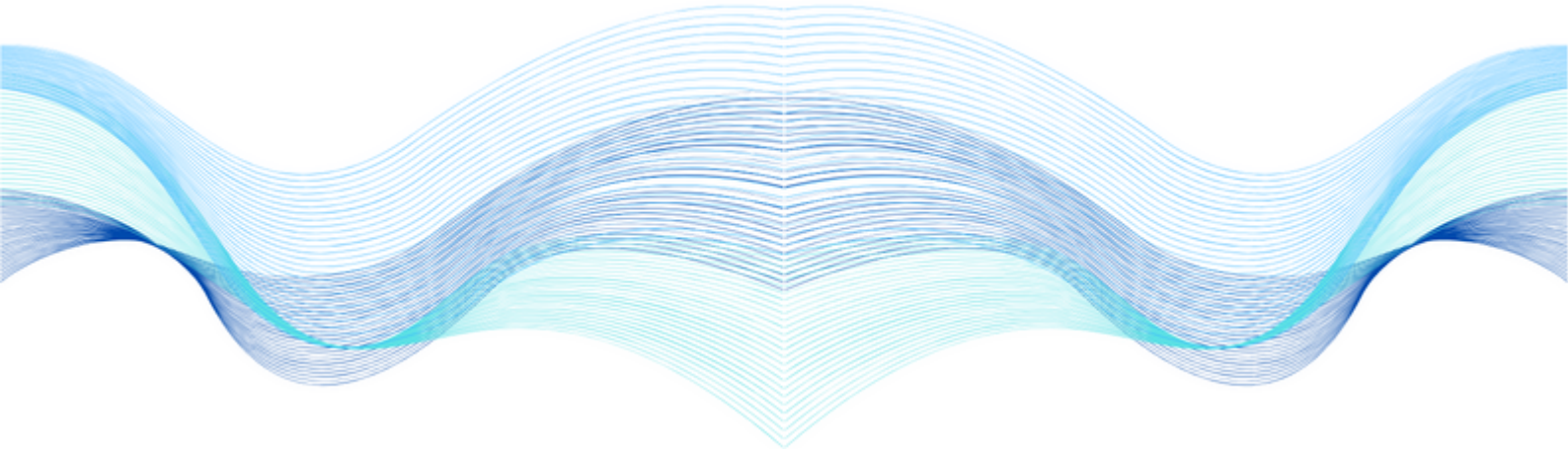
Parcours SYSMER

TP avec le BlueROV2

Julia CAUDRON, Mathys D'ARMAGNAC, Julie MENARD

Professeur encadrant :
Charly PERAUD

24 janvier 2025



1. Présentation du robot sous-marin BlueROV2

Le BlueROV2 est un robot sous-marin télécommandé conçu par la société Blue Robotics, utilisé dans divers domaines tels que l'exploration sous-marine, l'inspection de structures subaquatiques, la recherche scientifique et la surveillance environnementale.

L'une des caractéristiques du BlueROV2 est son système de propulsion à huit moteurs. Ces moteurs sont disposés de manière symétrique autour du corps du robot :

- Quatre sont orientés verticalement aux quatre coins du robot, permettant de contrôler les mouvements de translation verticale, ainsi que les mouvements de roulis et de tangage.
- Quatre sont orientés horizontalement en configuration X symétrique, permettant tous les mouvements de translation dans le plan horizontal du robot, ainsi que la rotation autour de l'axe vertical (angle de lacet).

Le BlueROV2 est équipé d'une gamme de capteurs et de dispositifs de contrôle qui en font un outil polyvalent et hautement performant pour les missions sous-marines. Chaque robot est équipé des éléments suivants :

- Afin de visualiser en temps réel l'environnement sous-marin, le BlueROV2 est équipé d'une **caméra frontale** de haute résolution, permettant une observation détaillée des zones explorées.
- Un **capteur de pression intégré** fournit des données précises sur la profondeur à laquelle le BlueROV2 opère.
- Positionné à l'avant du robot, le **sonar mono-faisceau** offre une capacité de détection avancée, permettant de repérer les obstacles et de cartographier les fonds marins avec une grande précision.
- Une **carte à micro-contrôleur** associée au logiciel *ArduSub* permet de faciliter la navigation et les manoeuvres du BlueROV2 et d'assurer son contrôle et sa stabilisation.

