Control of Underwater Robots
Practical work
Implementation of a Proportional controller on the Blue ROV

Under the supervision of Vincent Creuze (LIRMM, University of Montpellier) and Vincent Hugel (COSMER, University of Toulon).

The PID controller is one of the most common controllers used on underwater vehicles, mainly because it is easy to implement and quite easy to tune. There are however some rules to follows to obtain the best performances. This practical work aims at discovering these rules, step by step.

What do you need to prepare, and how will you be graded?

There will be two practical sessions, which will take place on March 26th and 27th, 2024.

The topic you are currently reading is common to both practical sessions, so you have 5 hours (3 hours + 2 hours) to complete it. The topic should be meticulously prepared to allocate sufficient time for the comprehensive execution of practical work. Questions requiring preparation at home are delineated by green boxes.

To ensure timely completion, it is strongly advised to also prepare the outline of the report in advance and designate one student from each group for writing during the practical work.

Each group should arrive with a minimum of two computers, configured and tested in accordance with Professor Hugel's earlier instructions, to be able to control the Blue ROV.

You will be in groups of 4 or 5 students and each group is required to submit one collective report in PDF format via email at the conclusion of the second practical session (only one report per group of 4-5 students). Please ensure that the PDF file is named with the family names of all group members. Submissions should be made no later than 5 minutes after the scheduled end time of the session.

The reports should be sent to: vincent.hugel@univ-tln.fr and vincent.creuze@umontpellier.fr .

Reports submitted beyond the agreed-upon time will result in a reduction of the grade.

Your involvement will also be individually assessed during the two practical sessions.

Thank you for your cooperation, and we hope you will enjoy the experiments.

Step 1: Computing a thruster's control input depending on the desired thrust

The Blue ROV 2 is actuated by Blue Robotics T200 thrusters.

https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/

Each thruster is controlled by an Electronic Speed Controller (ESC).

https://bluerobotics.com/store/thrusters/speed-controllers/besc30-r3/

The ESC is controlled by applying an RC signal, similar to the one used to control most of existing servomotors. Although this signal is not exactly a PWM signal, many people call it a PWM.

To be initialized, the ESC waits for an initial « stopped signal » (i.e., a 1500 microseconds PWM signal), maintained during a few seconds. Then the ESC is controlled by a PWM signal from 1500 to 1900 µs for forward thrust and from 1100 to 1500 µs for reverse thrust. From 1470 to 1530 µs, there is a "dead zone", where the thruster remains stopped.

To implement any ROV's controller, one first needs to find the mathematical relation between the desired thrust (in kgf or daN) and the corresponding RC signal (PWM) that has to be sent to the ESC.

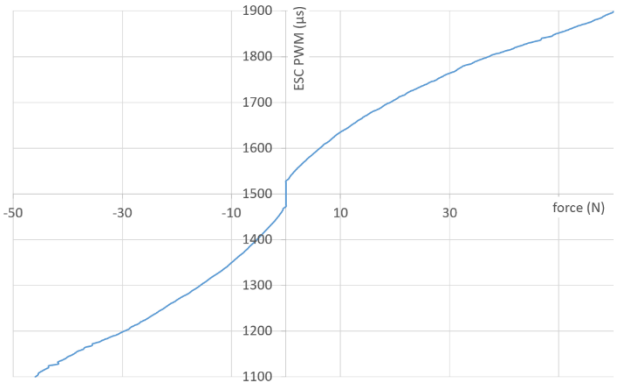
For this purpose, download the performances charts published by the thrusters' manufacturer:

https://cad.bluerobotics.com/T200-Public-Performance-Data-10-20V-September-2019.xlsx

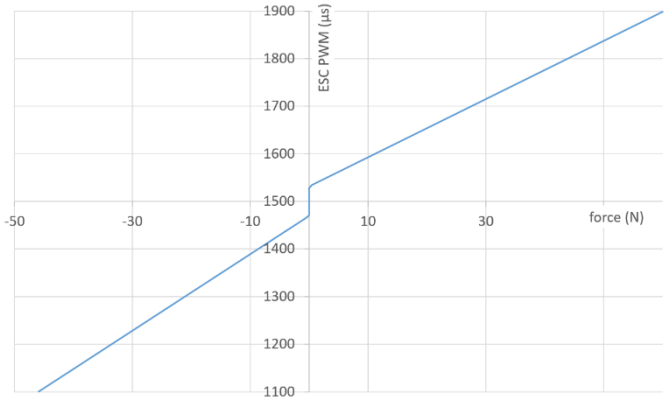
Now, plot the PWM curve (in µs, vertically) as a function of the desired thrust (in Newton, horizontally) assuming that the voltage of the ROV's battery is about 12 V. To simplify, consider that 1kgf = 10N.

