Control of Underwater Robots
Practical work #1
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.

Questions needing to be prepared at home are surrounded by green boxes.

Step 1: computing the PWM vs thrust relation of the thrusters

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.

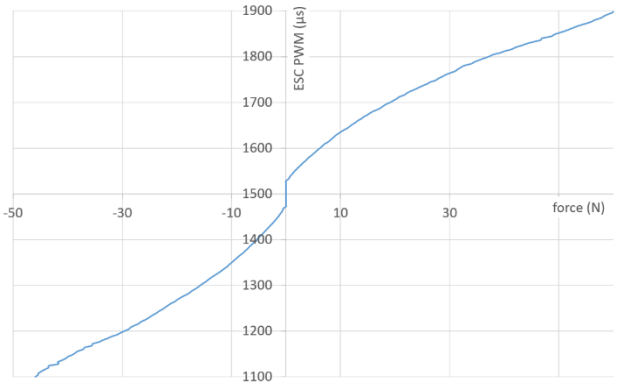
For this purpose, download the performances charts that are published here by the thrusters’ manufacturer:

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

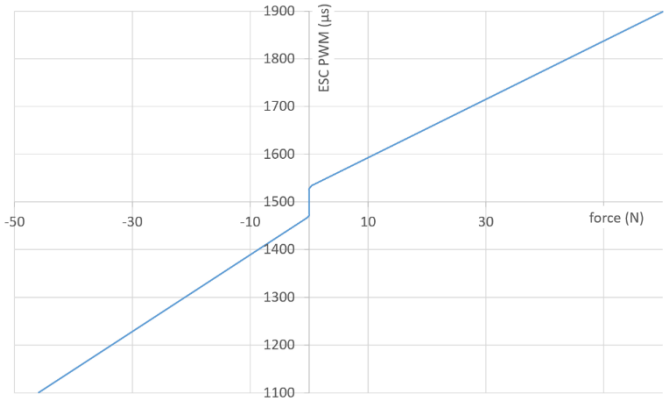
Now, plot the PWM curve (in µs, vertically) as a function of the desired thrust (in Newton, horizontally) assuming that the voltage of the ROV’s battery is about 16 V. To simplify, consider that 1kgf = 10N.

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 16 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 figure below.



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 16 V battery voltage). Please, detail the calculation of the coefficient on a separate document.

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

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

Step 2: Experimental validation of the PWM vs thrust function

There are 4 vertical thrusters on the standard Blue ROV 2. To produce a vertical force f_z each thruster should produce $f_z/_4$.

Let us now check the validity of the model that you found in Step 1:

- Attach a 1.5-Liter empty plastic bottle on the top of the ROV.
- Compute the PWM value to be sent to the ESC of the vertical thrusters to produce a force equivalent to 1.5kg.
- Apply this PWM value to the four vertical thrusters of the ROV.

If the model is correct, the bottle should almost submerge. It will however not completely submerge, because the ROV is naturally lightly buoyant, and a part of the thrust will be used to counteract the ROV's floatability and not only the bottle's one. Moreover, the linear interpolation done before underestimates the required PWM values.

Step 3: Estimating the floatability

Remove the bottle, and experimentally find the force needed to counteract the ROV's floatability, i.e., the additional 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 during the tutorials. The teachers can give you a small bottle of water.

Step 4: Implementing a Proportional controller (P) for the yaw

In what follows, we will first consider only one degree of freedom, namely 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 applied by the 4 horizontal thrusters.

A proportional controller means that you create a Γ_z torque 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, that you have to tune.

- Implement this controller in your code. How can you deal with the $[0; 2\pi]$ interval of definition of ψ? Think at 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.

Tip: To tune K_p, start with a small value and increase it gradually. If your robot oscillates a lot, this means that K_p is too big. Do not try to reach the perfection with a P controller…

Step 5: Implementing a Proportional controller for the depth

Now that you are familiar with the P controller, let us stop servoing ψ and try to servo another degree of freedom, namely the depth z.

