POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF3995

Projet de conception d'un système informatique

Documentation du projet répondant à l'appel d'offres no. A2024-INF3995 du département GIGL.

Conception d'un système aérien d'exploration

Équipe No **105**

Marc Jodel Dumesle ALCINDOR	2000081
Bailly Jonathan Clovis YARO	2128882
Zakarya KHNISSI	1989641
Samuel KOUAKOU	2068795
Axelle LETIEU	2152361 A. T.

Septembre 2024

1. Vue d'ensemble du projet

1.1 But du projet, porté et objectifs (Q4.1)

Le but de ce projet est de concevoir et de développer une solution technique adaptée à l'appel d'offres proposé par l'Agence Spatiale, visant à mettre en place un système d'exploration multi-robot autonome. Ce système permettra à une équipe de deux robots d'explorer une pièce d'un bâtiment de taille moyenne en utilisant une caméra et un capteur laser, tout en rapportant les informations recueillies à une interface opérateur basée sur le Web. Les membres de l'équipe projet auront pour objectif de développer cette solution durant la période impartie, en s'appuyant sur des concepts clés acquis dans les projets intégrateurs précédents, notamment en gestion de projet, développement logiciel embarqué et robotique. La planification sera principalement contrainte par les livrables à produire selon l'échéancier établi, tout en intégrant les nouvelles technologies nécessaires à l'implémentation des fonctionnalités demandées.

Un autre objectif majeur de ce projet est d'acquérir et approfondir les connaissances techniques liées à l'interaction entre la partie embarquée des robots, la station au sol et la simulation des systèmes. Cela comprendra l'intégration d'un simulateur pour tester les fonctionnalités avant leur déploiement sur les robots physiques. Le projet permettra également de professionnaliser le travail à travers la documentation, la gestion du code, l'estimation des coûts et la mise en place d'un calendrier de développement détaillé, suivant les exigences de l'appel d'offres. Les livrables attendus comprendront une version partielle du système avec des fonctionnalités de base, suivie d'une version finale où toutes les fonctionnalités demandées seront implémentées et prêtes à être déployées sur des robots physiques.

1.2 Hypothèse et contraintes (Q3.1)

On peut identifier plusieurs hypothèses et contraintes liées à ce projet. Tout d'abord, il est attendu que l'équipe de développement, bien que confrontée à des défis techniques et à de nouvelles technologies, soit capable de proposer une solution satisfaisante pour l'Agence. Cela demandera aux membres de l'équipe d'acquérir rapidement des compétences techniques spécifiques, notamment en ce qui concerne l'utilisation des robots AgileX Limo et des technologies associées. De plus, les spécifications fournies par l'Agence doivent être suffisamment claires et détaillées pour éviter toute ambiguïté lors du développement. Les exigences sont énoncées dans un document officiel, garantissant ainsi que l'équipe se concentre sur les aspects cruciaux du projet sans perdre de temps en clarifications inutiles. Le temps imparti à l'équipe pour réaliser ce projet a été jugé suffisant, à condition qu'elle adopte une organisation efficace, incluant un travail collaboratif en dehors des heures de cours et des laboratoires afin de respecter les délais.

En ce qui concerne les contraintes, elles sont nombreuses et strictes. D'abord, le prototype doit impérativement être implémenté en utilisant les deux robots AgileX Limo fournis par l'Agence, sans possibilité de déroger à cette règle. Ensuite, la communication entre la station au sol et les robots physiques doit obligatoirement passer par un réseau WiFi fourni par l'Agence, ce qui impose une contrainte technique importante. Les robots ne peuvent utiliser que les capteurs installés par l'Agence, à savoir l'IMU, la caméra 3D, la caméra RGB et le lidar, pour accomplir leurs tâches. La station au sol, pour sa part, devra être un ordinateur portable ou un PC, sans autres options matérielles.

Sur le plan logiciel, les robots doivent obligatoirement être programmés sous Ubuntu, installé sur leur ordinateur de bord, bien que l'usage de machines virtuelles, comme Docker, soit autorisé pour faciliter le développement. L'interface utilisateur utilisée sur la station au sol doit être identique, que ce soit pour les robots physiques ou pour les robots simulés, avec des ajustements uniquement pour les fonctionnalités spécifiques à chaque système. Le contrôle des robots se fera entièrement à bord de ces derniers, et la station au sol ne devra envoyer que des commandes de haut niveau, telles que le lancement de missions ou des mises à jour, sans dicter les mouvements précis des robots. Enfin, toutes les composantes logicielles, à l'exception de celles embarquées sur les robots physiques, devront être conteneurisées avec Docker, afin de garantir la reproductibilité du système, d'éviter les problèmes de compatibilité, de faciliter le déploiement, et d'assurer une évaluation standardisée et équitable pour tous les étudiants.

Ces contraintes imposent un cadre strict au projet, mais garantissent que le produit final sera robuste, reproductible et adapté aux exigences de l'Agence.

1.3 Biens livrables du projet (Q4.1)

Le projet sera divisé en trois livrables, conformément aux directives de l'Agence. Le premier livrable, attendu pour le vendredi 20 septembre 2024, correspondra à la Preliminary Design Review (PDR). À ce stade, l'équipe devra remettre un prototype préliminaire, basé sur la proposition initiale soumise en réponse à l'appel d'offres. Ce prototype démontrera les fonctionnalités de base du système, mais ne sera pas encore complet.

Ensuite, le vendredi 1er novembre 2024, le deuxième livrable marquera la Critical Design Review (CDR). Ce livrable présente un système partiellement fonctionnel, tel que décrit dans le document "Système d'exploration multi-robot : Exigences techniques". Le système devra inclure certaines fonctionnalités clés, mais ne sera pas encore entièrement finalisé.

Enfin, le mardi 3 décembre 2024, le dernier livrable sera évalué lors de la Readiness Review (RR), marquant ainsi la fin du projet. À cette étape, l'équipe devra remettre un produit final entièrement fonctionnel, incluant toutes les fonctionnalités décrites dans le document d'exigences techniques. Ce livrable devra être complet et prêt pour une utilisation réelle, conformément aux attentes de l'Agence.

2. Organisation du projet

2.1 Structure d'organisation (Q6.1)

Pour le projet de cette session, après plusieurs réunions d'équipe, nous avons opté pour une structure d'organisation collaborative et flexible. En effet, afin d'assurer une répartition équitable des tâches, mais aussi pour veiller à ce que toute l'équipe contribue pleinement au projet, nous avons séparé les tâches en deux sous catégories: techniques et administratives.