You should find something similar to this curve.

Remark: this curve corresponds to an 18 V battery voltage, but you need to draw it for a 12 V battery.



This curve could be interpolated by polynomials, but this would later induce more computation for the microcontroller when it will execute the PID control. To limit the computation, we will simply interpolate this curve by two straight lines, as depicted on the next figure.



In this example (**18 V**), the interpolation is:

For negative thrusts: $PWM(f) = 1471 + 8.1 * f$

For positive thrusts: $PWM(f) = 1529 + 6.1 * f$

where PWM is the duration (in µs) of the PWM signal sent to the ESC and f is the force (in kgf or daN) produced by the T200 thruster.

Now, find a similar expression for the linear interpolations of your curve (corresponding to a 12 V battery voltage). Please, detail the calculations on a separate document.

For negative thrusts: $\boldsymbol{PWM(f) =} \quad + \quad * \boldsymbol{f}$

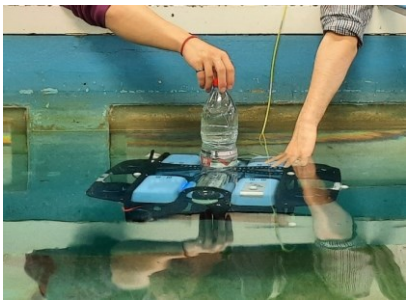
For positive thrusts: $\boldsymbol{PWM(f) =} \quad + \quad * \boldsymbol{f}$

On the same figure, draw the original curve and the interpolated one.

Step 2: Experimental validation of the thruster's model

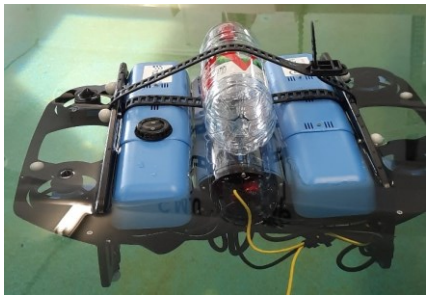
Before applying any control input to the robot, it is important to measure its floatability, i.e., the force that the thrusters should produce to make the ROV neutrally buoyant in the water (i.e., not sinking, nor floating).

Tip: You can use one of the two methods described in the tutorials. The teachers can give you a 1.5L bottle of water.



- Use the bottle to estimate the ROV's floatability (see the figure on the right). If the floatability is greater than 1.5 kgf, ask for additional weights to the teachers.

Once you have measured the floatability, it is time to check your thrusters' model. There are 2 or 4 vertical thrusters on the Blue ROV 2 (4 thrusters is the so-called "heavy configuration"). To produce a vertical force f_z each thruster should produce $\frac{f_z}{2}$ or $\frac{f_z}{4}$, depending on your Blue ROV's configuration. To check you thrusters' model:



- Attach a 1.5-Liter empty plastic bottle on the top of the ROV.
- Compute the PWM value to be sent to each of the 4 vertical thrusters to produce an overall force equivalent to 1.5kg + floatability. Note that, the overall produced vertical force is the sum of the 4 thrusters' forces.

- Apply this PWM value to the four vertical thrusters of the ROV in the `PressureCallback` function. This function applies the indicated PWM to each of the 4 thrusters.

If the model is correct, the bottle should almost submerge. It will however not completely submerge, because the linear interpolation done before underestimates the required PWM values (ask the teacher for explanations if you do not understand why). We do not require the interpolation to be perfect, as it will be included in the closed loop of the controller, which will compensate for most imperfections.

Step 3: Implementing a Proportional controller for the depth

In what follows, we consider only the depth control (z).

The force and torque vector created by the controller $\boldsymbol{\tau} = \begin{bmatrix} f_x \\ f_y \\ f_z \\ \Gamma_x \\ \Gamma_y \\ \Gamma_z \end{bmatrix}$ is reduced to $\tau = f_z$,

where f_z is the force that the vertical thrusters produce together.

A proportional controller means that you create a f_z force proportional to ε, the depth error:

$$f_z = K_p \times \varepsilon = K_p \times (z_{des} - z)$$

where z_{des} is the desired depth, z is the ROV's measured depth, and K_p is the proportional gain, which needs to be tuned.

- Implement this controller in your code.

- Test your controller for a step input (meaning that the desired depth is a constant value, $z_{des} = 0.5\ m$) and try to tune the K_p gain (no more than 10 trials, as it is normal that you have a large residual error at steady state).

- Plot and comment the curves (depth, thrusters' values).

<u>Tip:</u> To have a first estimate of the value of the K_p gain, try to figure out how strong you should push to maintain the ROV at a constant depth. For instance, would you push/pull with a force of 10 daN if the position error is 5 cm? Certainly not! Such considerations should help you to bound the values to be tested for the K_p parameter.

