Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería de Organización Industrial

PJE - ROBOTIQUE MOBILE

Autor:

Javier Dehesa Rodríguez

Responsable de Intercambio en la Uva:

Marta Herráez Sánchez

Ecole Nationale Supérieur d'Arts et Métiers

Campus Lille

Valladolid, septiembre 2022.

TFG REALIZADO EN PROGRAMA DE INTERCAMBIO

TÍTULO:	PJE - ROBOTIQUE MOBILE
ALUMNO:	Javier Dehesa Rodríguez
FECHA:	12/01/2022
CENTRO:	Campus de Lille
UNIVERSIDAD:	Ecole National Supérieur d'Arts et Métiers
TUTOR:	Adel Olabi

Resumen:

Este proyecto consiste en tomar la plataforma Turtlebot3 y estudiar los modelos que le permiten desplazarse mediante el mapeo del entorno, para acabar generalizando las estrategias de navegación a las plataformas móviles industriales. Para ello, se estudiará la plataforma, se pondrán en marcha las herramientas de simulación asociadas y se probarán los algoritmos disponibles en la plataforma para adaptar y optimizar su funcionamiento. Esta plataforma está equipada con una interfaz principal que acepta comandos en lenguaje ROS (Robot Operating System).

Palabras claves:

ROS, SLAM, TurtleBot3, LiDAR, Raspberry Pi.

Abstract:

This project consists of taking the Turtlebot3 platform and studying the models that allow it to move by mapping the environment, in order to generalise the navigation strategies to industrial mobile platforms. To do this, the platform will be studied, the associated simulation tools will be implemented and the algorithms available on the platform will be tested in order to adapt and optimise its operation. This platform is equipped with a main interface that accepts commands in ROS (Robot Operating System) language.

Keywords:

ROS, SLAM, TurtleBot3, LiDAR, Raspberry Pi.

Table des matières

1 Mise en cohérence des objectifs du projet :

Le projet d'expertise « Robotique mobile » consiste à prendre en main la plateforme Turtlebot3 et à étudier les modèles qui lui permettent de se déplacer en cartographiant l'environnement, afin de pouvoir éventuellement généraliser les stratégies de navigation à des plateformes mobiles industrielles. Pour cela, il faudra étudier la plateforme, mettre en place les outils de simulation qui lui sont associés, et tester les algorithmes disponibles sur la plateforme afin de les adapter et d'optimiser leur fonctionnement.
Cette plateforme est munie d'une interface principale acceptant les commandes en langage ROS (Robot Operating System).

Une première étape a été de réaliser un état de l'art des techniques de SLAM (Simultaneous Localization And Mapping), permettant au robot de réaliser la tâche mentionnée ci-dessus.

Le groupe du projet est constitué de :

- Yannis FAKIR
- Javier DEHESA RODRIGUEZ

Pendant le projet nous nous sommes partagé les tâches afin de mener le projet le plus efficacement possible.

Notre professeur encadrant est :

- Adel OLABI

Nos remerciements vont à Mr BELLE et Mr DESCHODT, qui nous ont apporté leur aide et leurs connaissances tout au long de la réalisation de ce projet.

2 Planification du projet :

Pour l'organisation du travail à faire nous avons proposé une planification pour être conscient des situations suivantes :

- ✓ Établir les étapes de travail
- ✓ Prendre conscience des délais
- ✓ Faire face aux imprévus en étant conscient des conséquences
- ✓ Coordonner le travail
- ✓ Fournir des outils pour suivre les progrès de l'extérieur

Nous avons établi le planning provisionnel pour le déroulement de notre projet. Pour cela nous avons pris en compte toutes nos séances prévues pour le PJE et ensuite nous avons fait la première répartition des tâches entre les séances. On a essayé d'organiser le travail de façon que le développement du projet soit cohérent avec le déroulement de notre PJE.
Le diagramme de Gant suivant illustre cette organisation, qui a été adaptée selon l'avance du projet à cause des difficultés découverts.

