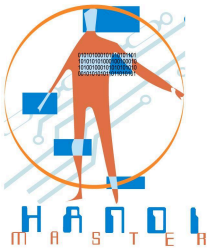
Rapport du projet AAA



EyeRobot

Master MIASHS Parcours Technologie et Handicap
Université Paris 8

Marie Fernandez
Tiffany Voguin

Professeur encadrants :
Salvatore Anzalone et Dominique Archambault

Remerciements

Nous tenons tout d’abord à remercier nos deux professeurs encadrants, Dominique Archambault et Salvatore Anzalone, pour leur aide précieuse et leur investissement, tout au long du projet. Ensuite, nous aimerions remercier Geoffrey, qui nous a été d’une grande aide lors de nos séances au lutin. Enfin, nous aimerions décerner une mention spéciale au pocophone de SNCF Connect & Tech, sans lequel, nous n’aurions jamais pû tester ROS Mobile. Merci.

Contexte

Le projet AAA vise à évaluer nos compétences dans les unités d'enseignement Agent artificiel, Android et Arduino. Pour répondre à ces attentes, nous avons décidé de développer une partie de notre projet collaboratif, qui a pour nom, EyeRobot. Ce projet a pour objectif de permettre à des enfants, atteints d'une paralysie des membres supérieurs, de télé-opérer un robot roulant, en temps réel, à l'aide de la commande oculaire, depuis un ordinateur. A cette activité nous voulons ajouter un jeu en réalité augmentée.

Pour réaliser le jeu en réalité augmentée, il est essentiel de connaître la position en temps réel du robot et d'avoir une bonne compréhension de son environnement afin de placer les objets virtuels dans des zones accessibles pour lui. Étant donné la complexité et la charge de travail que nécessite ce traitement, nous avons choisi de traiter la partie cartographie de l'environnement et la position en temps réel du robot dans le cadre du projet AAA.

Dans ce contexte, nous avons conçu et développé un prototype de robot téléopéré via une application Android qui cartographie l'espace en temps réel. Cette application permet de contrôler le robot à distance à l'aide de la diffusion du flux vidéo de la caméra placée sur le robot.

Dans ce rapport, nous détaillerons le fonctionnement de ce prototype, en justifiant nos choix de conception et nous terminerons par les perspectives futures de ce projet.

1- Choix des composants

Le choix des composants matériels et logiciels a été guidé par les besoins spécifiques de notre projet, notamment la capacité de téléopérer un robot en temps réel tout en intégrant la cartographie et sa localisation. Dans la suite de cette partie nous détaillerons les caractéristiques de l'ensemble des composants ainsi que pourquoi nous les avons choisis.

1.1 - Hardware

- Caméra Realsense D435i

La caméra Realsense D435i est une caméra de profondeur, elle intègre une unité IMU (Inertial Measurement Unit). L'IMU est utilisée pour la détection de mouvements et de rotations de l'appareil selon 6 degrés de liberté essentiels pour la localisation en temps réel. Cette caméra nous permet de récupérer toutes les données essentielles pour SLAM.

- Jetson Nano

La Jetson Nano a été retenue, après deux tests infructueux sur raspberry 4 et 5, pour sa puissance de calcul, suffisante pour exécuter des algorithmes complexes de localisation et de cartographie (SLAM) en temps réel. En effet, la Raspberry Pi 4 manquait de puissance de calcul pour ce type de tâches intensives et la Raspberry Pi 5 a été écartée, en raison d'un problème de compatibilité avec les bibliothèques de la caméra Realsense D435i.

- Moteurs DC et kit de montage robotique
- Clé Wifi
- Pont en-H L293D

Le L293D est un un circuit intégré qui permet de contrôler à la fois la direction et la vitesse d'un moteur DC à partir des signaux envoyés par le Jetson Nano.

1.2 - Software

- ROS (Robot Operating System) Melodic

Le Robot Operating System (ROS) est un framework open-source conçu pour le développement d'applications robotiques. ROS repose sur une architecture modulaire où les différentes fonctionnalités sont divisées en nœuds, qui communiquent entre eux à l'aide de messages publiés ou souscrits via des topics ou des services. ROS permet de diviser le projet en plusieurs composants indépendants (ou nœuds).
ROS a également été choisi pour sa compatibilité avec les bibliothèques nécessaires à notre projet, notamment :

- imu_filter_madgwick : pour le traitement des données de l'IMU intégrée à la caméra Realsense.
- rtabmap_ros : pour l'implémentation des algorithmes SLAM, nécessaires à la cartographie et à la localisation.
- robot_localization : pour fusionner les données de capteurs et fournir une estimation précise de la position en temps réel du robot.

    - Librealsense : driver de la caméra

- Android Studio (Java, XML, Kotlin)

Utilisé pour la conception de l'application mobile.

- Python

Utilisé dans la programmation de la Jetson Nano

- Rviz

RVIZ est une interface graphique ROS permettant de visualiser de nombreuses informations, en utilisant des plugins pour de nombreux types de topics disponibles. Pour notre projet, nous l'utilisons pour visualiser la carte 3D.

