SIREHNA NAVAL GROUP | AQUA.BOT

Challenge Aqua.Bot

Rapport Team 42 Lyon

42 | LYON Auvergne-Rhône-Alpes

Sommaire

1. Présentation de l'équipe
2. Organisation durant la compétition
    a. Découverte de l'environnement
    b. Organisation et répartition des tâches
    c. Notre évolution au fil du temps
3. Guide d'installation
4. Présentation des modules
    a. Module de navigation
    b. Module de pathfinding
    c. Module de perception
5. Difficultés rencontrées
6. Idées d'amélioration

Présentation de l'équipe

Hugo Ferraud
22 ans

Licence de mathématiques et informatique.

github.com/hferraud

Ethan Delage
18 ans

Baccalauréat spécialités mathématiques et physique.

github.com/EthanDelage

Tom Damerose
19 ans

Baccalauréat spécialités mathématiques et informatique.

github.com/tdameros

Notre équipe représente fièrement l'école 42 Lyon Auvergne Rhône-Alpes, où nous sommes tous les trois en deuxième année. L'école 42, fondée sur un concept novateur d'apprentissage par projets et par la pratique, nous a permis de développer nos compétences en programmation et de cultiver notre esprit d'équipe. Sans professeur ni cours, l'école 42 Lyon fait aujourd'hui parti du réseau 42 comportant plus de 18 000 étudiants autour de 52 campus répartis dans le monde.

Organisation durant la compétition

Dans le cadre de la compétition Aquabot 2023, notre équipe s'est engagée dans l'automatisation d'un drone de surface (USV) pour la sécurisation d'une zone maritime. Pour concrétiser cet objectif, nous avons initié notre démarche en explorant l'environnement de développement et de simulation mis à notre disposition par les organisateurs. Par la suite, nous nous sommes organisés de manière à répartir efficacement les taches nécessaires au développement de notre solution.

1. Découverte de l'environnement

Robot Operating System (ROS) est une plateforme open-source conçue pour simplifier le développement de logiciels de robotique. Notre première étape a été de nous plonger dans la documentation officielle de ROS afin de comprendre les concepts fondamentaux, la structure de communication basée sur les messages, les nœuds, les topics, et les services.

Notre exploration de ROS a débuté par la création de notre premier nœud, une entité logicielle qui communique avec d'autres nœuds via la passerelle ROS. Ce premier nœud avait pour objectif de nous permettre de piloter notre drone de surface (USV) en publiant des valeurs sur les topics associes à la vitesse et l'angle du moteur.

L'environnement de simulation Gazebo nous a permis d'intégrer notre nœud de pilotage du bateau et de prendre connaissances des effets de notre nœud sur l'USV. Ainsi nous avons pu visualiser concrètement les effets de notre nœud sur l'USV, offrant ainsi un premier aperçu de l'interaction potentielle de notre solution avec l'environnement simulé du challenge.

En résumé, cette phase initiale d'exploration de l'environnement ROS a posé les bases nécessaires pour le développement ultérieur de notre solution. Nous avons réussi à établir une connexion fonctionnelle entre notre nœud de pilotage et le système de simulation, jetant ainsi les fondements de notre participation à la compétition. La prochaine étape consistera à approfondir notre compréhension de ROS pour développer des fonctionnalités plus avancées et répondre de manière efficace à la mission de *"Patrol and Follow"*.