Chaque robot est alimenté par une batterie, placée dans un caisson cylindrique étanche et transparent, offrant une autonomie d'environ 3 heures pour une utilisation moyenne des propulseurs. Le BlueROV2 est relié à l'ordinateur via un câble ombilical de flottabilité légèrement positive, permettant la transmission des données. Ce système garantit une connexion stable et fiable entre le robot et l'ordinateur de contrôle, facilitant ainsi la surveillance et la commande à distance du BlueROV2 lors de ses missions sous-marines.

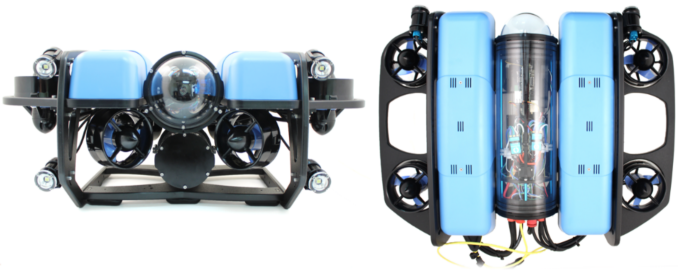


FIGURE 1 – Vue de face et de dessus du robot sous-marin BlueRov2

2. Test en bassin du BlueROV2

a) Prise en main et communication avec le BlueROV2

Dans un premier temps, nous avons manipulé le robot à l'aide d'une télécommande branchée à un ordinateur portable. Cela permet une première exploration du système avant de passer à l'asservissement. En connectant la télécommande à l'ordinateur, on peut contrôler les mouvements du ROV dans toutes les directions : translation avant-arrière, rotation sur les axes, et déplacements latéraux. Cependant, nous n'avons pas pu plonger. Cette phase est importante pour vérifier que tous les composants du ROV fonctionnent correctement et pour effectuer les ajustements nécessaires avant de passer à la mise en place d'un asservissement pour une conduite plus précise et autonome.

FIGURE 2 – BlueROV2 dans l'eau

Afin que le BlueROV2 se déplace dans l'eau à l'aide de la télécommande, nous avons lancé différentes commandes dans plusieurs terminaux :

— **ros2 launch autonomous_rov run_mavros.launch** pour la communication avec le ROV
— **ros2 launch autonomous_rov run_gamepad.launch** pour la communication avec la manette de jeu
— **ros2 launch autonomous_rov run_listener_MIR_joy.launch** pour notre code
— **ros2 launch ping_sonar_ROS run_pinger.launch** pour la communication avec le pinger

Enfin, si on modifiait notre code python, il était important de build le package avec la commande **colcon build** et de sourcer avec **source install/setup.bash** dans le dossier ∼/**ros2_ws**.

b) Code de commande du ROV

i - Compréhension du code

La partie commande du ROV est en partie automatisée, elle est gérée par un script Python appelé par la commande **ros2 launch autonomous_rov run_listener_MIR_joy.launch** qui tourne en continu dès le démarrage du ROV.

Le code étudié permet de commander le robot sous-marin en utilisant ROS 2. Ainsi, on peut récupérer les données des capteurs, envoyer des commandes aux moteurs mais aussi implémenter des asservissements sur certains paramètres comme la profondeur ou le cap à l'aide de PIDs. Différents types de fonctions sont utilisées, comme les **callbacks** pour récupérer les données des capteurs.

Voici ci-dessous le détail de certaines fonctions que nous avons étudié :

— La fonction **timer-callback** permet d'envoyer les commandes demandées aux moteurs selon les 6 degrés de liberté disponibles grâce à la fonction **setOverrideRCIN** ayant comme arguments : pitch, roll, throttle, yaw, forward, lateral.