2 - Montage du robot

Pour ce prototype, nous avons opté pour un robot roulant à deux roues, complété par un stabilisateur à l'avant. La structure a été conçue pour être à la fois simple et modulable. En utilisant plusieurs kits de montage, nous avons pu assembler facilement les différentes parties du robot.

Étant donné que nous ré-utilisons les mêmes pièces dans le cadre de notre projet collaboratif et que d'autres composants devront être ajoutés, nous avons veillé à ce que le prototype soit entièrement démontable, pour faciliter les ajustements futurs.

Le robot est organisé sur deux étages. Le premier étage accueille les deux moteurs qui sont fixés et connectés à une breadboard, elle-même reliée à la Jetson Nano. Le second étage accueille la Jetson Nano dans son boîtier ainsi que la caméra montée à l'avant pour capturer l'environnement. Grâce au port USB de la jetson Nano, nous pouvons directement brancher la caméra.

Nous aurions aimé pouvoir rendre le robot totalement indépendant d'une prise secteur en le branchant à une batterie nomade. Cependant, nous avons dû changer la carte tardivement et malheureusement, la batterie et le câble d'alimentation de la jetson Nano, ne sont pas compatibles.

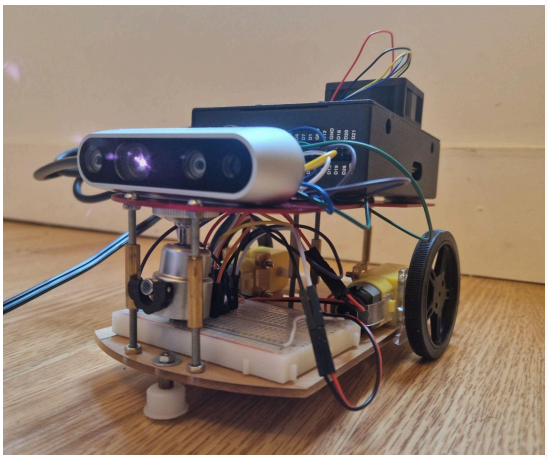


Image 1. Robot vu de face

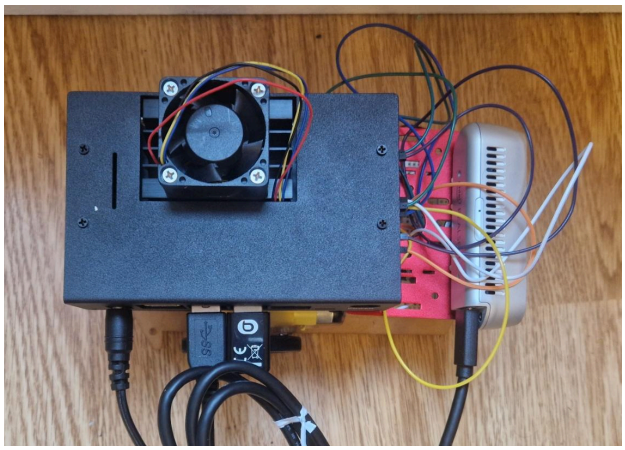


Image 2. Robot vu du haut

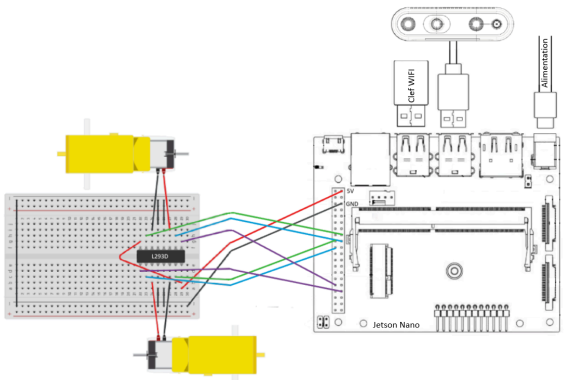


Image 3. Schéma de montage du robot

3 - Contrôle des moteurs

Les roues de notre robot étant fixes, il est nécessaire de faire varier la vitesse des moteurs afin de contrôler la direction du robot.

Il faut donc ajuster la vitesse de rotation des moteurs de chaque roue indépendamment. Pour avancer en ligne droite, il faut que la vitesse des deux roues soit égale et aille dans le sens horaire. Pour que le robot recule, il faut que les deux roues tournent à la même vitesse dans le sens anti-horaire. Pour tourner à droite, il faut que les roues aillent dans le même sens mais, que la roue droite, tourne plus vite que la roue gauche et inversement pour tourner à gauche.

Pour ce faire,nous avons utilisé les GPIO de notre Jetson nano, ainsi que Le L293D est un double pont-H, en raccordant les sorties de façon appropriée, il est possible de constituer deux pont-h. Il est ainsi possible de commander deux moteurs distincts, dans les deux sens et indépendamment l'un de l'autre. Pour cela, nous avons développé la classe moteur_controle.py.