Pour pouvoir bien regarder et analyser le diagramme de Gantt vous pouvez cliquer sur le lien suivant :

- https://sharing.clickup.com/g/h/kg3vq-28/954977713a0b6aa

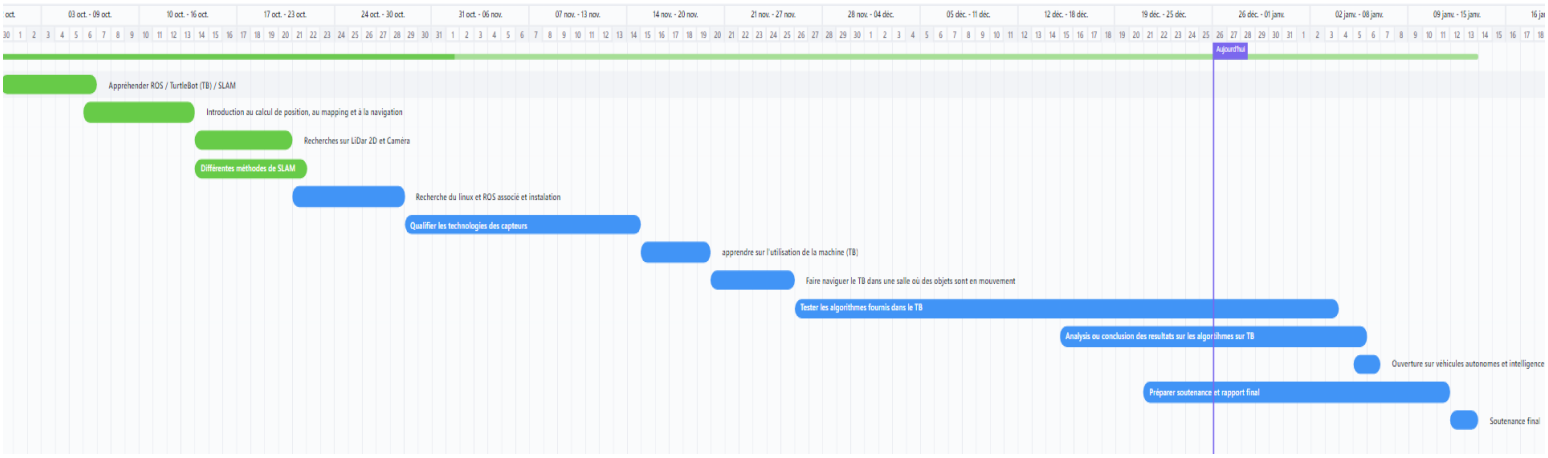

Figure 1 Planification initiale

Les taches de notre PJE que nous avons fixés au début du projet sont les suivantes :

1. Appréhender ROS/ TurtleBot (TB) /SLAM
2. Introduction au calcul de position, au mapping et à la navigation
3. Rechercher sur LiDar 2D et Caméra
4. Différentes méthodes de SLAM
5. Recherche du Linux et ROS associé et installation
6. Qualifier les technologies des capteurs
7. Apprendre sur l'utilisation de la machine (TB)
8. Faire naviguer le TB dans une salle où des objets sont en mouvement
9. Tester les algorithmes fournis dans le TB
10. Analyses ou conclusion des résultats sur les algorithmes sur le TB
11. Ouverture sur véhicules autonomes et intelligence artificielle
12. Préparer soutenance et rapport final
13. Soutenance finale.

Cependant, en raison au retard de l'arrivée du robot, nous avons été contraints de modifier les taches du projet. Ce retard a affecté et retardé notre organisation. Nous avons finalement reçu et pu manipuler notre plateforme TurtleBot 3, montée par Mr Olabi, fin novembre. Par conséquent notre planning a été modifié et nous avons réorganisé les taches á réaliser :

