



**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. H2024-INF3995 du département GIGL.

Conception d'un système aérien d'exploration

Équipe No 106

*Anh Pham
Anas Barbouch
Andy Tran
Nour Asmani
Justin Lefrançois*

Mars 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 un logiciel pour une équipe de deux robots afin que ces derniers explorent leur entourage de façon efficace.

1. Le résultat de cette exploration devra être reporté sur une interface web auquel seront connectés les robots.
2. Une carte en deux dimensions doit afficher tous les obstacles détectés par les robots à l'aide de leurs différents capteurs.
3. L'interface web doit permettre à ses utilisateurs de pouvoir démarrer, arrêter ainsi que suivre l'avancement d'une mission des robots.
4. En plus de communiquer avec la station au sol, les robots devront communiquer entre eux à l'aide d'un signal 2.4 ou 5 GHz afin de s'informer de leur progression et de différentes informations.
5. Enfin, un simulateur Gazebo doit être disponible afin de pouvoir simuler le comportement et l'algorithme d'exploration des robots sans avoir les avoirs physiquement.

En complément, la réalisation de ce projet doit s'assurer de respecter différentes contraintes prédéterminées. Par exemple, un budget nous est imposé et ne doit pas être dépassé; une contrainte de temps s'impose et finalement un esprit d'équipe doit être conservé tout au long du projet.

Au niveau des livrables, ce projet est composé de trois remises différentes. D'abord, la remise du PDR (Preliminary Design Review) qui est ce document, ensuite la CDR (Critical Design Review) et enfin la RR (Readiness Review).

1.2 *Hypothèse et contraintes (Q3.1)*

Hypothèse: simplification des tâches à l'aide de bibliothèques diverses

1. Nous croyons d'abord que plusieurs bibliothèques seront en mesure de nous aider. Par exemple, l'exploration autonome sera probablement issue d'une bibliothèque tout comme la détection d'obstacles. De plus, la carte sera probablement générée à l'aide de la bibliothèque SLAM qui permet au robot de naviguer tout en générant une map automatiquement [1]. Pour le front-end, plusieurs plugins Javascript sont disponibles afin d'accélérer grandement le processus de développement. À titre d'exemple, nous croyons utiliser un plugin intitulé 'rviz' qui est une bibliothèque avec interface ROS permettant d'afficher facilement des cartes deux et trois dimensions sur une interface web [2].

La contrainte liée à cette hypothèse semble être le fait que l'intégration des bibliothèques peut être plus ou moins complexe.

Ensuite, pour les contraintes, il y a le temps et le budget. Le temps sera très limité pour l'équipe étant donné que nous avons tous plusieurs travaux sur lesquels nous devons travailler dans les mêmes délais que ce projet. Un document hebdomadaire spécifiant ce qui a été fait et ce qui sera fait est aussi une contrainte importante pour le suivi de l'Agence Spatiale. On suppose que l'Agence Spatiale couvrira tous les coûts de développement, de matériel, de logiciel et toute dépense liée. On s'attend aussi à ce que l'équipe soit engagée durant tout le long du développement du projet. On suppose aussi que les exigences techniques seront respectées et que l'entrepôt Git permettra un bon suivi de l'avancement du développement.

1.3 Biens livrables du projet (Q4.1)

En ce qui concerne les artefacts qui devront être créés durant le projet, il y a d'abord bien entendu les trois principaux documents tel que la PDR, CDR et RR. Ces documents doivent être remis respectivement aux dates suivants :

PDR: 16 février 2024

CDR: 26 mars 2024

RR: 16 avril 2024

PDR - Plan de réponse

Le PDR est un document détaillant la proposition technique et organisationnelle pour répondre à l'appel d'offre. Pour ce projet spécifique, les requis du PDR sont :

- **Développement d'une interface web** : Cette interface doit permettre d'interagir avec le robot et, par extension, avec la station au sol (backend). Les fonctionnalités requises comprennent la capacité d'envoyer des commandes telles que "démarrer mission", "arrêter mission", et "localiser" directement les robots.
- **Simulation dans Gazebo** : Il est nécessaire de disposer d'une simulation fonctionnelle dans Gazebo, incluant les deux robots et différents obstacles pour tester les fonctionnalités dans un environnement virtuel.
- **Contrôle physique du robot** : Le système doit permettre le contrôle direct et physique du robot, assurant une interaction efficace pour les opérations réelles.

CDR - Critical Design Review

Le CDR est une étape critique de révision du design du projet avant de procéder à la réalisation. Pour ce projet, le CDR doit confirmer que :

- **Autonomie dans l'exploration** : Le robot doit être capable d'explorer son environnement de manière autonome, identifiant et évitant les obstacles sans intervention humaine.
- **Sauvegarde des données d'exploration** : Les données collectées par le robot au cours de son exploration doivent être systématiquement sauvegardées dans une base de données pour une utilisation ultérieure et analyse.

RR - Revue de réalisation

Le RR est une évaluation post-projet ou post-phase importante visant à analyser les performances et les résultats obtenus. Pour ce projet, le RR doit vérifier que :

- **Finalisation des requis** : Tous les aspects nécessaires au bon fonctionnement du système doivent être achevés, assurant que le projet répond à toutes les attentes initiales.
- **Implémentation des fonctionnalités restantes** : Cela inclut le retour automatique du robot à la base en fin de mission, la détection d'élévations négatives pour éviter les chutes, l'établissement d'une zone de sécurité pour opérer, ainsi que l'affichage de la position en temps réel et des cartes des zones explorées lors des missions précédentes.

2. Organisation du projet

2.1 Structure d'organisation (Q6.1)

Nous allons organiser le travail en utilisant la méthodologie Agile avec 3 sprints (un sprint par remise). Pendant chaque sprint, les tâches seront attribuées aux membres de l'équipe en fonction de leurs rôles et responsabilités. Nous communiquerons chaque semaine, soit en personne soit via Discord, pour discuter des progrès, résoudre les éventuels défis et assurer l'alignement sur la direction du projet.