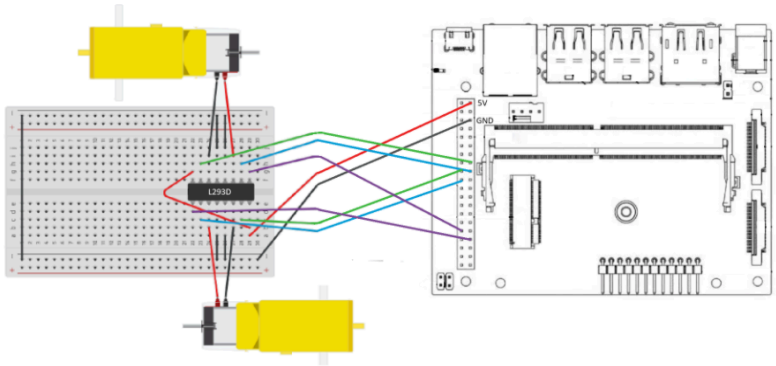


Figure 1. Schéma du montage permettant le contrôle des roues

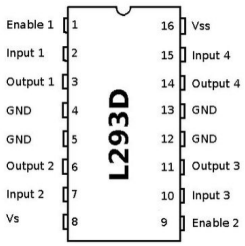


Figure 2. Schéma du microcontrôleur L293D

Voici un tableau expliquant les fonctions des différentes broches de la L293D (cf. Figure 2) et leurs utilisations pour notre projet (cf. figure 1).

Broche	Nom	Description	Utilisation
1, 9	Enable	Permet d'envoyer la tension sur les sorties du moteur.	Relié au Pin 35 et 36, met en tension les moteur au lancement de moteur_control.py
2,7,10,15	Input	Permettent de commander le sens du courant envoyé vers les Output	Relié au pin 16,18,19,21
3,6,11,14	Output	Raccordement aux moteurs.	Branché au moteur
4,5,13,12	GND	Raccordement à la masse (GND)	Raccordé au pin 3
8,16	VS, VSS	Alimentation.	La pin 1 de la jetson nano délivre 5V, les deux broches sont alimentées avec la même source.

Dans moteur_controle.py, quatre fonctions ont été créées pour contrôler les mouvements du robot :

Tourner_droite

- Envoie du courant sur la broche Pin 16 de la Jetson Nano, permettant de faire tourner le moteur droit dans le sens horaire.
- Coupe l'alimentation des broches Pin 18, Pin 19, et Pin 21, mettant à l'arrêt le moteur gauche.

Tourner_gauche

- Envoie du courant sur la broche Pin 18 de la Jetson Nano, permettant de faire tourner le moteur gauche dans le sens horaire.
- Coupe l'alimentation des broches Pin 16, Pin 19, et Pin 21, mettant à l'arrêt le moteur droit.

Avancer

- Envoie du courant sur les broches Pin 16 et Pin 18 de la Jetson Nano, permettant de faire tourner les deux moteurs dans le sens horaire.
- Coupe l'alimentation des broches Pin 19 et Pin 21 de la Jetson Nano.

Reculer

- Envoie du courant sur les broches Pin 19 et Pin 21 de la Jetson Nano, permettant de faire tourner les deux moteurs dans le sens antihoraire.
- Coupe l'alimentation des broches Pin 16 et Pin 18 de la Jetson Nano.

4 - Téléopération du robot

4.1 - Fonctionnement de l’application

L’application a un écran d’accueil, une page de connexion, un menu et enfin le jeu.
Pour que l’application fonctionne il faut que le robot (jetson Nano) et le téléphone sur lequel est l’application soient connectés au même réseau WIFI. De plus, il est nécessaire de connaître l’adresse IP de la Jeston Nano.

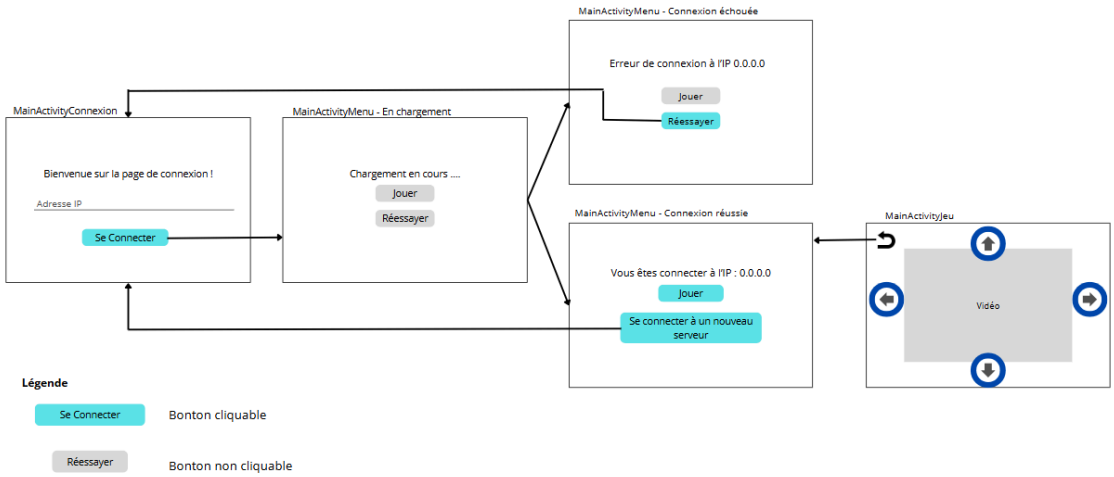


Figure 3. Maquette fonctionnelle de l'application android

4.1.1 - Page d'accueil

La page d'accueil demande à l'utilisateur de rentrer l'adresse ip du robot afin de s'y connecter.

