command to the container while the ROS process is working. The idea, then, is to work from a second container.

14 GitLab and Software Architecture

Students: Martin Galliot, Guillaume Garde, and Théo Massa

You can find the Git here

14.1 Git Organization

The use of Git has proven to be an essential tool for effective code management and coordination among different groups. Each group was responsible for a specific part of the project, corresponding to a dedicated folder in the Git repository.

Each group followed a well-defined development methodology, particularly regarding the use of Git branches. Each group worked on its own namesake branch.

With this setup, members of each group could work in parallel on their respective tasks without fearing interference with the work of other teams. Features were developed, tested, and validated independently before being first integrated into the *develop* branch and then into the *main* branch.

This also allowed for simplified review of each group's work, except for groups where the use of Git was not relevant. A solution of logbooks was proposed to track the progress of each group throughout the sessions, but this solution was ultimately not adopted.

Finally, the overall work was merged into the *workspace* branch in the *ws_karstex* folder. More details are provided in the "Software Architecture" section below.

14.2 Pipeline

This configuration file enables the automatic building of ROS2 packages across different branches of the GitLab repository using GitLab CI/CD. It defines three build jobs that execute under various conditions based on the commit branch name or the pipeline source:

- develop: This job runs when the commit branch name is "develop" or when the pipeline source is a merge request event. It tests the validity of the ROS2 project workspace build.
- main: This job runs when the commit branch name is "main" or when the pipeline source is a merge request event. It performs the same tasks as the preceding job but for the main branch.

This setup allows for flexibility by only executing jobs for the main and develop branches. Consequently, developers can freely create branches to develop new features. Job verification of the workspace's proper functioning will, therefore, be performed only for the most significant branches that consolidate common work.

However, it was not utilized to its full potential because the creation of a common workspace was carried out late in the project.

14.3 Software Architecture

Since each group coded separately, it was important to agree on the names of the ROS2 topics used.

They were defined as follows:

- Sensors: /sensors/sensor_name/data
- Processing: /processing/type/data

Each group developed its own package, which was then imported into the *ws_karstex* workspace defined above.

Here is a general diagram summarizing the software architecture of the robot:

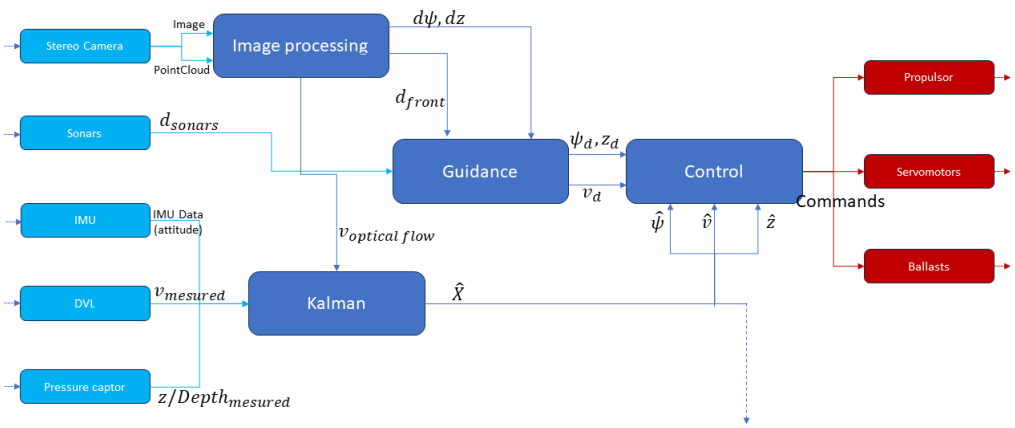


Figure 73: Software Architecture

15 Results

15.1 Deliverable

We have created:

- State and Hydrodynamic Equations: We have modeled Karstex with equations applied to our mechanical architecture.
- ROS2 Drivers for: stereo camera, echosounders, thrusters, servomotors. You can find them on our Git. We have included the exact component for which these drivers work and how to launch ROS2 Nodes. All these drivers are used in the software part.
- Electronic Architecture: The overall architecture, data and power flow study, and PCB design have been completed.
- Mechanical Architecture (CAD): The complete CAD of Karstex can be found on our Git. The mechanical system has been designed to meet the requirements of the underwater environment, electronics (sensor placement, communication between components of two distinct tubes), and hydrodynamic equations.
- Software Architecture: The overall architecture we have created enables each of the software parts, drivers, and intelligence to work together smoothly for seamless assembly. We have also standardized the names of ROS2 topics for project consistency.
- Gazebo Simulation of Karstex: We have also performed a Gazebo simulation of our robot to demonstrate the functionality of the software part. It moves in a 3D representation of the Fontanilles Karst and simulates the sensors we have on the real robot. It reflects the overall system architecture, the drivers created for each component, and relies on the defined state equations.

15.2 To go further...

To progress with this project, the mechanical assembly, as presented in the CAD, needs to be completed, followed by real-world testing of the robot's performance. Initial testing can be conducted in the robotics laboratory pool at ENSTA Bretagne, followed by trials in actual karst environments, such as the Fontanilles karst.

Initially, our project aimed to deliver a physical robot equipped with all implemented components and codes for testing in real karst conditions. However, achieving this goal within a 60-hour timeframe for creating an autonomous underwater robot exploring an unknown and irregular environment seems overly ambitious. Nevertheless, we take pride in our collaborative efforts throughout this project, which have yielded a viable mechanical, electronic, and software solution for exploring such challenging environments.