Certains membres de l'équipe seront chargés de la coordination et de la gestion du projet. L'objectif avec ce rôle est de nous assurer que les réunions hebdomadaires, les échéances et la répartition des tâches soient respectées. Ils auront également la responsabilité des rapports d'avancement hebdomadaires à communiquer aux responsables tous les lundis avant 9h. Il s'agit d'un rôle important afin d'assurer la progression efficace du projet.

Une autre partie des membres de l'équipe sera en charge du développement de la station au sol. Cela inclut le développement de la partie client, serveur, la création et la gestion de la base de données associée, mais également la liaison avec le logiciel embarqué.

D'autres membres travailleront en parfaite collaboration sur le code de la partie embarquée afin d'implémenter les fonctionnalités demandées mais aussi afin de veiller à optimiser continuellement les algorithmes. Enfin, toutes les fonctionnalités implémentées aussi bien sur la partie logiciel embarquée seront testées et validées par certains membres de l'équipe.
Le tableau ci-dessous illustre précisément et en détail les assignations des rôles et les responsabilités qui en découlent.

Rôle	Assignation	Responsabilité
Coordination et Gestion	● Alcindor	Gérer les tâches et les échéances, documenter les rapports.
Développement de la station au sol	● Yaro ● Alcindor ● Kouakou ● Khnissi ● Letieu	Développer le client et le serveur. Créer de la base de données. Gérer les interactions client-serveur, serveur et logiciel embarqué.
Développement de la partie embarquée	● Yaro ● Alcindor ● Kouakou	Implémenter des noeux pour la navigation et le bon fonctionnement de la mission du robot.

	● Khnissi ● Letieu	
Test et validation	● Alcindor ● Kouakou ● Yaro	Implémenter les tests pour les algorithmes. Assurer le bon fonctionnement des requis.

Figure 2: Assignations des rôles des membres de l'équipe

2.2 Entente contractuelle (Q11.1)

Pour ce projet, nous avons opté pour un contrat de livraison clé en main avec un prix ferme. En effet, ce type d'entente contractuelle exige que le contracteur, en l'occurrence notre équipe, soit responsable de la livraison finale du produit. Le paiement final ne sera alors émis que lors de la livraison et de l'acceptation du produit par le client. Après avoir pris en compte l'ensemble des exigences techniques du projet, nous avons choisi ce type de contrat pour plusieurs raisons. En effet, avec un prix ferme, nous garantissons une maîtrise du budget, afin d'éviter tout dépassement qui pourrait survenir. Nous entendons également faciliter la tâche du client en lui évitant la charge administrative pendant le développement. En effet, ce dernier n'aura qu'un suivi limité du développement des requis. En cas de modification ou de fluctuation des coûts, le client sera immédiatement averti afin d'éviter autant que possible tout retard dans le projet.

Bien que cela nous a demandé beaucoup de réflexions et une planification très rigoureuse et bien détaillée, nous confirmons notre choix de la livraison clé en main. Nous garantissons ainsi la livraison du produit final en respectant l'échéance mais aussi en maîtrisant les coûts.

3. Description de la solution

3.1 Architecture logicielle générale (Q4.5)

La figure ci-dessous montre l'architecture générale du système. On observe que les clients sont représentés par une interface utilisateur développée en Angular, qui permet aux utilisateurs de démarrer des missions et de suivre en temps réel l'état des robots. L'interface communique avec un serveur NestJS via le protocole HTTP. Le serveur est également connecté à une base de données MongoDB, où sont stockées les données des missions.

Le serveur NestJS communique avec un serveur Rosbridge à l'aide du protocole WebSocket. Ce dernier sert de pont entre le backend et les nœuds ROS 2, qui sont déployés sur les robots physiques ou dans un environnement de simulation. Les nœuds des robots ou de la simulation échangent leurs données et commandes via le protocole TCP/IP avec le serveur Rosbridge.

Tous les composants, qu'il s'agisse de l'interface utilisateur, le serveur NestJS, le serveur Rosbridge, ainsi que les nœuds de simulation et des robots, sont déployés dans des containers Docker. Le tout est orchestré à l'aide de Docker Compose, garantissant un environnement contrôlé et standardisé. Enfin, la couche système d'exploitation (OS) supporte l'exécution de Docker et des containers associés.

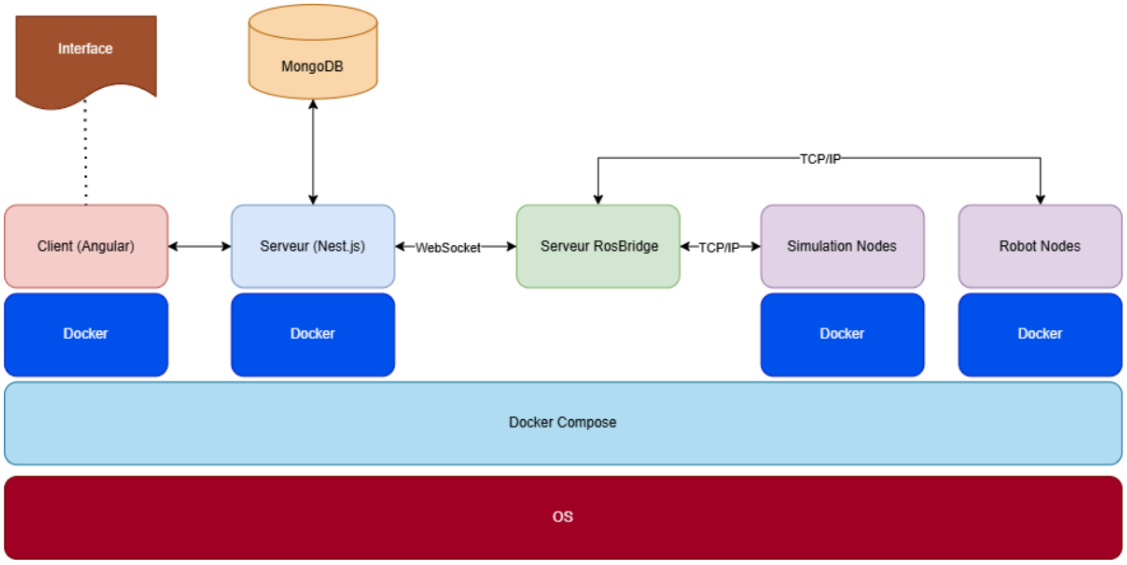

Figure 2: Diagramme de l'architecture logicielle générale

3.2 Station au sol (Q4.5)

Pour la conception du système de contrôle des robots physiques et simulés, nous avons choisi d'utiliser une architecture basée sur des services indépendants, chacun ayant un rôle bien défini pour faciliter la gestion des missions, des obstacles, et la communication entre les différents composants du système. Cette approche modulaire permet de séparer les responsabilités de manière cohérente tout en assurant une communication fluide avec le backend via une interface HTTP.