La page d'authentification permet de récupérer l'adresse IP pour connecter l'application au robot. L'activité gère les erreurs éventuelles. Le champ editText affiche un message d'erreur si l'utilisateur :

- valide un champ vide ;
- valide un champ avec un format incorrect d'adresse ip, grâce aux regex.

En cas d'erreur, un message d'erreur apparaît indiquant à l'utilisateur ce qu'il doit corriger dans sa saisie, avec un exemple, conformément au RAAM (Référentiel des critères d'accessibilité pour les applications mobiles).
Le label du champ de saisie prévient les erreurs en indiquant le format attendu (xxx.xxx.xxx.xxx) et que le champ est obligatoire. La page a un titre et les éléments interactifs sont focusables.

4.1.2 - Page de menu

L'adresse IP saisie par l'utilisateur est fournie en paramètre à l'activité MainActivityMenu. celle ci appelle la class SocketManager.java qui s'occupe de connecter l'application au robot.

Plus précisément, socketManager.java se connecte à deux serveurs par un protocole TCP (Transmission Control Protocol). TCP est un protocole fondamental d'Internet qui assure une connexion fiable entre deux points. Quand les données sont envoyées via TCP, le protocole garantit qu'elles arrivent dans le bon ordre et sans erreur.

socketManager.java établie deux connexions TCP distinctes :

- Une pour la vidéo (“videoSocket”) qui reçoit un flux continu de données

- Une pour les commandes (“CommandSocket”) qui permet d'envoyer des instructions au serveur

Cette classe utilise la classe “Socket” de Java qui implémente ce protocole TCP. Chaque socket crée un canal de communication bidirectionnel, comme un tuyau, permettant d'envoyer et recevoir des données de manière fiable entre l'application Android et les serveurs distants.

La gestion de deux sockets séparés permet d'isoler le trafic vidéo (qui peut être volumineux) du trafic de commandes (qui nécessite une transmission rapide et fiable), optimisant ainsi les performances de l'application.

Pendant la tentative de connexion, l'écran géré par MainActivityMenu affiche "Connexion en cours...". Si la connexion réussit, l'interface affiche l'adresse IP du serveur auquel l'utilisateur est connecté. En cas d'échec, un message d'erreur apparaît à l'écran.

L'interface comporte deux boutons principaux. Le bouton "Joueur" devient actif uniquement si la connexion est établie et permet de lancer une nouvelle activité (MainActivityPlay). Le bouton "Retour" change de comportement selon l'état : s'il y a une connexion active, il permet de se déconnecter et de revenir à l'écran précédent pour choisir un autre serveur ; en cas d'échec de connexion, il affiche "Réessayer" et permet de retenter la connexion.

4.1.3 - Page de jeu

L'interface de jeu de l'application Android a été développée pour permettre une communication efficace avec le robot. Le composant principal est un SurfaceView, choisi pour afficher le flux vidéo en temps réel. Ce composant est adapté à notre utilisation, car il gère l'affichage de contenus graphiques dynamiques de manière performante.

L'interface intègre quatre boutons directionnels : haut, bas, gauche et droite. Ces boutons utilisent des écouteurs tactiles (OnTouchListener) pour détecter les interactions de l'utilisateur. Chaque bouton détecte deux états : l'appui (ACTION_DOWN) et le relâchement (ACTION_UP). Un tableau de booléens mémorise l'état de ces boutons pour suivre les commandes de l'utilisateur. Ce système permet une détection précise des actions de l'utilisateur et leur transmission au serveur.

La communication avec le serveur s'effectue via une boucle de contrôle qui s'exécute toutes les 100 millisecondes. Cette fréquence établit un équilibre entre la réactivité et la charge

réseau. Les commandes sont transmises en chaînes de caractères formatées, par exemple "haut:true" lorsqu'un bouton est pressé, ce qui facilite leur interprétation par le serveur.

Le traitement du flux vidéo suit un protocole défini. Chaque image est précédée d'un en-tête de quatre octets qui encode sa taille, permettant au système de savoir exactement combien de données il doit lire pour obtenir l'image complète. Les images sont reçues au format JPEG, un format de compression adapté à la transmission de vidéo sur le réseau. Ces images sont ensuite décodées avec la classe BitmapFactory d'Android, qui convertit les données JPEG en bitmaps, c'est-à-dire en images que le système peut afficher. Ces bitmaps sont finalement affichés sur le SurfaceView à l'aide d'un Canvas, qui est l'outil de dessin principal d'Android.

Plusieurs threads sont utilisés en parallèle pour permettre à la fois la diffusion de la vidéo en continu ainsi que la gestion des différents boutons.

Un dernier bouton est présent sur l'interface, qui permet à l'utilisateur de revenir à l'activité précédente.

Cette architecture permet d'assurer le contrôle à distance en temps réel du dispositif, en combinant une interface utilisateur réactive et une communication stable avec le serveur distant.