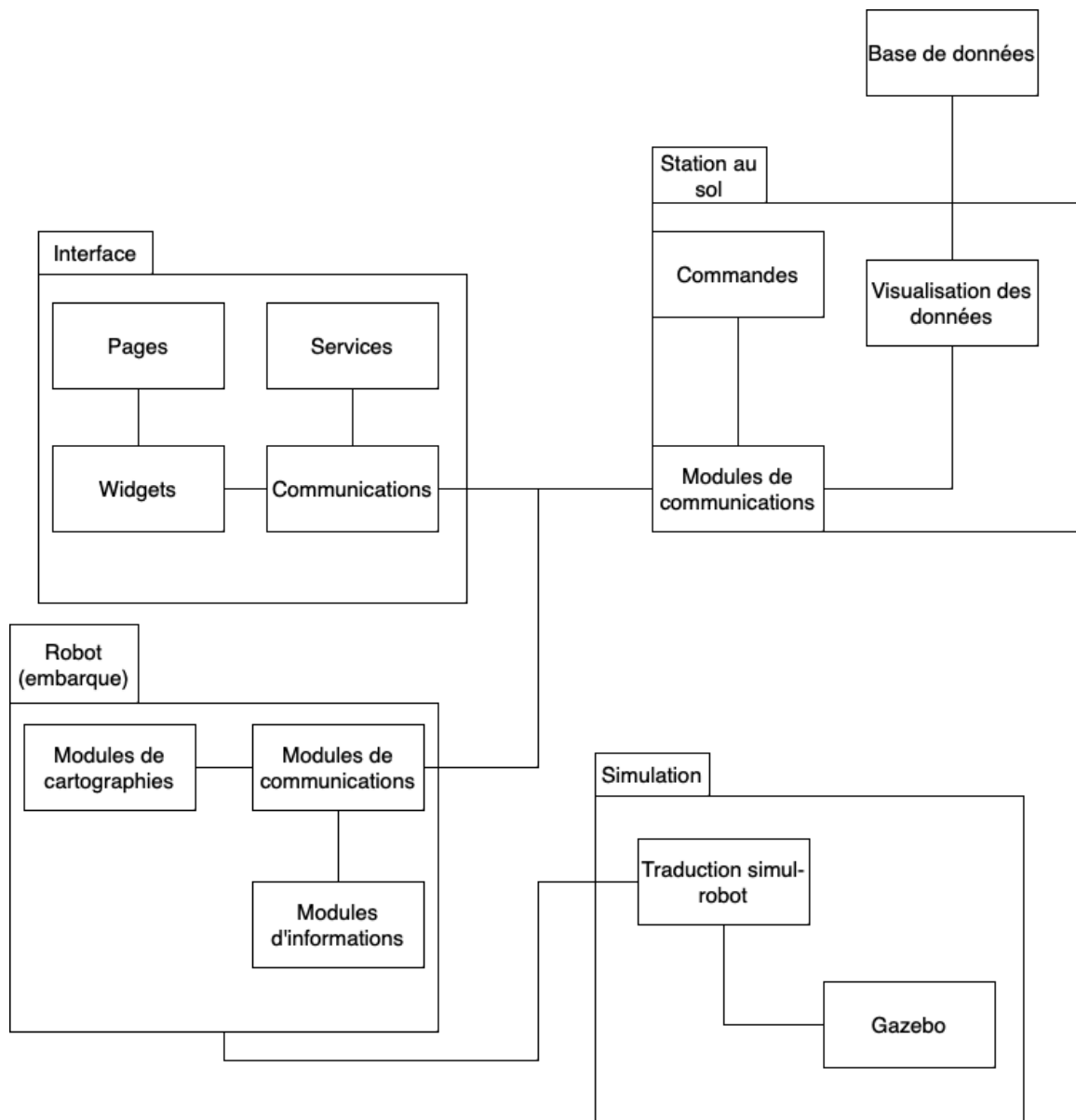
Ensuite, notre équipe a choisi une structure décentralisée en raison de la complexité prolongée du projet. Bien que cela demande plus de temps à organiser, cette approche favorise des échanges efficaces entre les membres, les maintenant impliqués et motivés tout au long du processus. Chaque membre a un rôle crucial: planificateur (Anh), vendeur (Justin), pacificateur (Andy), scrum master (Anas) et critique (Nour).

2.2 Entente contractuelle (Q11.1)

En ce qui concerne l'entente contractuelle, nous avons décidé d'aller avec le contrat à prix ferme et ce pour plusieurs raisons. Tout d'abord, il est attendu de nous de livrer un produit et non seulement un certain montant d'effort. Cette distinction est très importante car elle souligne l'importance de la réalisation concrète des objectifs du projet, plutôt que de se limiter à des contributions de travail mesurables en temps. Ensuite, nous avons une connaissance exacte de la demande grâce au document d'exigences techniques. En effet, ce document offre une compréhension précise et détaillée de ce qui est attendu du projet. Cela nous donne donc la capacité de fournir un contrat à prix ferme car nous savons exactement à quoi nous attendre et donc bien planifier les ressources, les délais et les coûts associés. Finalement, cette approche offre au promoteur, qui est sous pression d'avoir un prototype fonctionnelle pour un coût minimal, une assurance des coûts finaux.