Nous avons mis en place un service pour chaque fonctionnalité clé : (**MissionService**) pour gérer les missions, (**ObstacleAvoidanceService**) pour l'évitement des obstacles, et (**RobotStatusService**) pour suivre l'état des robots (batterie et statut). Chaque service est relié à l'interface HTTP qui centralise les requêtes envoyées par l'utilisateur depuis l'interface Angular. Cette interface permet également de basculer entre les robots physiques et la simulation grâce à un bouton de sélection. Ainsi, les commandes peuvent être redirigées dynamiquement vers le mode approprié.

- Le Serveur Rosbridge joue un rôle crucial en tant que passerelle de communication entre le backend et les nœuds des robots ou ceux de la simulation, en utilisant le protocole WebSocket pour une transmission

efficace des données. Les commandes envoyées par l'utilisateur transitent par Rosbridge pour être exécutées, que ce soit dans l'environnement simulé ou réel.

- Les données des missions (comme les cartes générées et les journaux) sont centralisées dans MongoDB. Les services comme (**MapService**) et (**LoggingService**) interagissent directement avec MongoDB pour récupérer et stocker les cartes et les logs des missions. Cela permet à l'utilisateur de consulter les missions passées via (**MissionHistoryService**), qui permet d'accéder aux anciennes cartes et aux journaux des événements critiques.

Afin de garantir une maintenance et une mise à jour aisée du système, nous avons intégré un (**UpdateService**) qui permet de mettre à jour le logiciel des robots de manière centralisée. L'état des mises à jour est également suivi pour s'assurer que les robots fonctionnent toujours avec la dernière version du logiciel.

Cette architecture suit certains principes bien établis comme le patron Singleton, utilisé pour garantir qu'une seule instance de chaque service est active et gère les requêtes de manière efficace et sécurisée. Nous avons aussi appliqué le patron Controller pour centraliser les requêtes dans l'interface HTTP, ce qui permet une gestion uniforme des commandes provenant de l'utilisateur et envoyées aux différents services.

En conclusion, cette approche modulaire et découplée assure une extensibilité facile du système. Les services encapsulent bien les rôles et les responsabilités, tandis que l'interface HTTP facilite la communication entre les composants du système et les robots (physiques ou simulés).

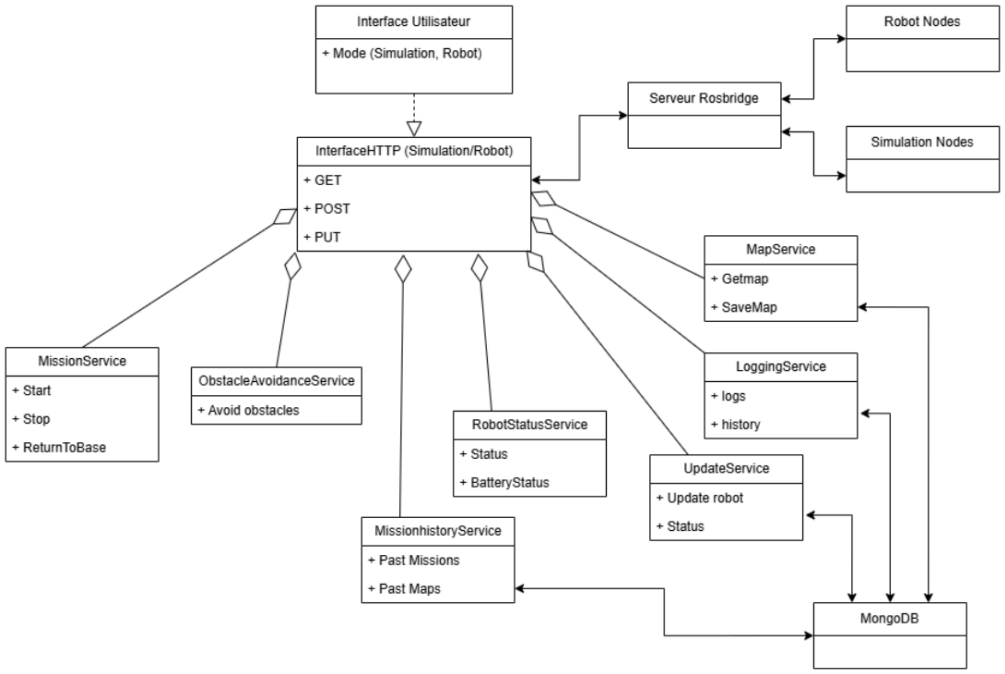

Figure 3: Diagramme de l'architecture de la station au sol

3.3 Logiciel embarqué (Q4.5)

L'architecture du logiciel embarqué pour les robots est construite autour de ROS2, un middleware qui gère la communication entre les différents nœuds du système. Les données issues des capteurs du robot sont collectées, filtrées, et transformées, puis utilisées pour ajuster les commandes de mouvement et de mission. Le tout est connecté au backend via Rosbridge pour permettre une interaction fluide avec la station au sol.

- Les capteurs incluent un LiDAR, une IMU, ainsi que des caméras 3D et RGB, qui fournissent en temps réel des informations cruciales pour la navigation et la détection des obstacles. Ces capteurs sont gérés par le gestionnaire de capteurs (**Sensor Manager**), un nœud responsable de centraliser et de transmettre les données des capteurs vers les autres nœuds du système.

Ensuite, les données sont envoyées au nœud de données capteurs, qui s'occupe de la collecte, du filtrage, et de la transformation des informations. Ce nœud joue un rôle clé en pré-traitant les données pour les rendre exploitables par les modules suivants du système, notamment le contrôle moteur et la gestion des missions.