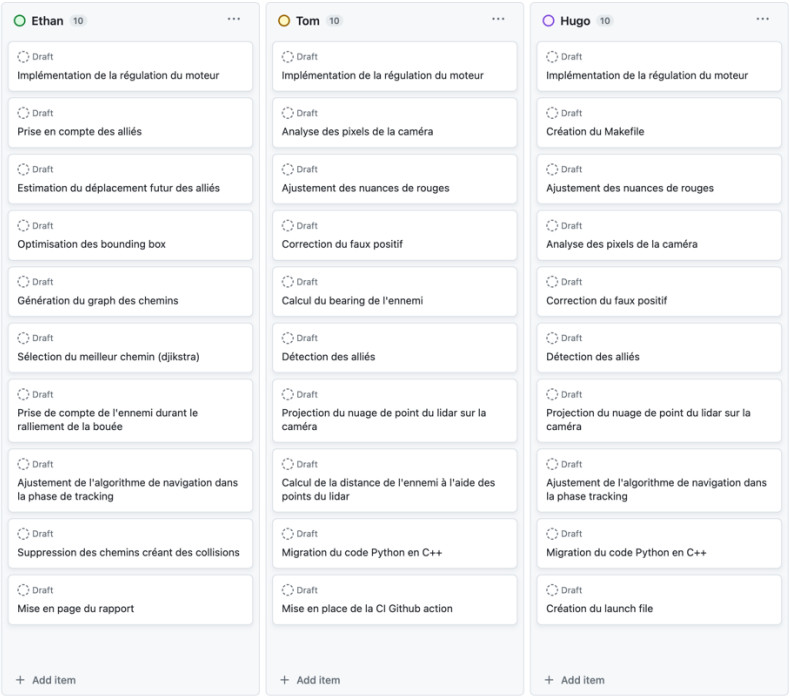
1. Organisation et Répartition des Tâches

Lors de la compétition, nous nous sommes organisés de manière à ce que chacun de nous puisse travailler de manière autonome, sans pour autant laisser totalement la responsabilité d'une partie à une seule personne. En effet, nous sommes tous les trois étudiants à l'école 42 et nous nous connaissons depuis plus d'un an. De ce fait, nous avons déjà travaillé ensemble sur des projets en collaboration et sommes donc habitués à ce type d'exercice.

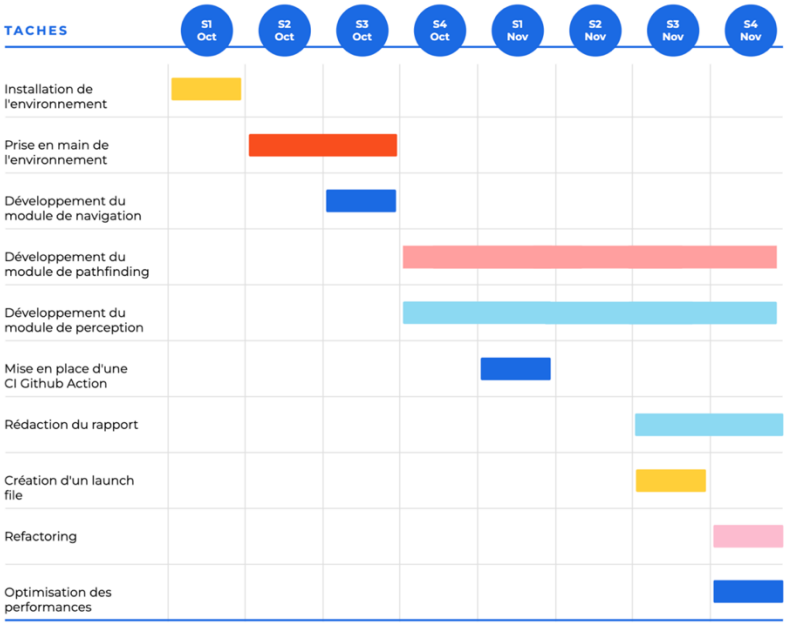
Notre approche a alors été de travailler d'abord en commun en effectuant quelques petits exercices et de nous familiariser avec l'environnement de simulation qui nous était totalement inconnu avant le début de la compétition. Nous avons ensuite posé les fondations du projet en développant le module de navigation ensemble, ce qui nous a par la suite permis de nous répartir les tâches afin de travailler plus efficacement.