— La fonction **joyCallback** permet de récupérer les entrées des boutons situés sur la manette, afin de sélectionner le mode de pilotage : manuel (bouton X), automatique (bouton Y) ou corrigé (bouton A).

— La fonction **mapValueScalSat** prend en entrée une valeur comprise entre -1 et 1, provenant soit de la manette, soit du mode automatique, et la convertit en une valeur de commande scalaire pour les moteurs, située entre 1100 et 1900 (sens inverse), avec une position neutre à 1500. Cela permet de régler la saturation des moteurs et d'éviter qu'ils ne tournent trop vite hors de l'eau.

— La fonction **OdoCallback** récupère les données des capteurs (comme la centrale inertielle IMU) et les utilise pour calculer les angles de roulis, tangage et lacet, ainsi que les vitesses angulaires associées.

Le code en entier repose sur la création d'une classe **MyPythonNode** représentant le nœud ROS de notre système. C'est au sein de cette classe que toutes ces fonctions sont définies, permettant le fonctionnement du système.

ii - Asservissement du cap

Suite à l'étude du code, nous avons voulu implémenter une boucle d'asservissement permettant de fixer le cap du ROV quand on appuie sur une touche de la manette.

Tout d'abord, on ajoute la partie permettant de récupérer le cap quand on appuie sur la touche A. La touche A correspond au mode correction du code. Ainsi, lorsqu'on utilise ce mode, on exécute la ligne suivante pour récupérer le cap actuel et le passer dans une nouvelle variable :

C'est cette variable qu'on vient rentrer dans notre boucle d'asservissement :

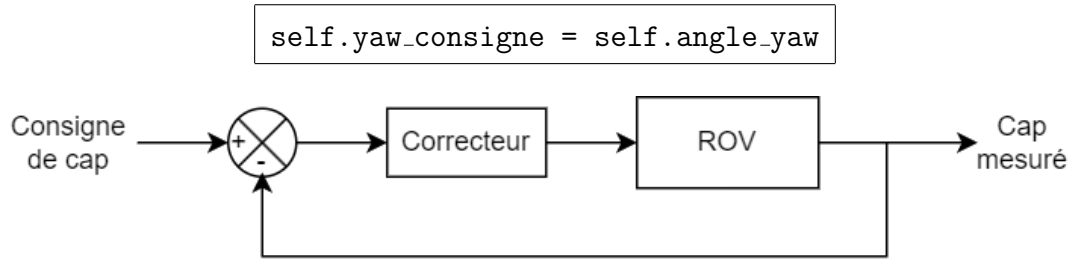


FIGURE 3 – Boucle d'asservissement du cap

Nous avons décidé d'implémenter un correcteur PID. En revanche, pour des raisons de temps, nous avons juste utilisé le correcteur proportionnel P. On a commencé par $kp = 0.1$ mais la réponse était très lente, tandis que pour $kp = 1$, le ROV est très réactif et précis. De plus, il ne change pas de son cap initial.

```
# Initialisation PID
self.kp = 1.0
self.ki = 0.0
self.kd = 0.0
self.error = 0
self.integral_error = 0
self.error_last = 0
self.derivative_error = 0
```

On vient ensuite entrer le code d'un PID dans la partie du mode correction.

```
# Correction PID
self.error = -(self.yaw_consigne - angle_yaw)
self.integral_error += self.error * 0.04 # Période d'échantillonnage IMU
self.derivative_error = (self.error - self.error_last) / 0.04
self.error_last = self.error
sortie_pid = self.kp * self.error + self.ki * self.integral_error + self.kd
* self.derivative_error
# Saturation de la commande
self.Correction_yaw = self.mapValueScalSat(sortie_pid)
# Affichage de la commande
self.get_logger().info("résultat = " + str(self.Correction_yaw))
```

Pour obtenir la période d'échantillonnage de la centrale inertielle du ROV (IMU), on utilise la commande : **ros2 topic hz/imu/data**. Cette commande vient faire la moyenne de la fréquence d'échantillonnage. Ici, on a une période de 0.04 secondes, ce qui nous semble assez rapide et concordant avec les données reçues.