- Le Middleware ROS 2 agit comme le cœur de l'architecture. Il assure la communication entre tous les nœuds du système, en utilisant des mécanismes de publication/souscription pour diffuser les informations, ainsi que des services et des actions pour les requêtes et la gestion des missions. Le Nœud de communication Rosbridge est directement connecté au middleware, permettant une interaction avec le backend via WebSocket. Cela permet au système ROS embarqué de recevoir des commandes de la station au sol, telles que démarrer ou arrêter une mission, et de renvoyer des informations sur l'état du robot et ses capteurs.
- Le Nœud de gestion des commandes (**Robot Command Node**) reçoit toutes les commandes envoyées depuis l'interface utilisateur via Rosbridge. Il distribue ensuite ces commandes aux autres nœuds, en fonction des actions à exécuter. Par exemple, les commandes de mission sont envoyées au Nœud de gestion des missions, tandis que les commandes de déplacement ou de contrôle moteur sont envoyées au Nœud de contrôle moteur.
- Le Nœud de gestion des missions (**Mission Node**) est responsable du suivi des missions. Il s'assure que les étapes de la mission sont respectées et surveille l'évolution en temps réel. Ce nœud est capable d'interrompre une mission en cas de problème ou de renvoyer le robot à son point de départ si nécessaire.
- Le Nœud de contrôle de la trajectoire (**Trajectory Node**), quant à lui, ajuste la trajectoire du robot en fonction des données de mission et des obstacles détectés. Il calcule en temps réel la meilleure trajectoire à suivre en utilisant des algorithmes de contrôle précis, comme le PID. Ce nœud est essentiel pour s'assurer que le robot suit un chemin optimal tout en évitant les obstacles.
- Le Nœud de contrôle moteur (**Actuator Manager et Actuator Node**) est celui qui exécute physiquement les commandes de mouvement. Il reçoit des informations du Nœud de contrôle trajectoire et des autres modules, puis ajuste la vitesse, la direction, et les mouvements du robot en fonction des commandes reçues.

L'architecture est ainsi conçue pour fonctionner de manière modulaire. Chaque nœud est spécialisé dans une tâche, ce qui permet d'assurer une gestion claire et précise des missions, du contrôle moteur, et de la communication avec le backend. Les données des capteurs sont utilisées pour ajuster en temps réel les actions du robot, garantissant une exécution fiable et réactive des missions.

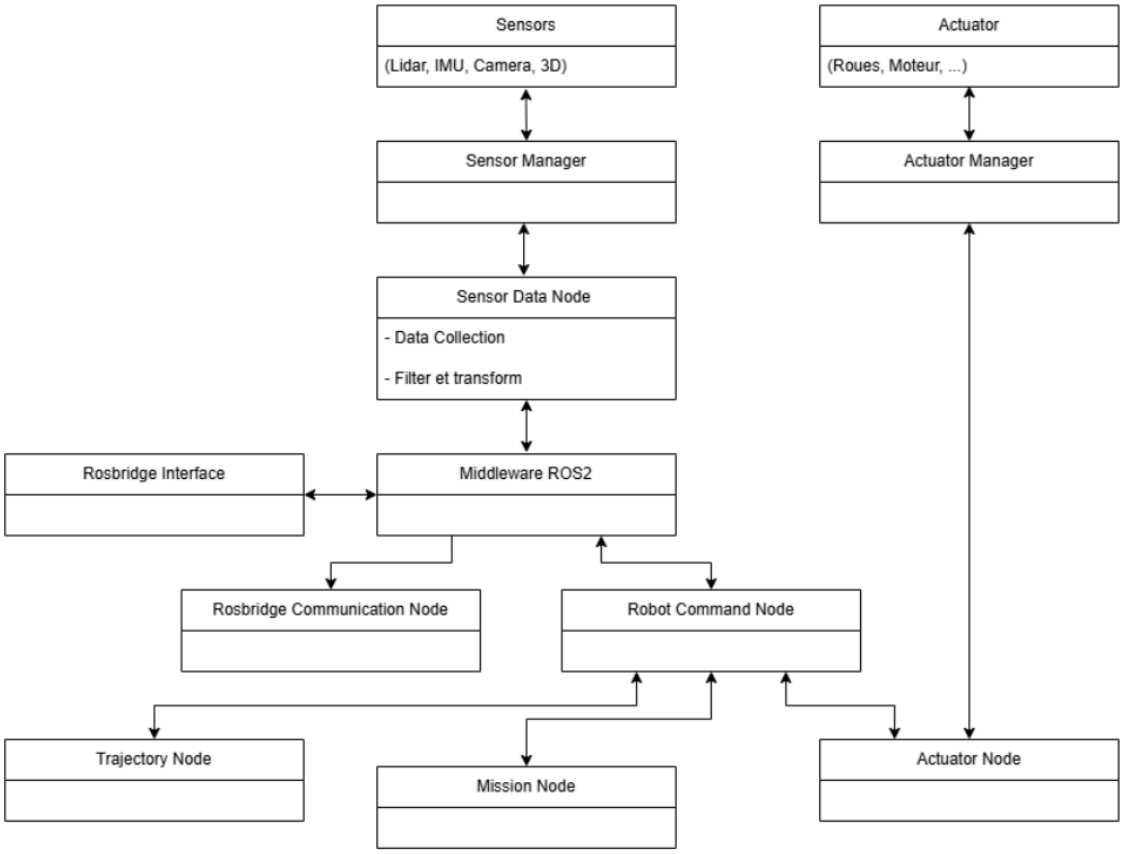

Figure 4: Diagramme de l'architecture du logiciel embarqué

3.4 Simulation (Q4.5)

L'architecture de simulation pour les robots est construite autour de ROS2, de la même manière que l'architecture du logiciel embarqué. Elle utilise un middleware pour gérer la communication entre les différents nœuds du système simulé. Les capteurs virtuels du robot, tels que le LiDAR simulé, l'IMU simulée, ainsi que des caméras 3D et RGB simulées, fournissent en temps réel des données équivalentes à celles d'un environnement réel. Ces capteurs sont gérés par le nœud de gestion des capteurs (**Simulated Sensormanager**), un nœud responsable de centraliser et de transmettre les données des capteurs simulés vers les autres nœuds du système.

Ensuite, les données sont envoyées au nœud de données des capteurs simulés (**Simulated Sensor Data Node**), qui s'occupe de la collecte, du filtrage, et de la transformation des informations des capteurs simulés. Ce nœud joue un rôle clé en pré-traitant les données pour les rendre exploitables par les autres modules du système, notamment ceux qui s'occupent de la gestion des missions et du contrôle des mouvements du robot dans l'environnement simulé.