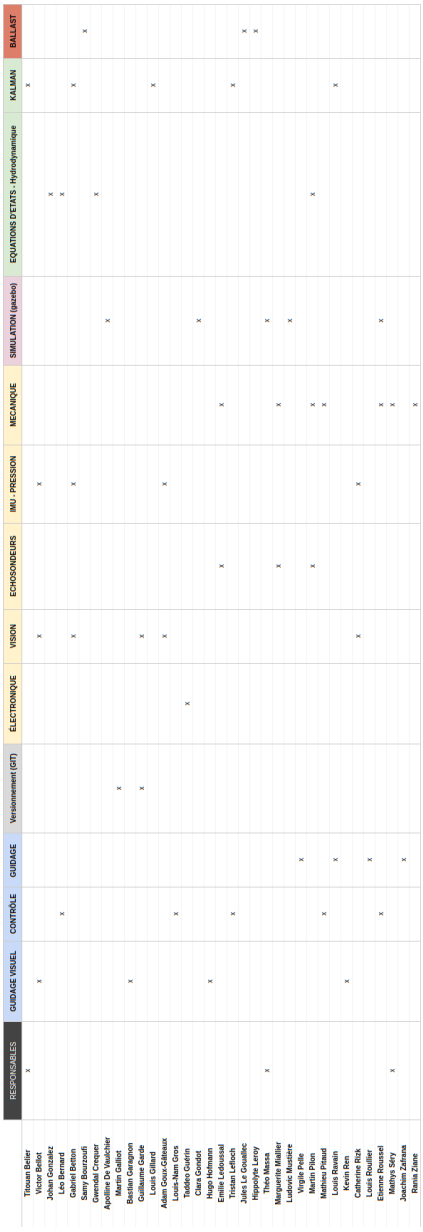
15.3 Conclusion

Each member's contribution forms a block that seamlessly integrates with the work of other team members, resulting in a cohesive whole. The mechanical architecture enables us to incorporate all initially chosen components for karst exploration and reconstruction.

Our work is entirely reusable for future projects. The modular nature of our approach allows for easy integration with other components, and the simulation confirms the feasibility of our technical solution for exploring underwater karst caves.

16 Annexe

Organisation des étudiants dans les différents groupes



	RESPONSABLES	GUIDAGE VISUEL	CONTRÔLE	GUIDAGE	Versionnement (GIT)	ÉLECTRONIQUE	VISION	ECHOSONDEURS	IMU - PRESSION	MECANIQUE	SIMULATION (gazebo)	EQUATIONS D'ETATS - Hydrodynamique	KALMAN	BALLAST
Titouan Belier	x													
Victor Bellot		x					x		x					
Johan Gonzalez												x		
Lilian Herard			x									x		
Gabriel Batton							x		x				x	
Samy Bouzzoufi														x
Gwendal Crequer												x		
Apolline De Vaulchier											x			
Marc Kaufer					x									
Bastian Campion		x												
Guillaume Garde					x		x							
Louis Gillard													x	
Adam Goux-Gateaux							x		x					
Louis-Nam Gros			x											
Théo Guichon						x								
Clara Gendot											x			
Hugo Hehmann		x												
Emilie Ledoussal								x		x			x	
Tristan Lelloch			x											x
Jules-Louis Lemaire														x
Hippolyte Leroy											x			
Théo Massa	x									x				
Marguerite Mathier								x			x			
Ludovic Mustière				x						x				
Virgile Pilia								x		x		x		
Matteo Pilia				x						x				
Mathieu Pitaud			x											
Louis Renoin				x									x	
Kevin Ren		x												
Catherine Rizk							x		x					
Louis Rocher				x										
Etienne Rousset			x							x	x			
Mathys Séry	x									x				
Joachim Zafrana				x						x				
Rania Ziane										x				

Figure 74: Tableau à double entrée de la répartition des élèves

RESPONSABLES	GUIDAGE VISUEL	CONTRÔLE	GUIDAGE	Versionnement (GIT)
Titouan Belier	Hugo Hofmann	Louis-Nam Gros	Louis Roullier	Martin Galliot
Théo Massa	Kevin Ren	Mathieu Pitaud	Joachim Zafrana	Guillaume Garde
Mathys Séry	Victor Bellot	Tristan Lefloch	Virgile Pelle	
	Bastian Garagnon	Etienne Roussel	Louis Ravain	

ÉLECTRONIQUE	VISION	ECHOSONDEURS	IMU - PRESSION	MECANIQUE
Taddeo Guérin	Catherine Rizk	Martin Pilon	Gabriel Betton	Mathys Séry
	Guillaume Garde	Emilie Ledoussal	Catherine Rizk	Rania Ziane
	Adam Goux-Gâteaux	Marguerite Miallier	Victor Bellot	Marguerite Miallier
	Gabriel Betton			Emilie Ledoussal
	Victor Bellot			Etienne Roussel
				Martin Pilon
				Mathieu Pitaud

SIMULATION (gazebo)	EQUATIONS D'ETATS - Hydrodynamique	KALMAN	BALLAST
Théo Massa	Gwendal Crequer	Tristan Lefloch	Hippolyte Leroy
Clara Gondot	Johan Gonzalez	Gabriel Betton	Samy Bourzoufi
Apolline De Vaulchier	Martin Pilon	Louis Ravain	Jules Le Gouallec
Ludovic Mustière	Léo Bernard	Louis Gillard	
Etienne Roussel		Titouan BELIER	

Figure 75: Organisation des étudiants dans les différents groupes