Au final, quand on ajoute manuellement une perturbation sur le cap, le ROV vient se recaler tout seul sur le cap initial. On voit bien sur la commande que plus la perturbation est grande, plus la commande vient à être élevée pour compenser cette perturbation.

iii - Asservissement de la distance entre la piscine et le ROV

Lors de la deuxième séance de TP, nous avons eu le temps de réaliser l'asservissement en distance. Le but était de faire avancer le ROV dans une direction voulue (cap asservi) sans toucher une paroi de la piscine. La distance entre le mur et le ROV est notre consigne. Le sonar mono-faisceau (pinger) fonctionne moins bien en dessous de 70 cm, c'est pourquoi nous avons décidé de mettre une consigne de 80 cm.

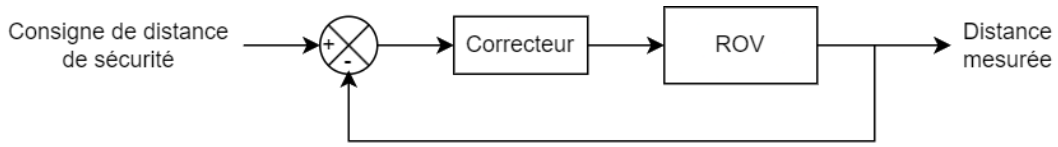


FIGURE 4 – Boucle d'asservissement de la distance

Le correcteur que nous avons implanté est le même que celui vu précédemment sauf que nous changeons la consigne. Cependant, avant d'asservir la distance entre la piscine et le ROV, on le fait avancer à une vitesse donnée de 1550 ce qui est assez lent ($v_{reel} = 0$ quand $v = 1500$).

Ainsi, quand le ROV s'approche à moins de 1 m de la paroi, l'asservissement s'active.

```
# Initialisation PID Pinger
self.kp_ping = 0.4
self.ki_ping = 0.0
self.kd_ping = 0.0
self.error_ping = 0
self.integral_error_ping = 0
self.error_last_ping = 0
self.derivative_error_ping = 0
```

```
# Partie Pinger
self.avance = 1550
self.get_logger().info("distance = " + str(self.pinger_distance))
# Condition d'asservissement
if (self.pinger_distance <= 1):
# Correction distance
self.error_ping = -(0.8 - self.pinger_distance)
self.integral_error_ping += self.error_ping * (1/8.3)
self.derivative_error_ping = (self.error_ping - self.error_last_ping) /
(1/8.3)
self.error_last_ping = self.error_ping
sortie_pid_ping = self.kp_ping * self.error_ping + self.ki_ping *
self.integral_error_ping + self.kd_ping * self.derivative_error_ping
# Saturation de la commande
self.avance = self.mapValueScalSat(sortie_pid_ping)
# Affichage de la commande
self.get_logger().info("résultat_ping = " + str(self.avance))
```

De la même manière que précédemment, nous utilisons la commande **ros2 topic list** pour trouver où se trouve la période d'échantillonnage du pinger.

On observe une oscillation permanente autour de la consigne. Cela pourrait se régler facilement avec un réglage plus poussé du PID.

Conclusion

Pour conclure, le BlueROV2 s'avère être un robot sous-marin très performant, adapté à une grande variété de missions. Grâce à ses moteurs, capteurs et systèmes de contrôle, il est capable d'effectuer des mouvements précis sous l'eau, comme le maintien du cap ou l'évitement d'obstacles. L'implémentation des boucles d'asservissement, comme pour le cap et la distance, a montré son efficacité pour guider le robot de manière autonome. Même si des ajustements restent à faire pour perfectionner la réactivité et la stabilité du système, ces tests ouvrent la voie à des applications sous-marines de plus en plus sophistiquées que le contrôle manuel par un opérateur.