- Le Middleware ROS2 agit, ici aussi, comme le cœur de l'architecture. Il assure la communication entre tous les nœuds simulés du système en utilisant des mécanismes de publication/souscription pour diffuser les informations et des services pour les requêtes. Le nœud de communication Rosbridge (**Rosbridge Communication Node**) est directement connecté au middleware, permettant une interaction fluide avec le backend via WebSocket. Cela permet au système simulé de recevoir des commandes de la station au sol, telles que démarrer ou arrêter une mission, et de renvoyer des informations sur l'état des capteurs simulés du robot.
- Le nœud de commandes du robot simulé (**Simulated Robot Command Node**) reçoit toutes les commandes envoyées depuis l'interface utilisateur via Rosbridge. Il distribue ensuite ces commandes aux nœuds appropriés, en fonction des actions à exécuter. Par exemple, les commandes liées aux missions sont transmises au nœud de mission simulée (**Simulated Mission Node**), tandis que les commandes de mouvement sont envoyées au nœud de trajectoire (**Simulated Trajectory Node**) et au noeud contrôleur du robot (**Simulated Motor Control Node**).
- Le nœud de mission simulée (**Simulated Mission Node**) est responsable de la gestion des missions dans l'environnement simulé. Il s'assure que les étapes de la mission sont respectées et surveille l'évolution des objectifs en temps réel. Ce nœud est capable d'arrêter ou de relancer une mission si nécessaire, en fonction des conditions dans la simulation.
- Le nœud de trajectoire (**Simulated Trajectory Node**) ajuste la trajectoire du robot simulé en fonction des objectifs de mission et des obstacles détectés. Il calcule en temps réel la trajectoire optimale, tout comme dans l'architecture embarquée physique, et envoie les commandes de mouvement au nœud contrôleur du robot (**Simulated Motor Control Node**).
- Le nœud contrôleur du robot (**Simulated Motor Control Node**) est celui qui exécute les commandes de mouvement dans la simulation. Il reçoit des informations du nœud de trajectoire (**Simulated Trajectory Node**) et des autres modules, puis ajuste la direction et la vitesse des actionneurs simulés pour reproduire les mouvements du robot dans l'environnement virtuel.

L'architecture de simulation est ainsi conçue pour refléter l'architecture du logiciel embarqué. Cela permet d'assurer une continuité entre les tests réalisés dans un environnement virtuel et le déploiement sur des robots physiques. Cette approche modulaire garantit que chaque nœud, qu'il soit simulé ou physique, est spécialisé dans une tâche précise, permettant une gestion efficace des missions, du contrôle des mouvements, et de la communication avec la station au sol. Les données des capteurs simulés sont utilisées pour ajuster en temps réel les actions du robot, garantissant une exécution fluide et fiable des missions, tout en permettant une transition rapide entre l'environnement simulé et le robot physique.

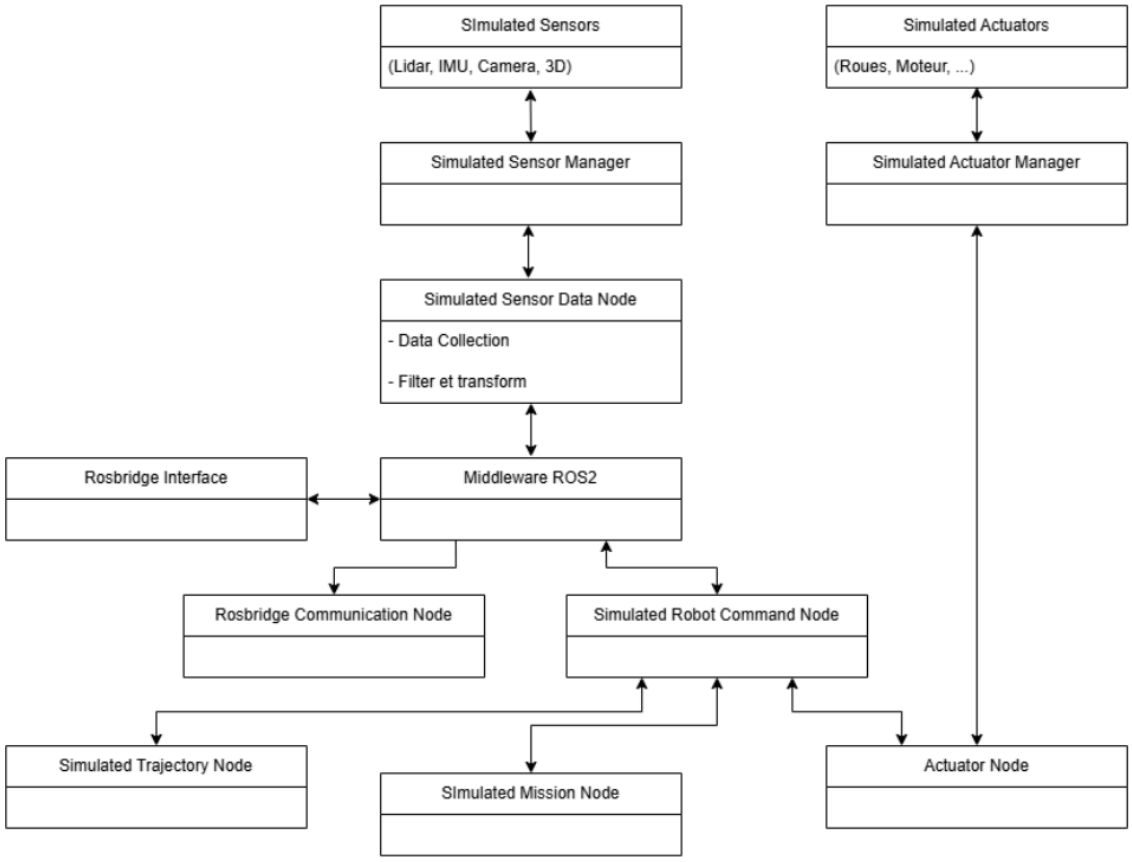

Figure 5: Diagramme de l'architecture de la simulation

3.5 Interface utilisateur (Q4.6)

Pour la conception de l'interface utilisateur pour le contrôle des robots physiques et simulés, nous avons opté pour une architecture modulaire basée sur Angular, en séparant les responsabilités entre les différents composants et services. Cette approche permet de gérer de manière claire et intuitive les interactions avec le robot physique ou la simulation via une interface unifiée. Le composant **HomePageComponent** sert de point d'entrée et permet à l'utilisateur de basculer entre le mode simulation et le mode robot physique grâce à un bouton de sélection.