4.2 - Architecture fonctionnelle de communication

Sur la Jetson Nano, la communication entre les différents nœuds (programmes exécutés sous ROS) s'effectue grâce à la publication et à la souscription à des topics. Les topics sont des canaux par lesquels les nœuds échangent des messages de différents types afin de communiquer efficacement.

Premièrement, nous lançons le fichier opensource_tracking.launch, qui démarre les nœuds nécessaires à la cartographie. Ce fichier configure également la diffusion du flux vidéo de la caméra sur le topic /camera/color/image_raw.

Ensuite, le nœud serveur.py souscrit au topic /camera/color/image_raw pour accéder au flux vidéo. Ce nœud utilise un protocole TCP/IP pour transmettre le flux à l'application Android, qui se connecte au serveur via le port 9090. L'application affiche alors le flux vidéo en temps réel sur son interface utilisateur. Grâce à l'utilisation combinée de ROS et des protocoles TCP/IP, et si la connexion Wi-Fi est de bonne qualité, la latence reste faible (généralement inférieure à 3 ms). Cette faible latence est suffisante pour permettre à l'utilisateur de contrôler le robot sans impact perceptible, notamment grâce à la vitesse lente nécessaire à la cartographie.

Pour ce qui est du contrôle du robot, au démarrage, le nœud serveur.py créer un second serveur sur le port 9091, dédié à la gestion des commandes. L'application Android envoie en continu l'état des boutons de direction, false si le bouton est relâché, true si le bouton est pressé. Ces informations sont publiées par serveur.py sur des topics spécifiques, par exemple /btn_press_haut.

De son côté, le nœud moteur_controle.py est abonné à ces topics et appelle la fonction correspondante dès qu'un message true est reçu.

Voici un exemple d'utilisation :

L'utilisateur appuie sur le bouton "haut" pour avancer.
L'état true du bouton "haut" est envoyé au serveur de commandes via le protocole TCP/IP.
Le nœud serveur.py publie le message true sur le topic /btn_press_haut.
Le nœud moteur_controle.py, abonné au topic /btn_press_haut, appelle la fonction avancer().

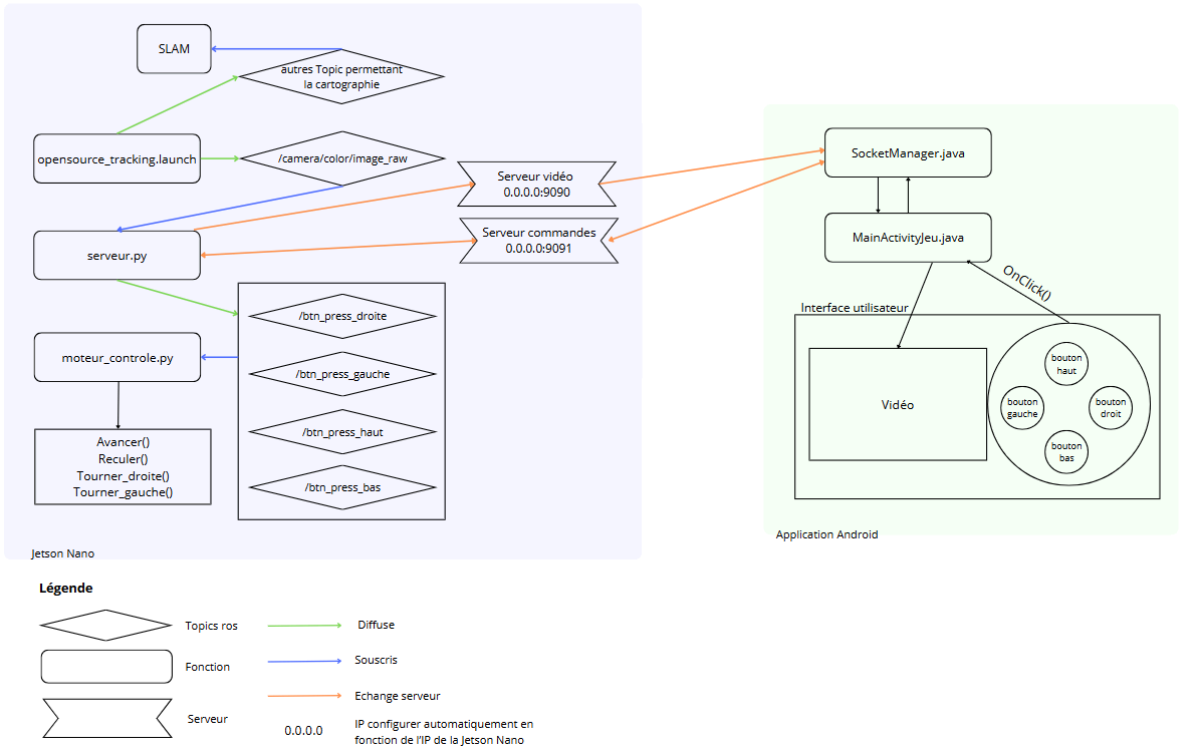


Figure 4. Schéma de l'architecture des communications entre le robot et l'application