A. Utilisation de *Git* et *Github*

Nous avons utilisé l'outil de gestion de version Git tout au long de la compétition. Git est un outil puissant et extrêmement utile lors de projets en équipe, permettant de travailler sur des "*branches*" puis de les mettre en commun sur la branche principale et de résoudre efficacement les potentiels conflits pouvant émerger. Sur notre dépôt Github une protection de la branche principale nécessitant une relecture du code produit par tous les membres de l'équipe a été mise en place afin de se tenir au courant des derniers changements. Enfin, une *"Continuous Integration Github Action"* a été déployée permettant de vérifier que chaque bout de code publié grâce à Git est totalement valide.



2. Notre évolution au fil du temps



Guide d'installation

Dans le cadre de ce projet, nos choix techniques ont été déterminés par la volonté de privilégier la simplicité et l'efficacité. Nous avons opté principalement pour l'utilisation du langage C++, tout en réservant l'usage de Python uniquement à des fins de prototypage. Cette décision repose sur des considérations de performances et de fiabilité.
Pour nous aider, nous utilisons deux librairies populaires en C++ : OpenCV, spécialisée dans le traitement d'image de la caméra, et Eigen, dédiée aux calculs matriciels.

Prérequis :

- OpenCV 4.8.1
- Eigen3
- C++ 17
- Ros Humble
- Gazebo Garden

1. Sourcer Ros

```
source /opt/ros/humble/setup.sh
```

2. Build le projet

```
make build
. install/setup.sh
```

3. Lancer une simulation

```
make launch
```

Différentes règles sont à disposition dans le Makefile pour lancer en mode headless, lancer les outils de debug (rqt, rviz) ou encore supprimer tous les fichiers de build.
Il est possible d'activer le rendu de la caméra en retirant les commentaires dans src/perception/Perception.cpp l.74.

Présentation des modules

1. Module de navigation

L'objectif de ce module est de piloter l'USV de manière efficace et optimisée dans l'environnement de simulation. Pour cela nous lisons sur un topic la distance ainsi que l'orientation de notre cible par rapport à notre drone afin de calculer l'angle et la vitesse nécessaire pour naviguer efficacement vers la cible.

A. Problème d'oscillation du moteur

Le premier prototype du modèle de navigation prenait en compte l'orientation de notre USV par rapport à la cible et orienter le moteur dans la direction de la cible. Le fait que l'orientation ne soit pas actualisée en temps réel, et l'inertie du bateau causant un effet d'oscillation autour du cap réel vers la cible.