Les composants (**SimulationPageComponent**) et (**RobotPageComponent**) sont dédiés respectivement à la gestion de la simulation et du robot physique. Chaque composant permet à l'utilisateur d'envoyer des commandes spécifiques, telles que démarrer ou arrêter une mission, ou encore consulter l'état du robot ou de la simulation. Ces actions sont accompagnées de commandes additionnelles comme l'identification du robot (via un signal visuel ou sonore) et la mise à jour des informations en temps réel.

Pour assurer la communication avec le backend, un service central appelé (**Simulation/RobotService**)interactions via des requêtes HTTP (Post, Get, Put) permettant de centraliser les communications avec le système ROS 2. Le composant (***SimulationPageComponent***) s'occupe des commandes pour le mode simulation en utilisant le service pour démarrer, arrêter, identifier et mettre à jour le statut des robots simulés. De même, le composant (***RobotPageComponent***) effectue ces actions pour les robots physiques en interagissant avec le service pour gérer les commandes et récupérer leur statut

Le backend centralise toutes les requêtes envoyées par les composants frontend. Il gère les commandes, qu'il s'agisse de démarrer une mission ou de récupérer l'état actuel des capteurs, et transmet ces informations au système ROS 2 ou à la simulation via **Rosbridge**. Ce dernier joue un rôle crucial en permettant la communication entre le backend et les nœuds du robot physique ou simulé, assurant ainsi une transmission fluide des données à travers un protocole WebSocket.

Pour la conception visuelle et la gestion des composants interactifs de l'interface utilisateur, nous avons choisi d'utiliser **Angular Material** et **Tailwind CSS**. **Angular Material** fournit un ensemble de composants UI prêts à l'emploi et bien intégrés avec Angular, tels que des boutons, des formulaires et des barres de navigation, ce qui simplifie le développement de l'interface utilisateur et garantit une cohérence visuelle. En complément, **Tailwind CSS** est utilisé pour la personnalisation des styles, en nous permettant d'appliquer facilement des classes utilitaires pour une mise en page flexible et réactive. Cela garantit que l'interface est non seulement fonctionnelle, mais également intuitive et moderne, tout en réduisant le temps nécessaire à la conception CSS.

L'utilisation d'Angular Material et Tailwind CSS assure également une grande adaptabilité de l'interface aux différents types d'appareils (ordinateurs, tablettes, etc.), en offrant une conception qui s'ajuste automatiquement à la taille de l'écran. Cela permet aux utilisateurs de contrôler les robots de manière fluide, que ce soit sur un poste de travail ou un appareil mobile.

Cette architecture modulaire et découplée assure non seulement une extensibilité facile, mais également une gestion efficace des services. Chaque service encapsule une responsabilité bien définie, tandis que les contrôleurs centralisent les interactions avec le backend. Cela permet une gestion claire des commandes envoyées à la simulation ou au robot physique, tout en garantissant une communication fluide et réactive entre les différents composants du système, le tout avec une interface utilisateur moderne et intuitive grâce à Angular Material et Tailwind CSS.

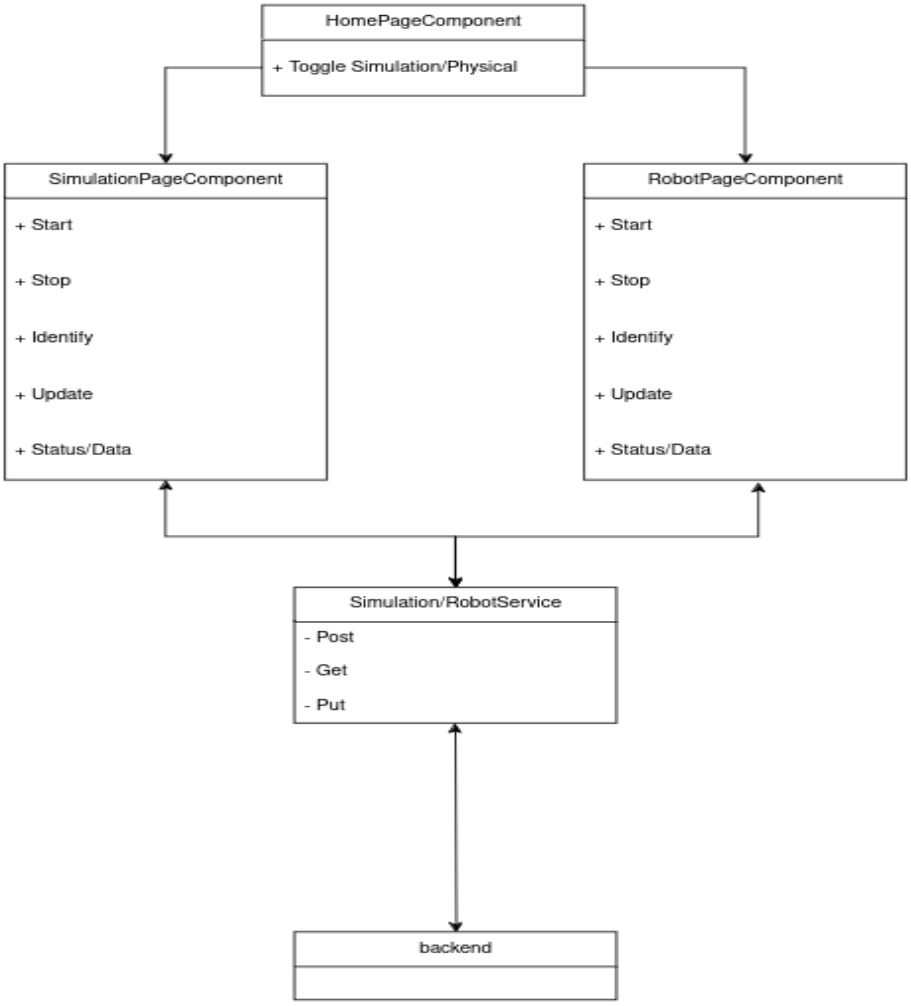

Figure 5: Diagramme d'architecture de l'interface utilisateur

3.6 Fonctionnement général (Q5.4)

Pour faire fonctionner le système à partir du code développé, il faut cloner le dépôt Git et avoir les permissions nécessaires pour le serveur NestJs, le client Angular et les composants logiciels MongoDB et Docker à l'aide des commandes appropriées (*npm install, docker compose up*). Ensuite pour l'initialisation de la base de données il est nécessaire de configurer MongoDB à l'aide des scripts fournis dans le répertoire database/ du projet pour créer et initialiser la base de données qui s'occupera du stockage des informations liés aux missions des robot.