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

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

- Write the equation of a Proportional controller for the depth control of the Blue ROV.
- Implement this controller in your code.
- Test your controller for a step input (meaning that the desired depth is a constant value, $z_{des} = 0.8\,m$) and try to tune the K_p gain (no more than 5 trials, as it is normal that you have a large residual error at steady state). *Warning: this K_p is different from the previous one and should be declared as a different variable in your code.*
- Plot and comment the curves (depth, thrusters' values).

Tip: 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.

Step 6: Implementing a depth controller with floatability compensation

During the previous tests, you have seen that a large steady state error remains. 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 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, $z_{des} = 0.8\,m$) 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 work also if the ROV would naturally sink. Then the sign of g would be negative.

Control of Underwater Robots
Practical work #2
Improving the control

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

In the previous practical work, you implemented a Proportional controller (P) to servo the yaw and the depth of the Blue ROV. Today, we will improve the behavior of the ROV by introducing smooth trajectories and by improving the controller.

Step 1: Creating a trajectory compatible with the ROV's dynamics

When the desired trajectory is a step input, this induces overshoots or oscillations in the trajectory tracking of the robot, because the step input is not physically feasible by the robot. Such trajectories make the tuning of any controller harder. To create a trajectory compatible with an underwater vehicle's dynamics, you should, for instance, replace the standard 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.

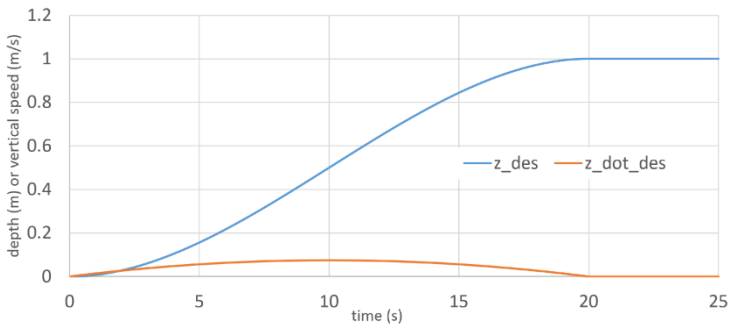
Then, a trajectory can be defined by the following cubic polynomial:

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 <u>at home</u> 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}= 0.8 meter.

Step 2: Implementing the trajectory

- Using the polynomial trajectory that you just designed, implement the P controller with gravity/buoyancy compensation (i.e., floatability compensation) designed during practical work #1.
- Test your and tune the K_p again.
- Plot and comment the curves (depth, thrusters' values).

Tip:

To avoid the useless integration of large initial errors 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 3: 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.

- Implement this controller in your code.
- Test your controller to track the trajectory generated at Step1 and try to tune the K_p and K_i gains.
- Plot and comment the curves (depth, thrusters' values).

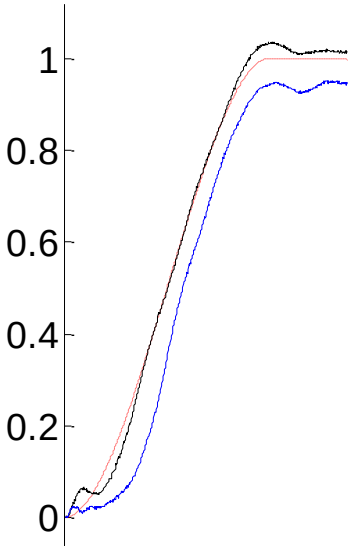
Tip:

*To implement the integral term, just create a "sum" variable that you reset at the starting time of the trajectory, and 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.

Step 4: Is the PI robust toward external disturbances?

- Without changing the gain obtained during the previous tests, attach an empty 1.5 Liter plastic bottle on the top of the ROV and plot the new depth 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 5: Proportional Integral controller (PI) for the yaw

- Implement the trajectory generator and the PI controller in your code to control the yaw.
- 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).

Tip:

To avoid the useless integration of large initial errors, let the trajectory start at the actual yaw angle of your ROV just before the beginning of the test.

OPTIONAL Step 6: Estimating the heave from the depth measurements with an alpha-beta filter

On our 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 our sensor, $\alpha = 0.45$ and $\beta = 0.1$ should be good values, but you can adjust them.

VERY OPTIONAL Step 7: Implementing a PID with floatability compensation

Implement now the whole PID controller below.

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

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