3. Description de la solution

3.1 Architecture logicielle générale (Q4.5)



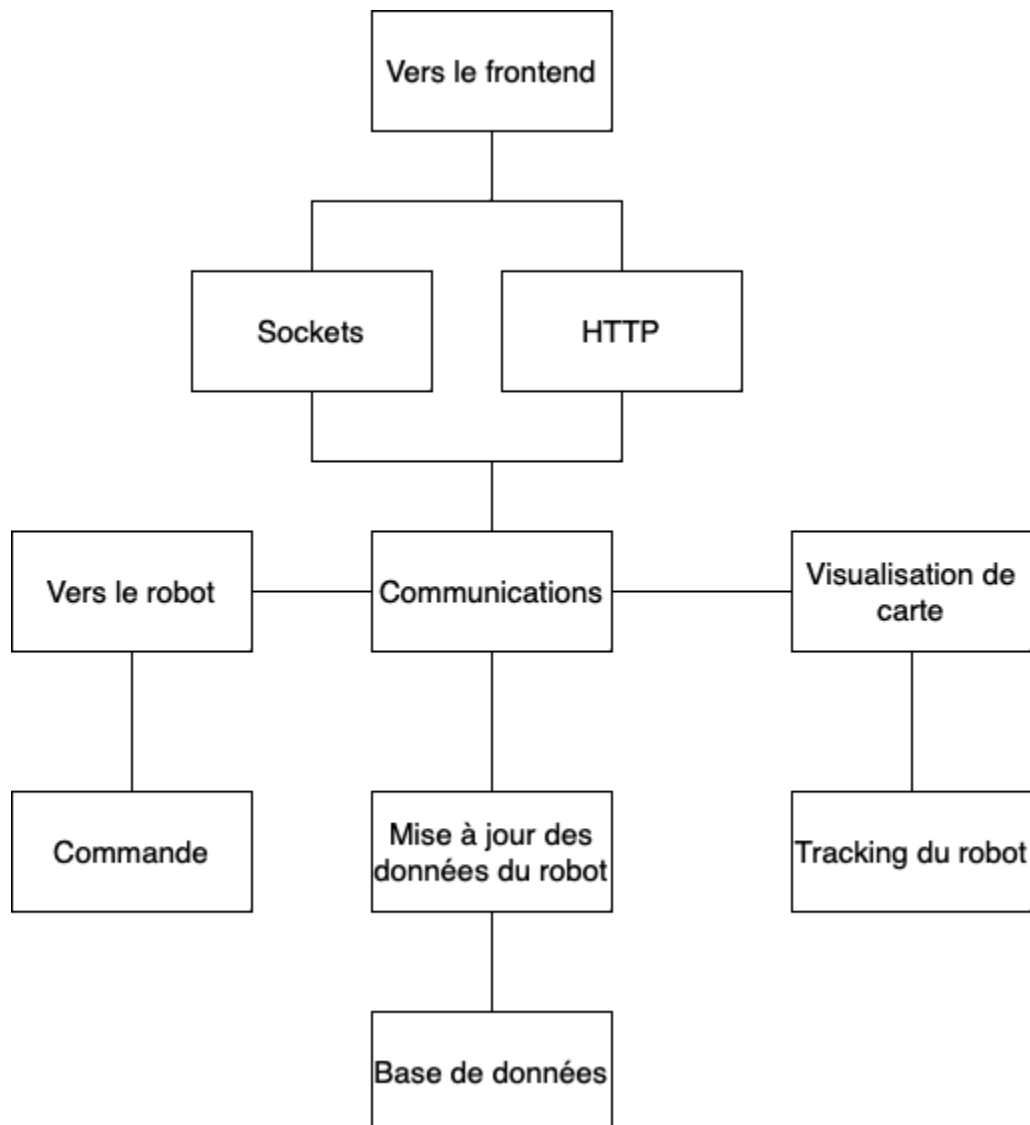
La station au sol doit communiquer les informations qu'elle a reçues du robot et de la base de données à l'interface utilisateur. Elle doit également envoyer les informations à la base de données [Requis R.F.17]. La simulation doit pouvoir exécuter la plupart des fonctions du robot. La simulation sera donc probablement le logiciel du robot avec quelques ajouts qui font fonctionner certaines choses, comme les roues. L'interface doit pouvoir afficher les informations du robots et donc, les recevoir.

(Voir si on a besoins d'ajouter autre chose)

Requis	Implémenter
R.F.1	X
R.F.2	X
R.F.3	X
R.F.4	X
R.F.5	X
R.F.6	X
R.F.7	X
R.F.8	X
R.F.9	X
R.F.10	X
R.C.1	X
R.C.2	X
R.C.3	X
R.C.4	X
R.Q.1	X
R.Q.2	X
R.F.11	
R.F.12	X
R.F.13	X
R.F.14	X
R.F.16	
R.F.17	X
R.F.18	X

R.F.19	
R.F.20	X
R.C.5	

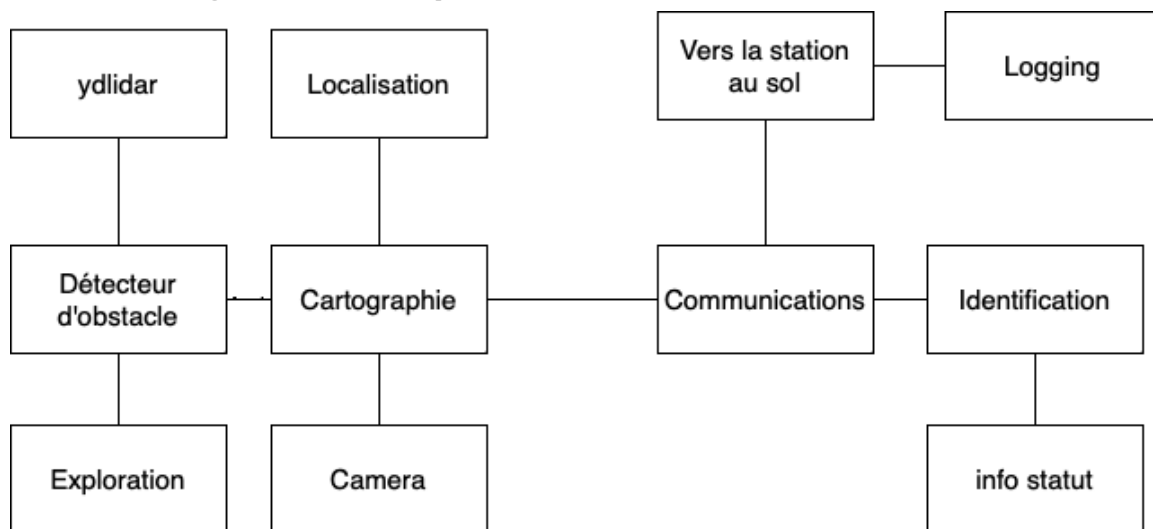
3.2 Station au sol (Q4.5)