4.3 - Autre option envisagée

Une solution envisagée a été d'utiliser les bibliothèques java de ros dans l'application android pour commander le robot. Nous avons trouvé une application, ROS Mobile, sur le PlayStore. La particularité de cette application est qu'elle utilise ROS, en tant que protocole de communication. En effet, cette application permet de se connecter au robot via ssh et de le diriger grâce à la création de widgets qui ne sont rien d'autres que des nœuds qui peuvent souscrire à des topics et recevoir des messages. Nous avons, grâce à cette application, pu tester que ROS fonctionnait pour commander notre robot, avec les boutons directionnels et recevoir un flux vidéo via le topic camera/color/image_raw.
Nous avons donc envisagé d'utiliser ROS comme protocole de communication car le code source de l'application, ROS Mobile, est sur Gitlab et nous avons, pendant un moment,

nourri l’espoir que nous arriverions à récupérer le code et à le modifier. Cependant, les fonctions ROS sur android sont obsolètes depuis la sortie d'android 11. Donc, nous avons dû changer de stratégie et utiliser un socket classique.

5 - La cartographie de l’espace en temps réel

5.1 - Description du fonctionnement de SLAM.

Cette section se concentre sur la partie “agent artificiel” de notre projet.
Afin de satisfaire nos objectifs de développement d’un jeu en réalité augmenté et de répondre aux attendus de l’évaluation, nous avons intégré à notre projet un algorithme de cartographie en temps réel, SLAM (Simultaneous Localization and Mapping).

L’objectif est que l’algorithme, en utilisant des capteurs, calcule des points en temps réel pour générer la carte. Cela nous permettrait de placer des objets en réalité augmentée à des positions spécifiques de l’espace pour créer un jeu vidéo.

En effet, SLAM est un algorithme qui utilise des capteurs de profondeur, des caméras et des unités de mesure inertielle (IMU) pour construire une carte de l’environnement sans avoir de connaissances préalables sur celui-ci. Les données IMU (Unité de Mesure Inertielle) sur la caméra sont utilisées pour mesurer et suivre les mouvements de la caméra dans l'espace. L'IMU est composée de capteurs qui mesurent la vitesse angulaire (rotation) et l'accélération linéaire dans les trois axes (x, y et z).

Dans le contexte de la caméra RealSense2, les données IMU fournissent des informations cruciales sur les mouvements de la caméra pendant son utilisation, ce qui permet d’améliorer la précision des processus de localisation et de cartographie (SLAM – Simultaneous Localization and Mapping). En effet, lorsque la caméra se déplace, l'IMU aide à déterminer son orientation et sa position en temps réel.

Une fois les données capturées, l’algorithme de SLAM transforme ces informations en nuages de points. Chaque mesure issue des capteurs est convertie en un point tridimensionnel dans l’espace, créant ainsi une représentation en 3D de l’environnement autour du véhicule. Les nuages de points générés sont ensuite analysés pour identifier et modéliser les obstacles présents dans l’espace. Ainsi, le SLAM reconnaît ces obstacles en fonction de leur densité, de leur position et parfois de leur forme. Pour améliorer la précision de la carte et de la localisation, des corrections et des ajustements des points sont effectués. Ces ajustements sont réalisés avec des techniques comme la fermeture de boucle (loop closure), qui permet de corriger les erreurs de localisation accumulées pendant les déplacements, en particulier sur de longues distances. Lorsque le véhicule repasse par une zone déjà visitée, le SLAM ajuste la carte et la position du véhicule pour les rendre plus exactes.

5.2 - Intégration de SLAM

Pour implémenter cet algorithme, nous avons utilisé la caméra stéréo RealSense D435i, qui offre des bibliothèques compatibles pour SLAM. Comme toute caméra stéréo, elle permet de mesurer la profondeur et d'enregistrer des informations à l'aide des unités de mesure inertielle (IMU), ce qu'une caméra classique ne permet pas. Plus précisément, la caméra stéréo capture deux images simultanées sous des angles légèrement différents, et en comparant les différences entre ces images (disparité), elle peut calculer la profondeur des objets dans la scène grâce à une triangulation stéréoscopique. Chaque pixel de l'image correspond alors à un point 3D dans l'espace. Ces données brutes, représentant l'environnement autour du véhicule, sont ensuite utilisées pour créer une carte précise de cet espace (voir explications plus haut).

Bien que la realsense soit déjà équipé de librairies pour faire tourner un algorithme comme SLAM, nous avons dû compléter avec trois autres librairies ROS :