To tune K_p, start with a small value and increase it gradually. If your robot oscillates, this means that K_p is too big. Do not try to reach the perfection with a P controller, it is impossible. We will study better solutions later.

Step 4: Implementing a depth controller with floatability compensation

During the previous tests, you have seen that large steady state errors happen with a proportional depth controller. This is due to the floatability of the ROV. A common and easy way to deal with this, is to add a constant floatability compensation term to the controller.

You measured the floatability at Step 3. Add it to the controller like this:

$$\tau_z = K_p(z_{des} - z) + floatability$$

- Implement this controller in your code.
- Test your controller for a step input (meaning that the desired depth is a constant value, $z_{des} = 0.5\ m$) and try to tune the K_p again (it should be smaller than in the previous test).
- Plot and comment the curves (depth, thrusters' values).

Remark: Our ROV is buoyant, but this would also work if the ROV would naturally sink. Then the sign of g would be negative.

Step 5: Creating a trajectory compatible with the ROV's dynamics

We just saw that using a step input as the desired trajectory induces overshoots or oscillations. This is because the step input is not physically feasible for the robot. Step inputs make the tuning of any controller harder. To create a trajectory compatible with an underwater vehicle's dynamics, you should, for instance, replace the step input by a simple cubic polynomial. The algorithm is given below.

Cubic trajectory generation algorithm

Let us call z_{init} , the initial depth value of the desired trajectory (at $t = 0\ second$). Usually, z_{init} is the depth measured by the depth sensor when the ROV is floating at the surface. Let us call z_{final} , the final depth of the desired trajectory. Let us call t_{final} the time needed to reach the desired final depth.

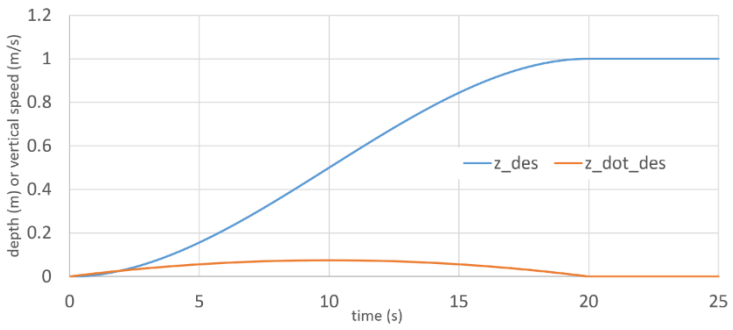
Then, a trajectory can be defined by the following cubic polynomial:

if $t < t_{final}$ then $z_{desired} = z_{init} + a_2 . t^2 + a_3 . t^3$

if $t \geq t_{final}$ then $z_{desired} = z_{final}$

with $a_2 = \frac{3(z_{final} - z_{init})}{t_{final}^2}$

with $a_3 = \frac{-2(z_{final} - z_{init})}{t_{final}^3}$



Similarly, the desired trajectory for $\dot{z}_{desired}$, the derivative of the depth, can be generated as:

if $t < t_{final}$ then $\dot{z}_{desired} = z_{init} + 2\ a_2 . t + 3\ a_3 . t^2$

if $t \geq t_{final}$ then $\dot{z}_{desired} = 0$

Code this algorithm at home and plot the obtained desired trajectories $z_{desired}(t)$ and $\dot{z}_{desired}(t)$, with $t_{final} = 20\ seconds$, z_{init} = 0 and z_{final}= 0.5 meter. Include the plots in your report.

Step 6: Implementing the polynomial trajectory

- Using the polynomial trajectory that you just designed, implement the P controller with gravity/buoyancy compensation (i.e., floatability compensation) designed earlier.
- Test your and tune the K_p again.
- Plot and comment the curves (depth, thrusters' values).

Tip:

To avoid sudden start of the thrusters due to the fact that your ROV does not actually start from $z = 0$ (because of the offset of the depth sensor), let the trajectory start at the actual depth of your ROV when it is floating at the surface.

Step 7: Proportional Integral controller (PI) for the depth

During the previous tests, you have seen that a steady state error always remains. To avoid this, one adds an integral term to the controller. This term integrates the error and thus allows to eliminate it completely:

$$\tau_z = K_p\tilde{z} + K_i \int_0^t \tilde{z}(t)dt + floatability$$

Where $\tilde{z} = (z_{des} - z)$ is the error along depth z, and K_p and K_i are respectively the proportional and integral gains of the PI controller.