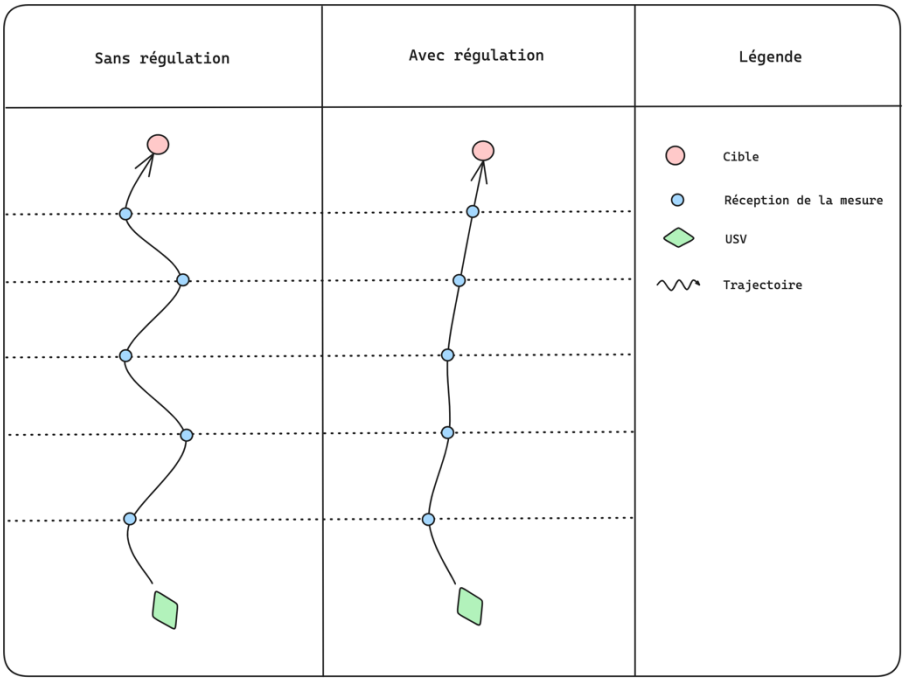


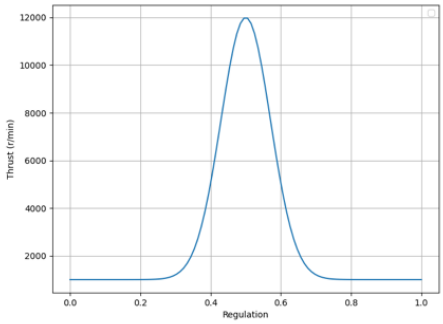
Figure 1. Schéma de l'impact du système de régulation sur l'oscillation

B. Système de régulation

Afin de pallier ce phénomène nous avons mis en place un système de régulation de l'orientation du moteur. Ce système vient faire varier la grandeur réglante (orientation du moteur) proportionnellement à / l'écart entre la mesure (orientation de l'USV) et la consigne (orientation de bateau par rapport à la cible). Le coefficient de proportionnalité est appelé gain du régulateur. Ce système permet de modifier efficacement et de manière graduelle l'orientation du moteur afin d'éviter le phénomène d'oscillations du moteur et donc se déplacer de manière plus optimale jusqu'à la cible.

C. Vitesse du moteur

Le problème de la vitesse du moteur s'est alors posé lorsque l'USV devait prendre un virage trop important, malgré le système de régulation, une vitesse trop importante provoquait une rotation complète du bateau avant même la réception du prochain message contenant la mesure. Nous avons donc cherché une fonction qui pourrait correspondre à notre vitesse par rapport à la grandeur réglante. Nous avons tout de suite pensé à une fonction gaussienne et en faisant varier quelques paramètres nous avons obtenu une fonction qui nous permet de nous déplacer efficacement dans l'environnement avec une vitesse maximale en ligne droite et une vitesse grandement réduite lorsque l'USV prend un virage serré.



2. Module de pathfinding

Le module de pathfinding a pour but de guider le module de navigation afin de ne pas entrer en collision avec l'environnement (rochers, alliés, ennemi). Ce module contient deux parties majeures : une partie permettant de cartographier le monde en temps réel, et la seconde visant à calculer le chemin optimal à emprunter.

A. Cartographie du monde

La cartographie se déroule à plusieurs échelles. Tout d'abord, au lancement de la simulation le module pathfinding récupère les données de chaque rocher (dans le fichier *'obstacles.txt'*) qui sont stockées sous forme de 4 points définissant une bounding box (boîte englobante). Cela constitue nos obstacles fixes sur la carte. Une fois les bounding box créer, nous générons un graphe contenant toutes les liaisons entre les bounding box. Ce graphe est un organe vital au bon fonctionnement du calcul de chemin optimal.