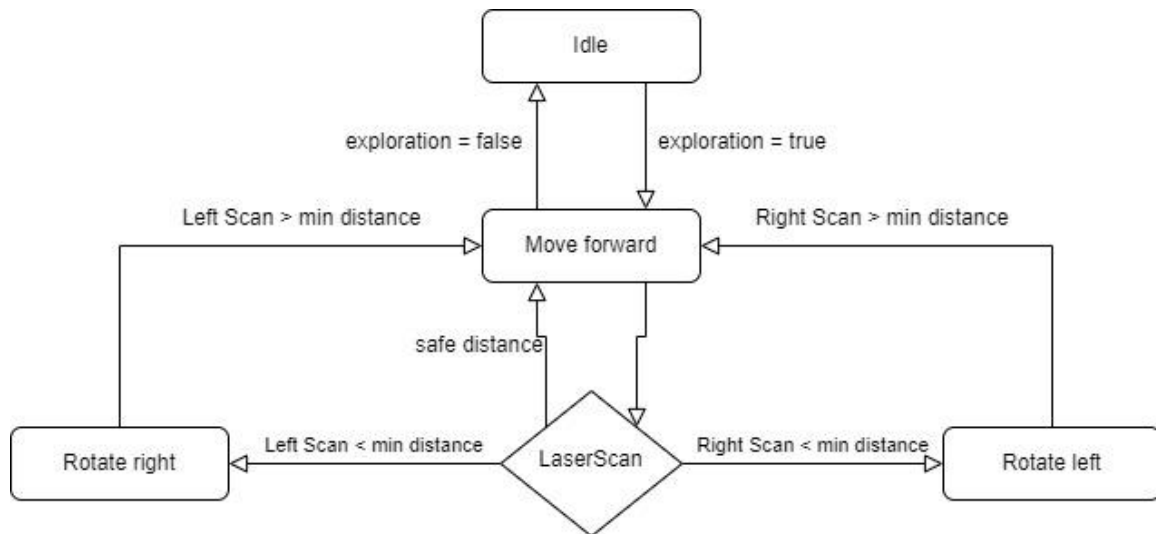
Pour communiquer avec l'interface, les protocoles HTTP et WebSockets sont utilisés. Le protocole WebSockets est utilisé dans les communications récurrentes comme les informations sur les robots. Pour le protocole HTTP, c'est surtout utiliser pour envoyer des commandes à l'appui de bouton sur l'interface web ainsi que pour recevoir de l'information une seule fois. Le backend de la station au sol va se charger de communiquer avec le robot grâce aux topics et

services ROS2 avec la librairie rclnodejs dans Node.js [3]. Le backend va recevoir les informations depuis les robots à une fréquence de ~1 Hz en étant un subscriber au topic ROS correspondant puis va les envoyer sous forme de websocket au frontend. Quand le frontend va envoyer les commandes cliquées par l'utilisateur de l'interface Web, le backend va envoyer au topic correspondant à la commande et aux robots qui doivent l'exécuter [R.F.1, R.F.2, R.F.6]. Pour la visualisation de la carte, le backend va recevoir l'information sur la carte analysée par le robot puis il va pouvoir la transmettre au frontend quand l'interface web fera la requête nécessaire [R.F.18]. Le backend se chargera aussi de mettre à jour la base de données où sont enregistrées toutes les informations nécessaires à garder [R.F.17, R.F.8].

3.3 Logiciel embarqué (Q4.5)

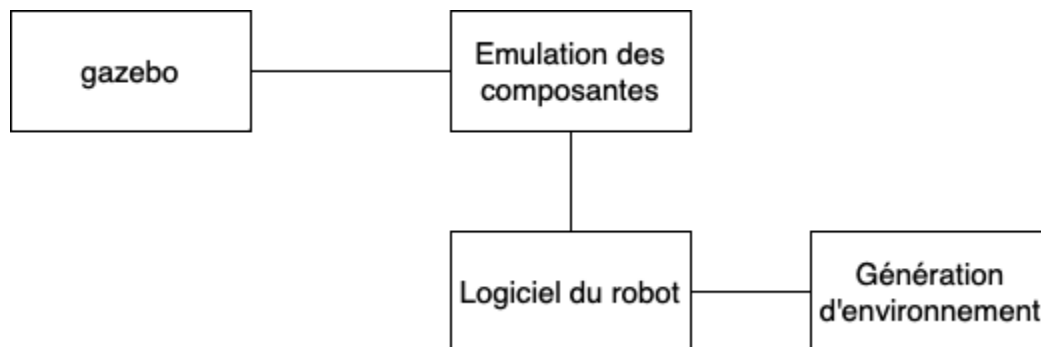


Pour cartographier la carte, nous aurons besoin d'interfacer avec le ydlidar pour trouver les obstacles et de Cartographier pour construire la carte à partir des données de ydlidar [Requis R.F.8] [4] [8]. Le ydlidar est aussi nécessaire pour détecter les changements de terrains [Requis R.F.13]. La localisation doit communiquer avec les logiciels de cartographie pour maintenir la position du robot dans la carte [Requis R.F.9]. Le robot doit pouvoir communiquer et envoyer la carte et les autres informations du robot vers la station au sol [Requis R.F.8-R.F.9-R.F.10-R.F.18]. La station doit aussi communiquer l'orientation initiale du robot [Requis R.F.12].