- imu_filter_madgwick
- rtabmap_ros
- robot_localization

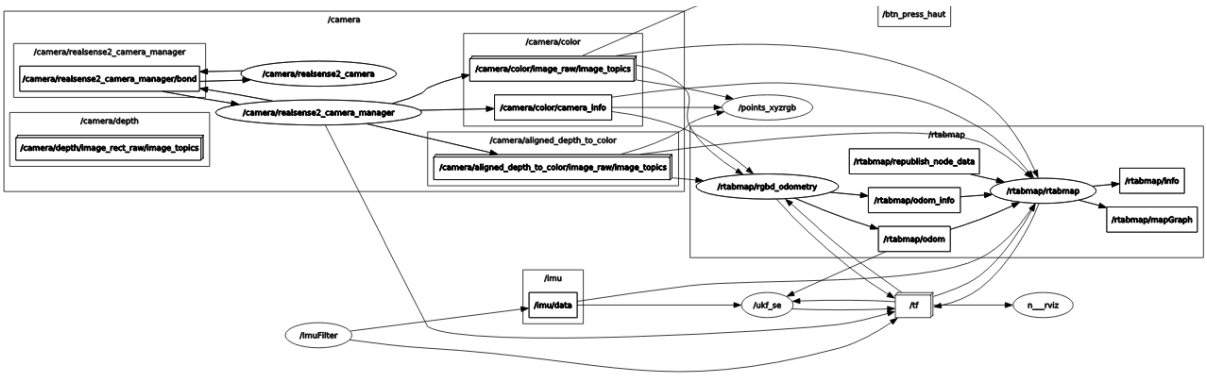


Figure 5. Schéma représentant les noeuds, les sujets et les messages pour SLAM

Le fichier realsense2_opensource_tracking.launch initialise plusieurs nœuds ROS interconnectés qui assurent la cartographie de l'espace et la localisation du robot depuis l'acquisition des données jusqu'à leur traitement et leur visualisation.

Tout d'abord, le gestionnaire de la caméra, identifié par le nœud /camera/realsense2_camera_manager, joue un rôle central en contrôlant la caméra et en publiant les données brutes via le topic /camera/realsense2_camera. Ce gestionnaire génère plusieurs topics dédiés dont les images couleur disponibles sur /camera/color/image_raw, les informations spécifiques à la caméra sur /camera/color/camera_info, ainsi que les données de profondeur accessibles via /camera/aligned_depth_to_color.

Les données acquises sont ensuite traitées par d'autres nœuds dont les nœuds gérant le traitement des données inertielles (imu), tels que /imu et /imuFilter, pour une analyse précise des mouvements. Ces informations sont ensuite transmises au nœud rtabmap via le topic /rtabmap/grid_odometry, qui permet d'assurer la localisation et le positionnement du robot. Le nœud /tf , qui souscrit et publie sur rtabmap est chargé de calculer les transformations

nécessaires entre les différents repères ou frames, garantissant ainsi une cohérence spatiale dans les données.

Enfin, en ce qui concerne la cartographie et la visualisation, le nœud rtabmap construit et met à jour en continu une carte de l'environnement. Les données associées sont publiées sur les topics /rtabmap/mapData et /rtabmap/mapGraph. Pour l'interface utilisateur, ces informations sont visualisées en temps réel dans RViz qui souscrit à /tf, ce qui permet à l'utilisateur de visualiser la carte qui se construit en temps réel.

5.3 - Résultats

L'algorithme SLAM, utilisé avec la caméra RealSense D435i, nous a permis de créer une carte en temps réel de l'environnement. Le système parvient à générer une représentation tridimensionnelle de l'espace, ainsi qu'à localiser le robot.

Néanmoins, plusieurs limitations importantes ont été identifiées lors des tests. Le temps de traitement pour générer la carte est relativement long, ce qui peut causer des problèmes dans certains contextes.

La présence d'objets en mouvement dans l'environnement du robot, tel qu'une personne qui marche, pose un défi. Ces éléments dynamiques perturbent la cohérence de la carte, compromettant parfois la précision de la reconstruction spatiale. Ces objets non fixes peuvent alors, à tort, être considérés comme des obstacles fixes et peuvent aussi apparaître plusieurs fois dans la carte en fonction de la direction de l'objet.

La position initiale du robot est importante. Au redémarrage de la cartographie, le robot doit être placé exactement au même point de départ que lors du premier lancement. Si ce n'est pas le cas, le système ne peut pas se localiser dans l'espace. Il considère alors que sa nouvelle position de départ est la même que l'ancienne et met à jour la carte.

Les mouvements rapides posent également problème. Lorsque la caméra bouge trop vite, le système perd le suivi et doit être réinitialisé. C'est pour cela que la vitesse du robot à été considérablement réduite.

Ces contraintes impacteront le développement du jeu en réalité augmentée. Le système fonctionne bien dans des conditions contrôlées, mais il faudra adapter le jeu pour tenir compte de ses limites. Il est nécessaire de définir une procédure d'initialisation précise, de prévoir comment gérer les pertes de suivi, d'adapter le jeu aux contraintes de vitesse de déplacement et à la présence d'objets mobiles.

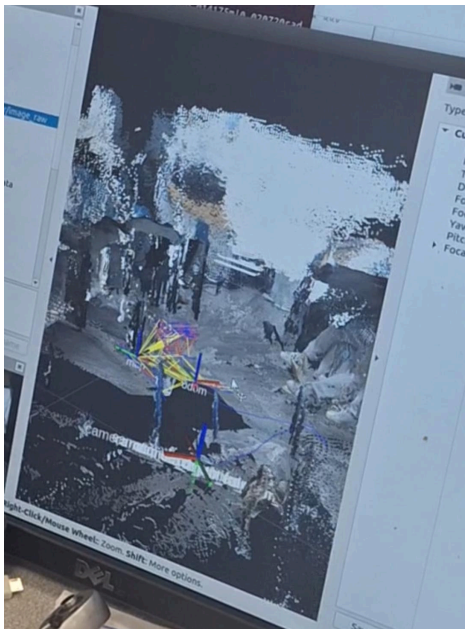


Image 4. Carte partielle de la salle C224.
Les segments colorés représentent la trajectoire du robot.