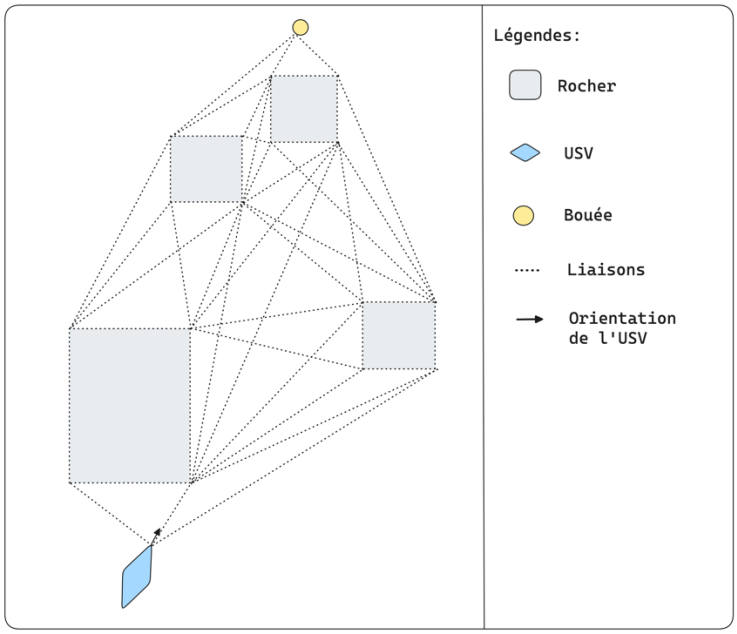


Figure 1. Graphe représentant les chemins possibles

Une fois la base de notre graphe établie, le module de pathfinding commence à prendre en compte les obstacles dynamiques (alliés, ennemi). Pour cela, *pathfinding* se connecte à deux topics ('/vrx/patrolandfollow/alert_position' et '/perception/pinger') afin de recevoir la position des alliés ainsi que celle de l'ennemi (calcule par le module de perception). À partir de la position et de l'orientation du bateau *pathfinding* crée une bounding box qui estime la position de l'allié dans les prochaines secondes. Ces bounding box sont alors incorporées dans le graphe pour préciser les zones interdites.

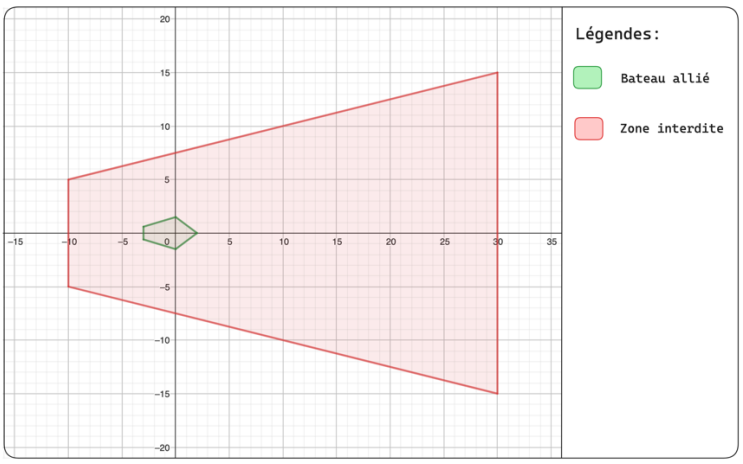


Figure 2. Bounding box de l'allié

B. Calcul du chemin optimal

Une fois le graphe généré, nous utilisons un algorithme de pathfinding (Djikstra) qui nous permet de calculer le chemin le plus court pour atteindre la bouée. Ne connaissant pas la position de cette dernière, nous devons la calculer. Pour cela nous allons nous aider du pinger bouée, qui nous permet d'obtenir l'angle et la distance de la bouée par rapport à notre bateau. Ces éléments nous permettent d'estimer la position de la bouée à plus ou moins 5 mètres. Cette imprécision est due au fait que nous ne prenons pas en compte toutes les informations de la simulation : notamment l'impact des vagues sur l'équilibre de notre bateau.
L'algorithme de Djikstra s'appuie sur les poids des arêtes (distance entre deux points sur la carte) pour calculer le plus court chemin. Cet algorithme est très performant et répond parfaitement à notre problème de calcul de chemin optimal.