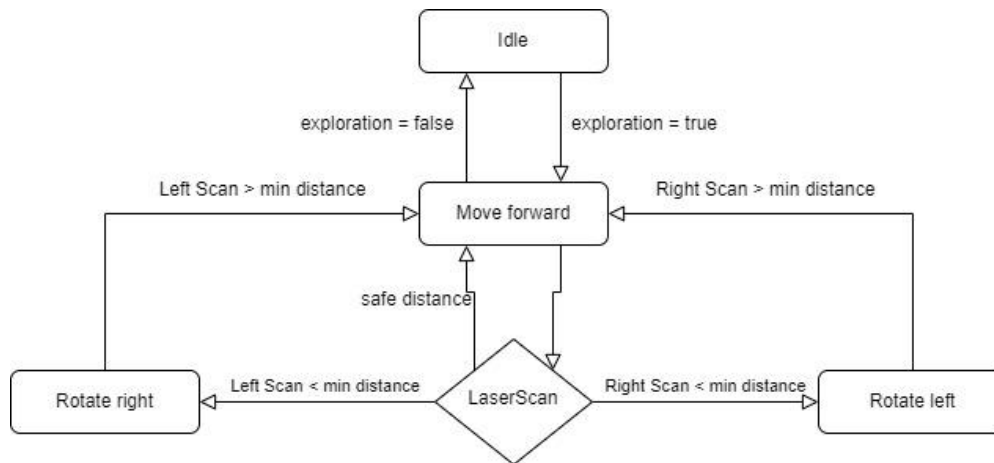


Nous avons ajouté l'exploration autonome et la détection d'obstacles pour la CDR. L'image ci-dessus représente le diagramme d'états du fonctionnement de l'exploration autonome et l'évitement d'obstacles (R.F.4 et R.F.5). Une fois que le robot reçoit mission start, il commence à bouger tout droit jusqu'à ce qu'il détecte un obstacle trop près en utilisant son senseur LIDAR. Si cet obstacle est plus près de la gauche, il tourne à droite jusqu'à ce que la distance soit sécuritaire, et il continue à avancer tout droit. Même logique pour l'autre côté. Ce cycle continue jusqu'à ce que le robot reçoive son arrêt.

3.4 Simulation (Q4.5)

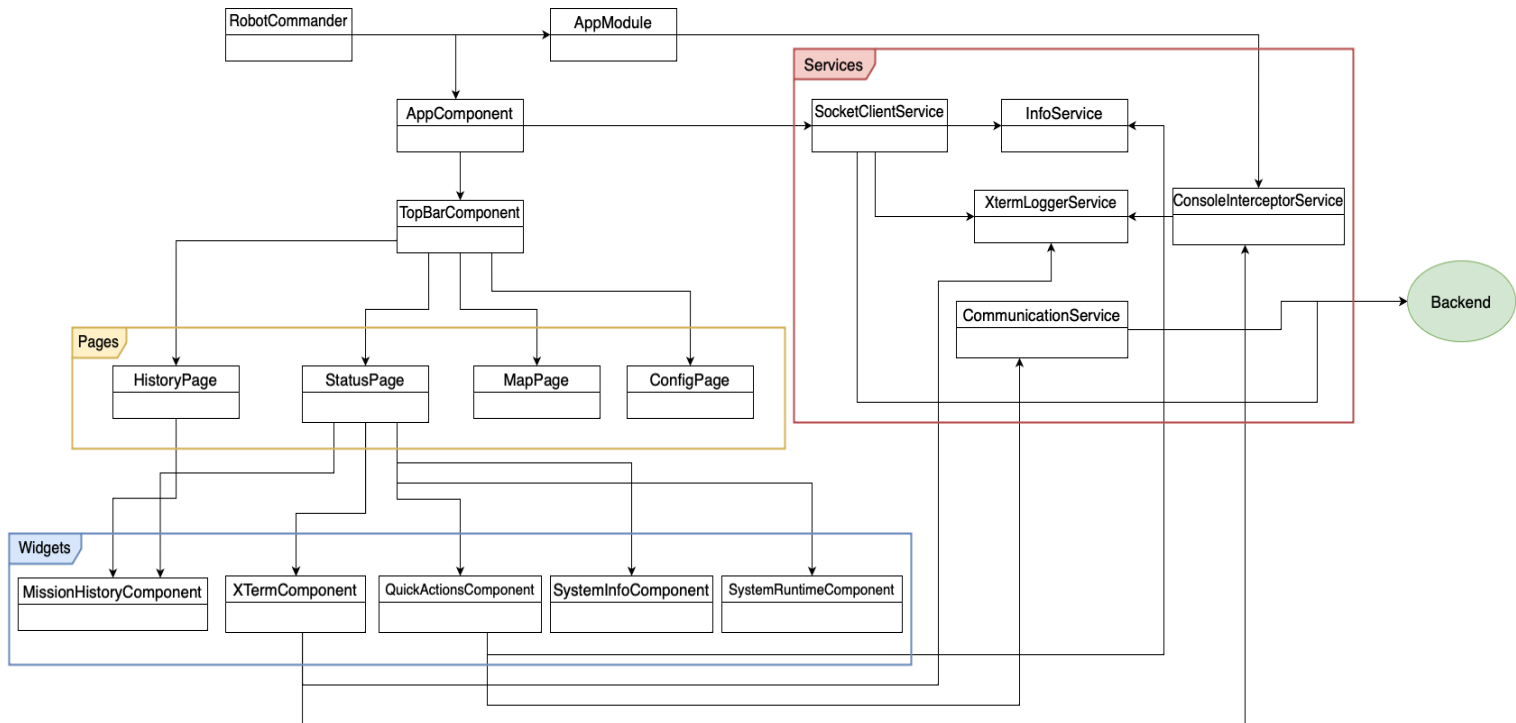


La simulation utilise toutes les composantes du logiciel utilisées dans le robot. Il y a certaines composantes qui doivent être utilisées pour contrôler des parties du robot dans la simulation. Par exemple, limo_chassis est utilisé pour contrôler les roues.