L'interface utilisateur est démarré en accédant au client Angular et se dirigeant vers l'URL fournie par le serveur NestJS et à partir de cette interface on peut lancer des missions, gérer les opérations et surveiller l'état des robots. Lorsqu'on envoie des commandes aux robots, elles transitent via le serveur NestJS vers le serveur Rosbridge et sont dirigées vers les nœuds ROS qui contrôlent les robots.

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

Pour développer le système des robots contrôlés par la station au sol, nous avons établi un budget en fonction des lots de travail de chaque membre de l'équipe et en fonction de leurs rôles. Le tableau ci-dessous résume l'estimation des coûts.

Rôle	Taux horaire	Nombres d'heures	Coût total ($ CAD)
Coordonnateur	145 $/h	126 heures (100 en gestion + 26 en développement)	**18270$**
Développeur	130$ / h	504 heures (126 heures de développement par membres)	**65520$**

Figure 6: Tableau d'estimation des coûts

En respectant les 630 heures-personnes demandées, nous concluons à un total estimé à **84450$**.

4.2 Planification des tâches (Q11.2)

Le tableau ci-dessous fournit un aperçu des principales fonctionnalités à développer ainsi que la durée estimée nécessaire pour les faire.

Tâche	Description	Durée estimée	Délai	Responsables
Amélioration l'interface de la station au sol	Rendre l'interface utilisateur plus intuitive. Placer les boutons manquants.	*1 semaine*	**27/09/24**	• Alcindor • Yaro
Évitement d'obstacles	Implémentation de la détection d'évitement d'obstacles	*1 semaine*	**03/10/24**	• Alcindor • Khnissi
Exploration autonome du robot	Implémentation de l'algorithme pour l'exploration autonome du robot (physique et simulation	*2 semaines*	**08/10/24**	• Yaro • Letieu • Kouakou

Retour à la base et gestion de la batterie	Programmation du retour à la base du robot(physique et simulation) lorsque la batterie est faible	*1 semaine*	**11/10/24**	• Kouakou • Letieu
Cartographie en temps réel	Génération des cartes à partir des données du capteur	*2 semaines*	**15/10/24**	• Alcindor • Yaro • Letieu
Affichage de la position du robot	Affichage en temps réel de la position du robot sur la carte	*2 semaines*	**19/10/24**	• Khnissi • Kouakou
Interface multi-appareils	Le système est accessible depuis plusieurs appareils	*1 semaine*	**21/10/24**	• Alcindor • Letieu
Rédaction du CDR	Rédaction du rapport à remettre	*En continu*	**30/10/24**	Tous les membres de l'équipe
Cartes 3D	Génération de cartes 3D à partir des capteurs du robot	*2 semaines*	**15/11/24**	• Khnissi • Yaro
Détection des chutes	Implémenter une prévention du robot pour éviter qu'il chute.	*2 semaines*	**22/11/24**	• Letieu • Khnissi • Kouakou
Base de données	Création et gestion de la base de données- Enregistrement des missions et des cartes générées	*2 semaines*	**05/11/24**	• Alcindor • Yaro • Khnissi • Kouakou
Finalisation et validation des fonctionnalités	Validation et tests des fonctionnalités implémentées	*2 semaines*	**29/11/24**	Tous les membres de l'équipe
Rédaction du RR	Rédaction du RR et soumission final du projet	*En continu*	**02/12/24**	Tous les membres de l'équipe

Figure 7: Tableau de planification des tâches

4.3 Calendrier de projet (Q11.2)

Le tableau ci-dessus représente le calendrier de projet que nous avons élaboré. Il fait état des différentes clés que nous avons définies afin de garantir le bon déroulement du projet mais également le respect des échéances et des différents livrables.

Phase	Date	Description
Preliminary Design review (PDR)	***20 Septembre 2024***	● Réponse à l'appel d'offre ● Connexion entre la partie embarquée et la station au sol ● Démonstration vidéo du bon fonctionnement du robot
Développement de la station au sol	***20 Octobre 2024***	● Développement du serveur et du serveur ● Assurer de façon continue une bonne connexion entre le serveur et le logiciel embarquée ● Création de la base de données
Développement de la partie embarquée	***25 Octobre 2024***	● Développement des différents nœuds du système. ● Intégration en continue de la liaison serveur et système embarquée
Critical Design Review (CDR)	***1 Novembre 2024***	● Test et validation du logiciel embarqué et de la station au sol. ● Remise du système avec des fonctionnement partiels
Readiness Review (RR)	***3 Décembre 2024***	● Test et validation en continue du logiciel. ● Remise d'un système complet avec l'ensemble des fonctionnalités du système.

Figure 8: Tableau de calendrier du projet

4.4 Ressources humaines du projet (Q11.2)

Notre équipe se compose de 5 ingénieurs dont les compétences humaines, organisationnelles et techniques représentent un atout majeur pour une atmosphère conviviale au sein de l'équipe. En effet, monsieur Alcindor notre responsable de projet, qui a la charge de la gestion globale du projet et du développement de certaines parties critiques du code, possède de l'expérience dans la gestion de projet et a une grande capacité à coordonner les équipes.

Spécialisé en systèmes embarqués, il possède les compétences en développement logiciel intégré à des systèmes embarqués. D'un autre côté, tous nos développeurs-analystes, chargés d'une grande partie de la conception logicielle du système ont des compétences avancées en conception logiciels avec une expertise accrue dans des langages comme Python, ROS, Javascript, mais aussi avec des framework comme angular en développement d'application web. Ils possèdent des connaissances solides en systèmes embarqués et dans des environnements de simulation comme avec gazebo.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

Le contrôle de la qualité est une étape essentielle dans notre processus de développement, et nous y accordons une grande importance au sein de notre équipe. Afin d'assurer un haut niveau de qualité pour chaque bien livrable, nous avons mis en place un processus de révision rigoureux. Chaque livrable doit passer par une révision systématique basée sur des lignes directrices précises. Les tâches sont réparties entre les membres de l'équipe, chacun étant responsable de certaines tâches spécifiques. La gestion des projets se fait via GitLab, où les *issues* sont attribuées aux membres de l'équipe en fonction de leurs responsabilités. Avant que le code ne soit intégré dans le projet, des *merge requests* sont créées. Ces *merge requests* sont examinées par d'autres membres de l'équipe, assurant ainsi que le code répond aux critères de qualité avant son intégration finale. Ce processus garantit que chaque contribution est soigneusement vérifiée et que les normes de qualité sont rigoureusement respectées.