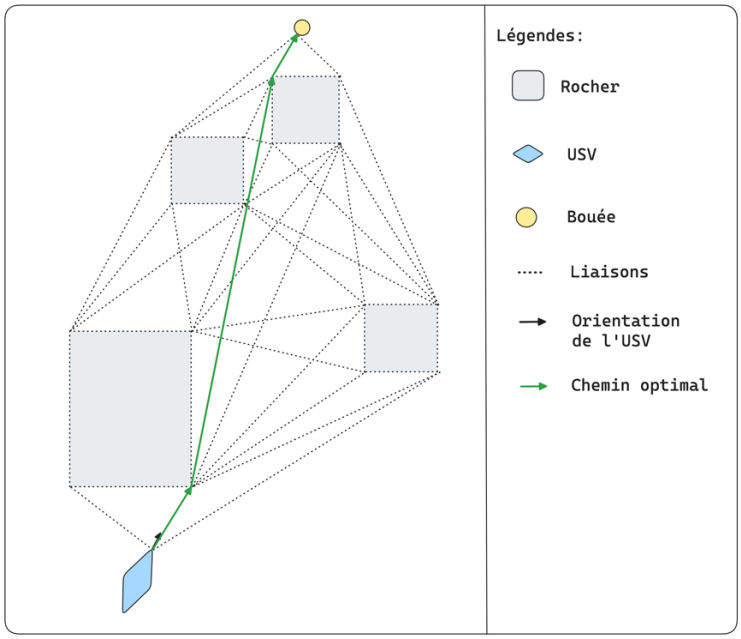


Figure 3. Chemin optimal généré par l'algorithme de Djikstra

Cet algorithme est aussi utilisé dans la phase de poursuite de l'ennemi pour pouvoir éviter les rochers ainsi que les alliés et donc éviter toutes collisions avec l'environnement extérieur. Le calcul de la position de l'ennemi est quant à lui fait dans le module de perception.

3. Module de perception

A. Analyse des images de la caméra

Notre première étape consiste à utiliser la caméra pour détecter la présence d'ennemis dans notre champ de vision. Nous utilisons une méthode de détection de pixels à l'aide d'OpenCV. Le bateau ennemi est identifié grâce à une nuance de rouges très précise que nous avons ajustée en utilisant la couleur du bateau (sur blender) ainsi que la lumière de l'environnement.
Une fois les pixels analysés, nous créons des rectangles englobant des regroupements de pixels proches représentant potentiellement l'ennemi.

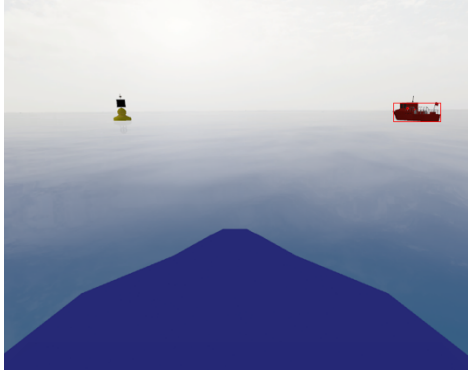


Figure 1. Détection de l'ennemi

Cependant, un problème vient se greffer à cela, il s'agit des faux positifs obtenus sur les alliées. En effet, les alliées possèdent une bouée rouge qui contient la même couleur que l'ennemi.

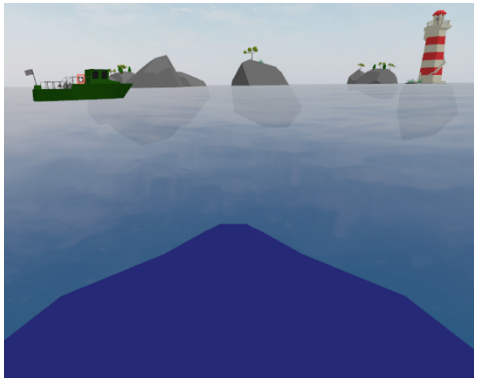


Figure 2. Faux positif sur la bouée de l'allie

Pour remédier à ce souci, nous faisons aussi une détection des alliées en utilisant le même processus vu ci-dessus mais cette fois-ci avec une couleur verte. Cela nous permet donc de corréler les 2 informations dans le cas où un rectangle rouge est détecté à l'intérieur un rectangle vert, nous pouvons en déduire qu'il s'agit bien de la bouée d'un bateau alliée et non pas d'un ennemi.