Nous avons ajouté l'exploration autonome et la détection d'obstacles pour la CDR. L'image ci-dessus représente le diagramme d'états du fonctionnement de l'exploration autonome et l'évitement d'obstacles (R.F.4 et R.F.5). L'algorithme d'exploration et de détection d'obstacles est exactement pareil qu'avec le robot physique. Une fois que le robot reçoit mission start, il commence à bouger tout droit jusqu'à ce qu'il détecte un obstacle trop près. Si cet obstacle est plus près de la gauche, il tourne à droite jusqu'à ce que la distance soit sécuritaire, et il continue à avancer tout droit. Même logique pour l'autre côté. Ce cycle continue jusqu'à ce que le robot reçoive son arrêt.

3.5 Interface utilisateur (Q4.6)



Nous avons implémenté les pages comme suit : une page indiquant le statut du/des robot(s) en marche, c'est-à-dire leur nom, temps de fonctionnement, logs, historique, boutons d'actions, etc; une page qui montre uniquement l'historique avec des options pour chaque mission effectuée; une page qui affichera dans le futur la carte générée en 2D avec des options; et finalement une page qui permettra de régler certains paramètres du/des robot(s). La `StatusPage` (page principale) utilise le plugin Javascript 'Gridster' qui permet de facilement réorganiser la vue en déplaçant les composantes (widgets) afin de rendre l'affichage facile d'utilisation pour chacun. Il sera aussi possible d'ajouter et d'enlever certains widgets au goût de l'utilisateur.

Pour les requis au niveau du UI/UX, notre interface web les respecte bel et bien. D'abord, le statut du système est toujours visible et actualisé à une fréquence de 1 Hz afin de garder l'utilisateur informé de tout changement. Ces informations sont communiquées clairement dans un langage compréhensible par quiconque tout en affichant aussi une console de log plus technique. La plateforme web est assez standard et intuitive afin que l'utilisateur puisse s'en servir facilement. Enfin, les erreurs sont bien présentées à l'utilisateur à l'aide de la console de log et l'utilisateur a la possibilité de supprimer et modifier l'historique des missions des robots en cas d'erreur. Ceci conclut donc les attentes en termes de UI/UX de l'interface web.

3.6 *Fonctionnement général (Q5.4)*

Afin de faire fonctionner notre système aisément, il faut d'abord avoir ROS2 Humble sur Ubuntu Jammy installé avec tous les requis qui vont avec ROS2. Pour le robot, il faut aussi installer YLidar-sdk pour pouvoir avoir accès au lidar installé sur le robot physique. Il faudra aussi le logiciel Ignition Gazebo Fortress pour pouvoir lancer la simulation. Les dépendances peuvent être installées en utilisant le script `install.sh` dans le repo public <https://github.com/Sh3mm/INF3995-Templates>. Le script se trouve sous `Installation/Ubuntu22.04/install.sh`. Pour ce qui est de YLidar-sdk, il peut être trouvé dans le repo YDLidar SDK[4].

Installation de ROS2 et Gazebo :

```
Unset
# installation de ROS2 et Gazebo
sudo ./install.sh

# installation de npm v18.19.0
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh
| bash
source ~/.nvm/nvm.sh
nvm install 18.19.0
nvm use 18.19.0
```

```
Unset
git clone
https://gitlab.com/polytechnique-montr-al/inf3995/20241/equipe-106/INF399
5-106.git

rosdep init
rosdep update
rosdep install --from-paths src --ignore-src -r -i -y --rosdistro humble

# installation de package python important
sudo pip install setuptools==58.2.0
sudo pip install playsound
```

Installation de YLidar-sdk :

```
Unset
# installation de YLidar-sdk
git clone https://github.com/YDLIDAR/YDLidar-SDK.git
```

```
cd YDLidar-SDK
mkdir build
cd build
cmake ..
make
sudo make install
cd ../../
rm -r YDLidar-SDK
```

Démarrage du front-end :

```
Unset
# Démarrage du frontend
cd ../frontend/robot-commander
npm i
npm start
```

Démarrage du backend :

```
Unset
# Démarrage du backend
# Il faut avoir fait colcon build dans la simulation_ws ou
# dans robot_physique_ws puis faire un source install/setup.bash
# Puis revenir dans le répertoire ../backend_server avec cd
npm i
npx generate-ros-messages
npm start
```

Démarrage du robot physique :

```
Unset
# Démarrage du robot physique
cd ../robot_physique_ws
colcon build
source install/setup.bash
ros2 launch robot_bringup robot.launch.py
```

Démarrage de la simulation :

```
Unset
# Démarrage de la simulation
cd ./simulation_ws
colcon build
source install/setup.bash
ros2 launch simulation_bringup diff_drive.launch.py
```

En cas de problème avec tf2_geometry_msgs :

```
Unset
cd /usr/local/include/
sudo ln -s /opt/ros/humble/include/tf2_geometry_msgs/tf2_geometry_msgs .
```

Par la suite, pour avoir accès à l'interface, accédez à l'adresse <http://localhost:4200> sur le navigateur de la station au sol. Vous pourrez y voir sur la première page la liste des robots connectés dans le widget 'Actions Rapides'. Vous pouvez maintenant lancer la mission pour les robots en appuyant sur le bouton Démarrer et vous pouvez l'arrêter à tout moment en appuyant sur le bouton Arrêter. Vous pouvez aussi identifier chaque robot en appuyant sur son bouton Localiser.