Tip:

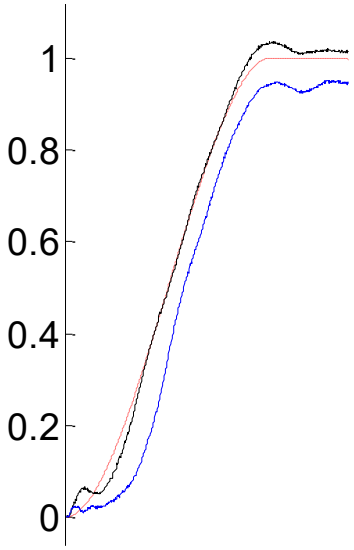
*To implement the integral term, create a "sum" variable that you reset at the starting time of the trajectory. Then, at each iteration, you do: "sum = sum + error*sampling_period;".*

To tune this controller, start by setting $K_i = 0$, and tune K_p first (tune it very "soft", i.e., lower than for the previous tests). Once K_p is tuned, start increasing K_i very slowly to make the steady state error converge to 0.

- Implement this controller in your code.
- Test your controller to track the trajectory and try to tune the K_p and K_i gains.
- Plot and comment the curves (depth, thrusters' values).

Step 8: Is the PI robust toward external disturbances?

- Without changing the gains found during the previous tests, attach an empty 1.5 Liter plastic bottle on the top of the ROV and plot the new depth trajectory and thrusters' values.



You should obtain something like the figure on the right, where the dotted red curve is the desired depth, the black curve corresponds to the nominal test and the blue curve is obtained when the bottle is attached to the ROV.

In fact, the PI or even the PID are not very robust controllers, and many other controllers exist to deal with the disturbances. Here are three examples:

Sliding Mode controller: https://www.youtube.com/watch?v=lcN1isXDhx4

L1 adaptive controller: https://www.youtube.com/watch?v=JmukulMxbxg

Saturation based nonlinear control: https://www.youtube.com/watch?v=7IIh61CRlIE

Step 9: Estimating the heave from the depth measurements with an alpha-beta filter

On your ROV, there is no sensor able to measure the vertical speed of the ROV, also called the heave and denoted w. To compute the heave, the alpha-beta filtering is an easy and computationally efficient way.

If you do not know already the alpha-beta filter, it is clearly explained here:

https://en.wikipedia.org/wiki/Alpha_beta_filter .

- **Read the page and implement it.** For the blue ROV depth sensor, $\alpha = 0.1$ and $\beta = 0.005$ should be adequate values, but you can adjust them if needed. The sampling frequency is about 58 Hz.

- Test it on the ROV.

Step 10: Implementing a PID with floatability compensation

- Implement the whole PID controller below, where $\dot{\tilde{z}} = \dot{z}_{desired} - \dot{z}$, with $\dot{z}_{desired}$ the desired speed trajectory designed at step 5 of this topic, and $\dot{z}$ the heave speed of the robot (i.e. w estimated with the alpha-beta filter above).

$$\tau_{PID} = K_p\tilde{z} + K_i\int_0^t \tilde{z}(t)dt + K_d\dot{\tilde{z}} + floatability$$

- Compare the behaviors of the PI and the PID controllers when you disturb the ROV by suddenly pushing on it.

OPTIONAL QUESTION: Implementing a Proportional controller (P) and a PI to control the yaw

In what follows, we first consider only the yaw (ψ , heading).

As we only consider ψ, the force and torque vector created by the controller $\boldsymbol{\tau} = \begin{bmatrix} f_x \\ f_y \\ f_z \\ \Gamma_x \\ \Gamma_y \\ \Gamma_z \end{bmatrix}$ becomes $\tau = \Gamma_z$,

where Γ_z is the torque created by the 4 horizontal thrusters.

Similar to what we did earlier for the depth, a proportional controller in yaw implies creating a torque Γ_z that is proportional to ε, the yaw error:

$$\Gamma_z = K_p \times \varepsilon = K_p \times (\psi_{des} - \psi)$$

where ψ_{des} is the desired yaw, ψ is the ROV's measured yaw, and K_p is the proportional gain, which needs to be tuned.

Implement this controller in your code. How can you deal with the $[0; 2\pi]$ interval of definition of ψ? Consider what can happen to $(\psi_{des} - \psi)$ around $\psi = 0$, and take appropriate precautions in your code.

- Test your controller for a step input (meaning that the desired yaw ψ_{des} is a constant value) and tune the K_p gain.
- Plot and comment the curves (yaw and thrusters' values).
- Try to disturb the ROV.

- Implement the trajectory generator and a PI controller to control the yaw.
- Test your controller to track the trajectory and tune K_p and K_i gains.
- Plot and comment the curves (depth, thrusters' values).

<u>Tip:</u> To avoid the useless integration of large initial errors, let the trajectory start at ψ_{init}= the actual yaw angle of your ROV just before the beginning of the trajectory and set the final value to $\psi_{final} = \psi_{init} + \pi/2$.