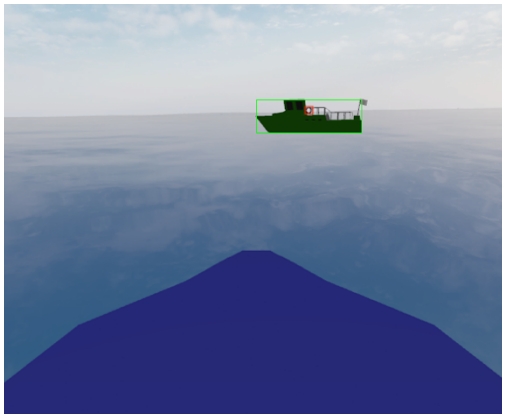


Figure 3. Correction du faux positif

Ensuite, notre stratégie consiste à sélectionner le plus grand rectangle rouge (sans les faux positifs) pour augmenter la probabilité de cibler précisément le bateau ennemi. Une fois identifié, nous procédons au calcul de l'angle de l'ennemi (bearing). Ce calcul combine la position du rectangle détecté avec le champ de vision de notre caméra (FOV), nous fournissant ainsi des informations précises sur l'orientation de l'ennemi par rapport à notre propre position.

B. Utilisations du lidar

Maintenant que nous avons détecté la présence de l'ennemi et que nous avons obtenu son angle, la prochaine étape consiste à utiliser le lidar. Le lidar est un capteur positionné sur l'USV qui envoie des rayons lumineux afin d'obtenir une distance par rapport à l'objet observé. Grâce au lidar, nous obtenons un nuage de points représentant l'environnement qui nous entoure.

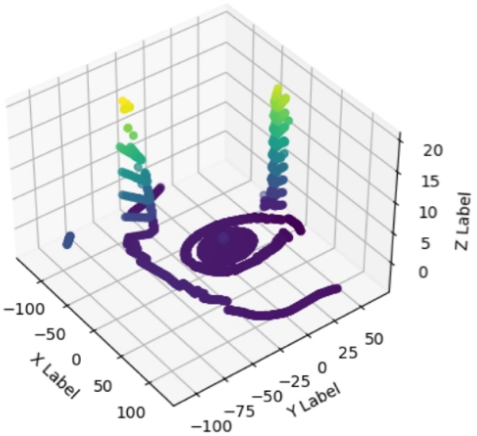


Figure 4. Visualisation du nuage de point sur MatPlotLib

Une fois le nuage de points du lidar obtenu, nous procédons à la projection de chaque point sur la caméra. Cette opération s'appuie sur les informations fournies par le topic de la caméra, telles que sa résolution en pixels et la matrice de projection. Grâce à ces données, nous sommes en mesure d'effectuer une projection des points du nuage correspondant sur l'analyse de la caméra.

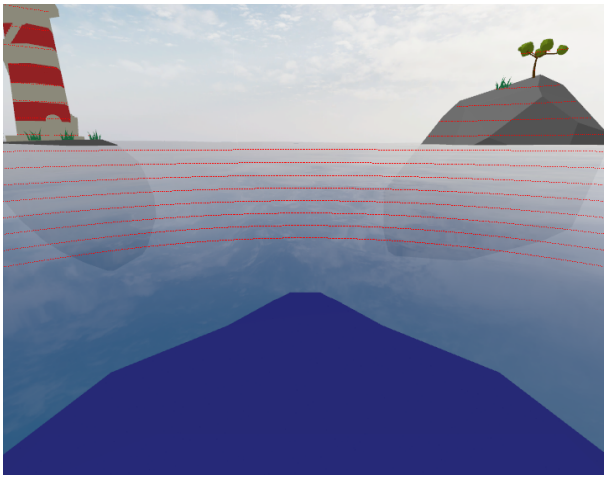


Figure 5. Projection du nuage de point sur la caméra

La projection des points du lidar sur la caméra nous permet ainsi de mettre en relation les coordonnés en deux dimensions obtenues avec les pixels de la caméra correspondants à ceux de l'ennemi. Cette opération nous permet alors d'évaluer la distance qui nous sépare du drone ennemi. En combinant la distance et l'angle par rapport à l'ennemi nous sommes alors en mesure de les communiquer via un topic au module de pathfinding qui établira ensuite un chemin optimisé afin de rejoindre l'ennemi.