4. Processus de gestion

4.1 Estimations des coûts du projet (Q11.1)

Pour ce qui est du coût, l'ensemble du matériel (robots, salle pour tester) a été fourni et les outils de développement sont gratuits, donc les seuls coûts à prendre en compte pour le projet est le salaire des contracteurs. On estime qu'un développeur va travailler sur le projet 8h par semaine et que le concepteur/développeur va mettre 4h dans la conception et 4h en développement par semaine. Le salaire du développeur est 130\$/h et 145\$/h pour le concepteur.

L'estimation des coût va donc ainsi :

$$\begin{aligned}
 &4 \text{ pers.} \times 130\$ \times 8\text{h} \\
 &+ 1 \text{ pers.} \times 145 \$ \times 4\text{h} \\
 &+ 1 \text{ pers.} \times 130\$ \times 4\text{h} \\
 &= 5260 \text{ \$/semaine}
 \end{aligned}$$

5260 \\$/semaine x 11 semaines totales = 57 860 \$

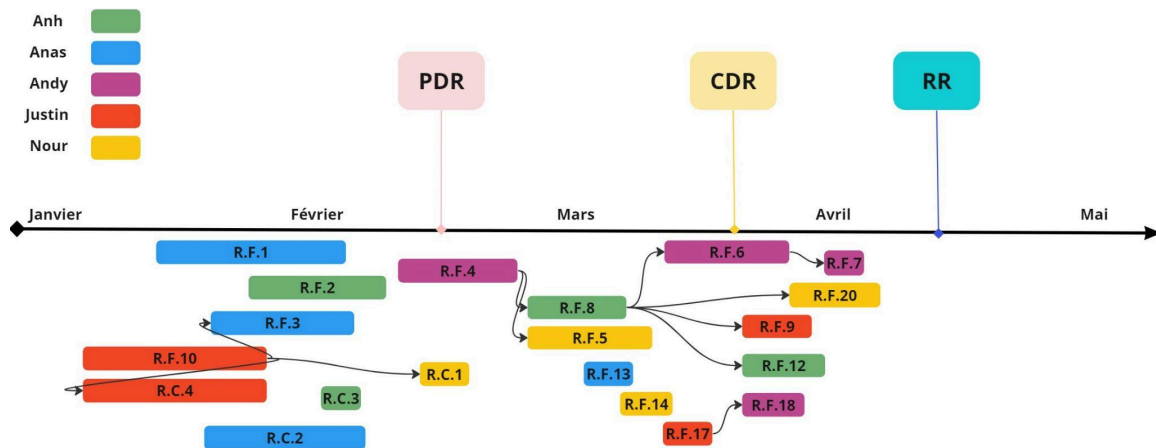
Notre estimation initiale du coût total est donc de **57 860 \$**

Par contre, après révision à la CDR, voici un ajustement de ce coût total :

$$\begin{aligned}
 &4 \text{ pers.} \times 130\$ \times 11\text{h} \\
 &+ 1 \text{ pers.} \times 145 \$ \times 6\text{h} \\
 &+ 1 \text{ pers.} \times 130\$ \times 6\text{h} \\
 &= 5995 \$/\text{semaine}
 \end{aligned}$$

Notre nouvelle estimation du coût totale est donc de 5995 \$/semaine x 11 semaines totales = **65 945\$**

4.2 Planification des tâches (Q11.2)



4.3 Calendrier de projet (Q11.2)

Semaine	Remise	Tâches
29 janvier		Simulation gazebo Début interface web Document de conception
5 février		Identification robot Fonction lancer mission Affichage état robot
12 février	PDR	Logs
19 février		Fonction retour à la base
26 février		Carte interface

4 mars		Communication entre 2 drones Algorithme de parcours
11 mars		Base de données (avec carte)
22 mars	CDR	Software update
25 mars		Fonction de spécification position et orientation initiale
1 avril		Détection élévation négative
8 avril		Zone de sécurité Logiciel complet dans docker
15 avril	RR	

4.4 Ressources humaines du projet (Q11.2)

	Anas Barbouch	Anh Pham	Andy Tran	Nour Asmani	Justin Lefrançois
Type	Concepteur, Développeur	Développeur	Développeur	Développeur	Développeur Frontend
Qualifications spéciales ou Expérience	Expérience en backend, en robotique et en gestion de projet	Expérience en développement fullstack et en devops	Expériences en système embarqués	Expériences en système embarquées	Expérience en développement web

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité (Q4)

Pour contrôler la qualité du travail dans notre équipe de robotique, nous suivons un processus que nous avons décidé en équipe. Pour chaque feature implémente, chaque membre de l'équipe font des tests pour assurer la qualité. De plus, chaque modification de code devra être approuvée par au moins un

pair. Cette révision de code permet d'assurer le fonctionnement et la qualité du code.

5.2 Gestion de risque (Q11.3)

Le projet comporte plusieurs risques qui peuvent être catégorisés en termes techniques, financiers et de gestion. Sur le plan technique, il existe un risque de que les différentes librairies qu'on planifiait utiliser ne fonctionnent pas bien avec notre matériel. Du côté financier, il est nécessaire de prendre en compte les coûts associés aux réparations ou au remplacement des pièces en cas de dommages. En ce qui concerne la gestion du projet, des risques sont liés à l'échéancier, notamment une mauvaise estimation du temps nécessaire pour terminer les tâches ou des problèmes techniques imprévus prenant du temps à résoudre. Pour atténuer ces risques, des mesures seront mises en place, telles que la manipulation prudente du robot et de respecter les règlements de la volière, puis une gestion proactive de l'échéancier en terminant les tâches précocement. De plus, des accords contractuels clairs ont été établis avec l'agence pour spécifier les responsabilités en cas de dommages physiques au robot (dépôt de 300\$).