5.2 Gestion de risque (Q11.3)

Ce projet est bien évidemment accompagné de nombreux risques auxquels notre équipe pourrait faire face.

- **Retard:** Le projet est séparé en plusieurs livrables ayant des échéances.Le principal risque serait d'être en retard dans le développement de nos fonctionnalités et requis. Ce retard pourrait gravement impacter les différentes remises

Solutions : Pour éviter cela, nous avons opté pour une bonne planification de notre évolution et la répartition des tâches en nous accordant sur des délais à respecter

- **Manque de ou mauvaise communication:** La gestion de projet fait également intervenir les interactions entre autres membres. Il existe un risque d'incompréhension qui pourrait entraîner des tâches mal réalisées ou non effectuées.
- **Solutions :** Nous avons décidé de favoriser une bonne ambiance basée sur une communication claire et ouverte à travers Discord et lors des réunions

- **Problèmes d'interactions entre les composantes:** Des dysfonctionnements pourraient intervenir par manque de cohésion entre les services, le frontend et le backend

Solutions: Les réunions une fois de plus nous aident à nous assurer de la synchronisation entre les composantes ainsi que les tests qui nous permettent de vérifier le comportement.

- **Problèmes de compatibilités entre les technologies :** Des problèmes peuvent survenir entre les technologies que nous utilisons: Tailwind, Angular, Angular Material). Ce qui nécessitera sûrement du temps pour y remédier.

Solution: Nous introduisons des outils de validation pour détecter s'il y a des problèmes le plus rapidement possible avant qu'il n'y ait d'impact plus grand.

- **Expérience Utilisateur de l'Interface Web :** L'interface opérateur basée sur le Web doit être efficace pour faciliter la surveillance et le contrôle des robots. Nous réaliserons des tests d'usabilité fréquents, y compris des simulations en conditions réelles, pour garantir une expérience utilisateur optimale.

5.3 Tests (Q4.4)

Comme nous l'avons mentionné dans la section précédente, les tests sont très utiles pour le développement du projet. Ils doivent être fait à tous les niveaux pour mener à bien notre évolution. Les tests sont fait au niveau du robot physique, au niveau de la simulation et de l'interface utilisateur.

Au niveau du robot physique:

Nous avons des tests d'intégration pour vérifier que le robot se connecte au backend, nous devons également tester le moteur pour guider la direction et les mouvements du robot via des tests fonctionnels.

Tâches et/ou requis	Tests
Détection des chutes	Tester les capteurs pour prévenir les chutes
Évitement d'obstacles	Vérifier les capteurs pour détecter et éviter les obstacles
Exploration autonome du robot	Vérifier le bon fonctionnement de l'algorithme d'exploration lors de la mission

Figure 9: Tableau des tâches et des tests relatifs sur le robot physique

Au niveau de la simulation:
Nous devons vérifier que les simulations sont correctes et que les données des capteurs s'accordent avec la réalité.

Tâches et/ou requis	Tests
Exploration autonome du robot	Simuler plusieurs scénarios d'exploration pour voir le comportement virtuellement.
Évitement d'obstacles	Tester la réactivité du robot devant un obstacle.
Cartographie à temps réel	Vérifier que la carte générée renvoie les bonnes informations et reflète l'environnement.

Figure 10: Tableau des tâches et des tests relatifs sur le robot simulé

Au niveau de l'interface:
Nous devons tester l'interface utilisateur et la mise à jour synchronisée des données et de la simulation sur l'interface.

Tâches et/ou requis	Tests
Interface multi-appareils	Tester si plusieurs appareils peuvent se connecter sans interférence
Affichage du déplacement du robot	Vérifier l'affichage à temps réel du robot et la qualité de
Validation des fonctionnalités sur l'interface.	Faire des essais pour s'assurer que toutes les fonctionnalités sont intégrées et fonctionnent avec différents utilisateurs.

Figure 11: Tableau des tâches et des tests relatifs sur la station au sol

Par ailleurs, pour chaque partie du code front-end, des tests sont créés avec **Jasmine** et **Karma**. Jasmine aide à écrire des tests pour vérifier que le code fonctionne bien, et Karma exécute ces tests automatiquement dans différents navigateurs. Cela permet de vérifier que tout marche correctement et de repérer les erreurs rapidement.

5.4 Gestion de configuration (Q4)

Notre projet utilise GitLab comme système de gestion de version pour suivre les modifications du code source. Nos branches portent le nom des membres de l'équipe et sont mises à jour selon leur progression individuelle.Chaque membre travaille dans une branche dédiée, et avant toute intégration dans la branche

principale, un *Merge Request* est créé afin de valider les modifications. Cela permet d'assurer un contrôle de qualité et une révision par un autre membre de l'équipe.

Les *Issues* sont créées pour chaque tâche à réaliser et sont classées avec des étiquettes correspondant au dépôt ou à la remise concernée, facilitant ainsi le suivi des développements. Le code source est organisé de manière à séparer le développement du front-end (développé avec Angular) du back-end (développé en ROS2 et Python), pour avoir une architecture modulaire et facilement maintenable.

Les informations sur l'évolution des tâches et des modifications apportées au code sont directement suivies via GitLab, notamment à travers les *Issues* et les pipelines d'intégration continue. Cela permet de documenter automatiquement les changements et de garantir que les différentes étapes du projet sont correctement suivies et tracées.

5.5 Déroulement du projet (Q2.5)

Dans notre équipe, nous sommes actuellement à la remise du PDR et nous respectons plutôt bien nos délais. L'avancée du travail est en accord avec notre échéancier, et nous avons d'ores et déjà programmé un rendez-vous pour réaliser des simulations avec le robot physique. Depuis le début du projet, plusieurs aspects se sont déroulés de manière satisfaisante. L'organisation de l'équipe et la proactivité de certains membres ont permis de maintenir un travail fluide, et nous avons réussi à adopter des normes et des règles de communication qui ont grandement facilité l'avancement du projet.

Cependant, nous avons également rencontré des difficultés. Il n'est pas toujours facile de trouver des horaires de réunion convenant à tout le monde. De plus, étant au début du projet, il est nécessaire de prendre le temps de justifier certains choix techniques pour la structuration globale. Enfin, l'adaptation aux nouveaux outils et la compréhension de leur fonctionnement ont posé des défis à plusieurs membres de l'équipe.