Les difficultés rencontrées

Nous avons rencontré nos premières difficultés avec la cartographie des obstacles. Initialement, nous avons entrepris de créer un parseur de fichier .sdf pour générer automatiquement des boîtes englobantes. L'objectif était de pouvoir ajuster la position des obstacles sans refaire manuellement le calcul des boîtes englobantes. Malheureusement, notre algorithme produisait des boîtes qui manquaient d'optimisation. Face à ce constat, nous avons préféré effectuer cette tâche manuellement afin de ne pas perdre davantage de temps sur ce point.

Notre seconde difficulté a été la mise en place du module de perception. À l'origine, nous avions envisagé d'utiliser une IA pour la reconnaissance des bateaux ennemis, conscients que la détection de pixels était limitée en cas de changement d'environnement. Nous avons tenté d'entraîner un classifieur OpenCV (cascade classifier) avec 1000 images de l'ennemi sous différents angles, mais les résultats se sont révélés décevants. À la suite de cela, nous avons exploré l'utilisation d'un modèle d'IA pré-entraîné. Bien que les

résultats aient été plus prometteurs, nous avons rencontré un nombre trop élevé de faux positifs par rapport à la détection par pixel.

Un troisième aspect a été la gestion des performances au sein de notre code, en particulier la section de l'algorithme de perception dédiée à l'analyse des points du lidar. Au départ, cette partie du code était développée en Python. Cependant, nous avons rapidement constaté des limitations en termes de performances, ce qui impactait significativement la réactivité du module. Afin d'optimiser cette section cruciale, nous avons pris la décision stratégique de migrer du langage Python vers le langage C++, tout en implémentant des techniques de multithreading.
Le passage à C++ et l'utilisation de multithreading ont été des choix délibérés pour améliorer considérablement les performances de notre algorithme de perception. Cette transition nous a permis d'exploiter pleinement les ressources disponibles du système, augmentant ainsi la vitesse de traitement des données provenant du lidar, tout en garantissant une réactivité de l'algorithme de suivi.

Idées d'améliorations

Avec un délai supplémentaire, notre objectif aurait été de concevoir un algorithme de patrouille complet, établissant une connexion entre le module de pathfinding et celui de perception dès l'approche de la bouée.
Actuellement, notre méthode repose uniquement sur la proximité de l'ennemi à la bouée, ce qui engendre des problèmes de collision avec celle-ci.
Notre idée serait de nous inspirer du problème de la galerie d'art pour déterminer des positions optimales, permettant au champ de vision de la caméra de couvrir efficacement l'environnement, améliorant ainsi la détection des ennemis.

Par ailleurs, une autre amélioration souhaitée concernait la mise en place d'un système de sécurité pour éviter les collisions avec les obstacles en situation de dernier recours à l'aide du lidar. Cette fonctionnalité permettrait d'offrir au bateau une autonomie totale, lui permettant de s'adapter à des changements d'environnement tels que le vent, les vagues ou le changement de la position des obstacles.

Remerciements

Nous souhaitons exprimer notre gratitude envers l'équipe de Sirehna pour avoir orchestré et supervisé la compétition. Nous tenons tout particulièrement à remercier chaleureusement Floriane Ripoche, Antoine Leporc, Guillaume Neron, Étienne Clopeau, Léo Farina et Nathan pour leur précieuse assistance, leurs conseils éclairés et leur accompagnement lors des différents workshops. Leur contribution a été essentielle dans notre parcours et a grandement enrichi notre expérience durant cette compétition.

Hugo Ferraud, Tom Damerose, Ethan Delage