5.3 Tests (Q4.4)

Simulation et robot physique : Nous faisons la simulation de notre code sur gazebo avant de l'utiliser sur les vrais robots afin de voir s'il y a des problèmes et des risques. Nous utiliserons aussi pytest pour faire les tests unitaires de nos propres nœuds ROS.

Tests du frontend: Nous utilisons Jasmine [9] pour faire des tests unitaires du front-end pour s'assurer que les fonctionnalités marchent.

Tests du backend: Nous utilisons Mocha [10] pour faire un ensemble de tests sur les fonctionnalités de l'interface et la communication avec le robot.

5.4 Gestion de configuration (Q4)

Nous utilisons git et GitLab pour gérer notre code. Notre répertoire GitLab est divisé en modules pour chaque partie:

INF3995-106: Entrepôt qui contient toutes les autres parties. Le fichier docker-compose permet de rouler tous les autres conteneurs et permet de démarrer le tout avec une commande.

robot_physique_ws: Le workspace ros pour contrôler le robot physique

simulation_ws: Le workspace pour faire rouler la simulation

front-end: Contient le code angular pour faire rouler l'application web de l'interface [5]. Dockerfile pour obtenir les dépendances et rouler [6].

backend_server: Le code du serveur qui interagit avec le front end et le robot. Contient aussi un Dockerfile pour obtenir les dépendances et rouler.

sounds: contient les fichiers de sons que le robot joue (ex: lors de l'identification)

5.5 *Déroulement du projet (Q2.5)*

Initialement, nous avons prévu d'intégrer deux bibliothèques couramment utilisées pour ce projet : Slamtoolbox pour la cartographie de l'environnement et Navigation2 pour l'exploration autonome [7]. Cependant, nous avons rencontré de nombreux problèmes liés aux *transforms*. Après plusieurs jours de débogage, nous avons conclu que c'était une perte de temps d'essayer de les faire fonctionner. Pour la cartographie, nous avons réussi à faire fonctionner une autre bibliothèque (cartographer-ros) [8]. En ce qui concerne la navigation, nous avons simplement décidé de créer notre propre nœud et notre propre algorithme. De plus, lors des tests avec les robots, le réseau était très lent et plantait souvent, ce qui rendait les tests de notre travail plus compliqué pour des raisons hors de notre contrôle.

En ce qui concerne les points positifs, la communication et l'organisation des membres de l'équipe est très efficace grâce à un serveur Discord que nous avons créé. Le serveur est divisé en canaux afin de rester organisé dans notre travail. Nous savons toujours sur quoi travaillent les autres membres. Et lorsque quelqu'un est bloqué sur une tâche, les autres membres en sont rapidement informés afin qu'ils puissent fournir leur support.

D'ici la remise du RR et en raison de notre rencontre HPR, nous voulons implémenter une réunion hebdomadaire (SCRUM) les mardi ou jeudi afin de se tenir au courant de qui travaille sur quoi, s'informer des problèmes rencontrés ainsi que se diviser les tâches de la semaine suivante.

6. Résultats des tests de fonctionnement du système complet (Q2.4)

Dans le frontend, nous avons mis en place des tests jasmine de base et pour l'instant tout fonctionne. Pour ce qui est de l'exploration, elle fonctionne bien en général, mais peut faire des cycles sans explorer l'entièreté du territoire parce que notre algorithme d'exploration maison bouge parfois un peu trop réactivement à l'environnement. Ensuite, pour le positionnement du robot physique, nous utilisons la position fourni par odom pour l'instant, ce qui peut

devenir inexacte après quelque temps à cause des imperfections. Nous allons changer éventuellement pour utiliser la position fournie par cartographe, qui réduit le bruit et est plus précise.

7. Travaux futurs et recommandations (Q3.5)

[RR seulement]

[Qu'est-ce qui reste à compléter sur votre système? Recommandations et possibles extensions du système.]

8. Apprentissage continu (Q12)

[RR seulement]

[Un paragraphe par membre (identifié en début de paragraphe) de l'équipe qui doit aborder chacun de ces aspects de façon personnelle:

- 1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*
- 2. Méthodes prises pour y remédier.*
- 3. Identifier comment cet aspect aurait pu être amélioré.]*

9. Conclusion (Q3.6)

[RR seulement]

[Par rapport aux hypothèses et à la vision que vous aviez du système lors du dépôt de la réponse à l'appel d'offre, que concluez-vous de votre démarche de conception maintenant que le système est complété?]

10. Références (Q3.2)

[1] "(SLAM) navigating while mapping," (SLAM) Navigating While Mapping - Nav2 1.0.0 documentation, https://navigation.ros.org/tutorials/docs/navigation2_with_slam.html (consulté Mar. 23, 2024).

[2] "rviz," ros.org, <https://wiki.ros.org/rviz> (consulté Mar. 23, 2024).

[3] "Rclnodejs," npm, <https://www.npmjs.com/package/rclnodejs> (consulté Mar. 23, 2024).

[4] Ydlidar, "YDLIDAR/YDLIDAR-SDK: Driver for receiving Yd Lidar data and more...," GitHub, <https://github.com/YDLIDAR/YDLidar-SDK> (consulté Mar. 23, 2024).

- [5] Angular, <https://angular.io/> (consulté Mar. 23, 2024).
- [6] “Accelerated Container Application Development,” Docker, <https://www.docker.com/> (consulté Mar. 25, 2024).
- [7] “Nav2,” Nav2 - Nav2 1.0.0 documentation, <https://navigation.ros.org/> (consulté Mar. 23, 2024).
- [8] “Cartographer Ros integration,” Cartographer ROS Integration - Cartographer ROS documentation, <https://google-cartographer-ros.readthedocs.io/en/latest/> (consulté Mar. 23, 2024).
- [9] “Jasmine Documentation,” <https://jasmine.github.io/> (consulté Mar. 26).
- [10] “Mocha,” <https://mochajs.org/> (consulté Mar. 26).

ANNEXES