5.4 - Difficultés rencontrées

Nous avons rencontré plusieurs difficultés pour intégrer SLAM. En effet, l'intégration de SLAM sur un système embarqué est particulièrement complexe, au vu de la puissance de calcul que l'algorithme demande et du matériel. (Abouzahir, 2017) montre que pour un SLAM optimal, il faut privilégier une architecture FPGA. Les cartes FPGA sont des cartes programmables bas niveau, avec de la programmation type assembleur. Seulement, cette solution n'était pas envisageable dans le cadre de notre projet.

En effet, il faut une bonne connaissance du matériel car il s'agit de programmation bas niveau, nous n'avions donc pas la compétence de programmer des cartes FPGA et ne pouvions pas la développer en 3 mois. Nous avons donc dû opter pour des cartes utilisant des architectures qui mixent GPU et CPU dont on sait, qu'elles sont moins performantes pour gérer des tâches à faible latence comme SLAM.

Nous avons donc commencé le projet avec une raspberry pi 5. Cependant, elle était incompatible avec les bibliothèques de la realsense, nécessaires pour faire fonctionner SLAM. Ensuite, nous avons essayé de faire fonctionner SLAM avec une raspberry pi 4, qui avait pour avantage d'être compatible avec les bibliothèques de la realsense et d'avoir à peu près les mêmes caractéristiques que la raspberry 5, à savoir des ports périphériques, un processeur, une carte wifi… Mais cette fois, la puissance de calcul de la raspberry 4 s'est avérée être insuffisante pour faire tourner SLAM même en changeant les paramètres dans le fichier de configuration, qui diminuent la qualité de la carte. Notre dernier choix s'est avéré être le bon afin de surmonter ces contraintes, sans passer par un ordinateur.

Notre critère était de trouver une carte avec les mêmes caractéristiques que les raspberries, mais avec une plus grande puissance de calcul. La carte qui répond aux critères est une carte Jetson nano. La seule chose qui lui manque est une carte wifi mais la présence de

nombreux ports périphériques, nous a permis de rajouter une clé wifi, afin de pouvoir la connecter à l'application android. Pour que la solution fonctionne correctement sur jetson, nous avons dû modifier quelques valeurs dans opensource_tracking.launch.

En effet, étant donné la puissance de calcul juste suffisante de la jeston nano, pour arriver à avoir une image fluide, nous avons dû modifier plusieurs paramètres de configuration, qui ont détérioré la précision de la cartographie. Nous avons diminué la taille des points dans camera.launch et augmenté la valeur de la fenêtre maximale de tolérance temporelle (en secondes), entre les messages provenant de différents topics, pour qu'ils soient considérés comme "synchronisés", de manière approximative. Il a été fixé à 0.2, ce qui augmente le délai de synchronisation entre les capteurs de la librairie rtabmap, pour que les données des odomètres soient pertinentes, et que le robot se repère dans l'espace.

6 - Conclusion et Perspectives

Nous avons rempli nos objectifs pour ce projet. En effet, nous avons développer un prototype fonctionnel de robot téléopéré capable de cartographier son environnement en temps réel. L'intégration de la caméra RealSense D435i, couplée à l'algorithme SLAM via ROS, nous permet d'obtenir une représentation tridimensionnelle de l'espace, constituant ainsi une base pour le développement futur du jeu en réalité augmentée.

Cependant, il reste des points que nous souhaiterions améliorer, pour la suite du projet.

Premièrement, lorsque nous avons commandé le matériel, nous étions sur rapsberry 5. Nous avions donc naturellement commandé une batterie externe compatible avec ce matériel. La Jetson nano est apparue dans le projet, une semaine avant le rendu et la batterie externe que nous avions pour la raspberry n'est pas compatible avec notre nouvelle carte. Donc, actuellement, notre robot est limité dans ses mouvements par la longueur du câble.

De plus, il faudrait pour la suite du projet que nous enregistrions plusieurs cartes car actuellement, seule la dernière carte s'enregistre dans la base de données. Cela permettra à l'utilisateur de jouer dans plusieurs espaces. Nous avons donc un travail sur la base de données à faire.

Il faudra également que, peu importe le point de départ du robot, la carte se construise et se précise au fur et à mesure des explorations (loop enclosure). Dans le cadre du projet collaboratif, ce travail sera essentiel pour placer les objets en réalité augmentée. En effet, pour la suite de notre travail, la carte devra être précise pour pouvoir détecter les obstacles en temps réel et placer les objets dans des endroits accessibles à notre robot, comme le sol, dans l'angle où se situe la caméra.

Enfin, l'application est également améliorable. Nous aimerions développer une nouvelle activité qui permette de connecter le robot, depuis l'application, à internet. En l'état, il se connecte automatiquement à notre point d'accès mais il faudrait pouvoir le configurer, la première fois au moins, sur l'application.