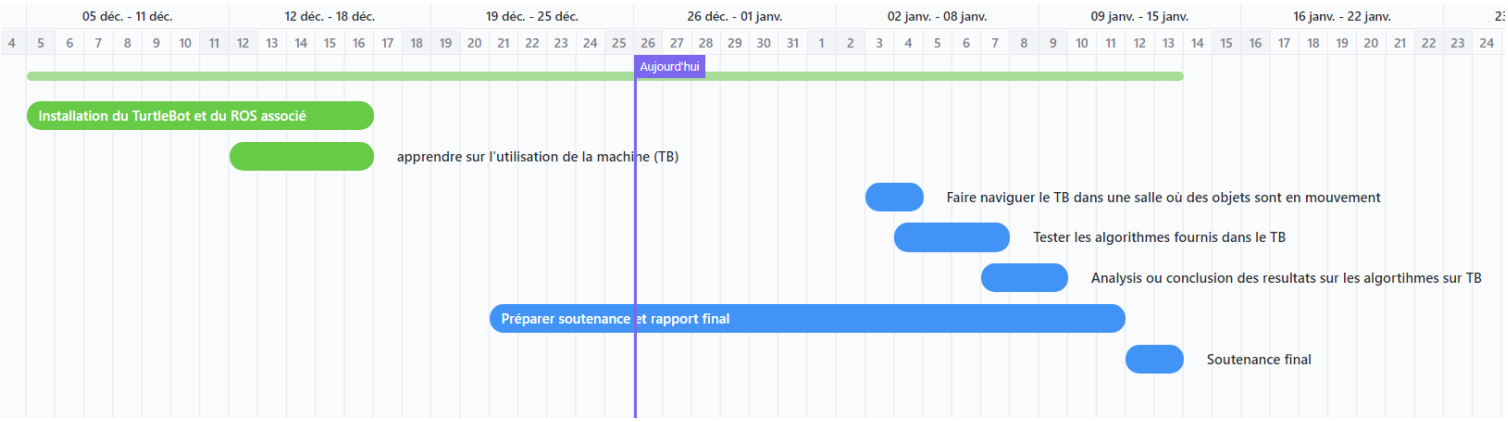

Figure 2 Planification adaptée

Pour pouvoir consulter les détails du diagramme de Gantt modifié et adapté vous pouvez cliquer sur le lien suivant :

- https://sharing.clickup.com/g/h/kg3vq-88/a17fe21bfc0b491

Les nouvelles taches sur lesquelles nous allons travailler sont alors les suivantes :

1. Installation de la plateforme TurtleBot et du ROS associé.
2. Apprendre sur l'utilisation de la machine (TB)
3. Faire naviguer le TB dans une salle où des objets sont en mouvement
4. Tester les algorithmes fournis dans le TB
5. Analyses ou conclusion des résultats sur les algorithmes sur le TB
6. Préparer soutenance et rapport final
7. Soutenance finale.

Cahier des charges en situation de fonctionnement :

Les fonctions identifiées sont proprement définies dans le CDCF. Les critères pour vérifier la réalisation des fonctions sont aussi définies. En plus, on estime les actions et les niveaux pour évaluer les critères, ainsi que leur flexibilité.

Problématique	Actions	Niveau	Flexibilité
FP1 : explorer l'environnement	L'algorithme doit permettre de parcourir efficacement l'environnement	Limiter les passages au même endroit et les croisements de la trajectoire avec elle-même	
FP2 : cartographier l'environnement	L'algorithme doit permettre de cartographier l'environnement	Le bruit de cartographie devra être limité grâce au couplage entre l'odométrie et le lidar	
FP3 : s'adapter aux variations de configuration de l'environnement	L'algorithme doit pouvoir faire contourner les obstacles	Le robot ne doit pas s'arrêter au moindre obstacle	
FP4 : garantir la sécurité des utilisateurs	L'algorithme doit s'adapter à la présence de murs ou de piétons	Le robot doit contourner les piétons et changer de direction s'il va vers un mur, voire s'arrêter si besoin.	Aucune

1. Contraintes applicables

Le travail dans un atelier doit être possible. La présence de personnes et de machines dans le champ de travail du robot ne doit pas l'empêcher d'avancer, il doit être en mesure de contourner la personne sans arrêt prolongé.

2. Cadre de réponse

On proposera des algorithmes fonctionnels, les parties commande et communication entre les sous-systèmes étant gérée par le turtlebot. Les sous-ensembles doivent pouvoir communiquer entre eux selon la même norme.

3 Les robots d'avant et d'aujourd'hui :

Un robot est une machine contrôlée par un ordinateur et programmable pour se déplacer, manipuler des objets et effectuer des travaux au même temps qu'elle interagi avec son environnement. Son utilité principale est de remplacer les personnes dans les tâches répétitives, difficiles, désagréables et dangereuses. Il permet de les effectuer d'une forme sécure rapide et précise.

Définitions du robot :

« Dispositif multifonctionnel reprogrammable conçu pour manipuler et/ou transporter des matériaux à l'aide des mouvements programmés pour effectuer une variété de tâches. »

« Machine ou dispositif électronique programmable, capable de manipuler des objets et d'effectuer des opérations préalablement réservées aux humains. »

Un robot doit respecter trois critères clés :

- Programmable, ce qui signifie disposer des capacités computationnelles et de manipulation des symboles (le robot est un ordinateur)
- Capacité mécanique, qui le capacite pour réaliser des actions dans son environnement et pas être un seul processeur des données (le robot est une machine).
- Flexibilité, puisque le robot peut fonctionner selon une vaste gamme des programmes et manipuler des matériaux des différents manières.

Avec tout ça, nous pouvons considérer un robot comme une machine complémentée par un ordinateur ou comme un ordinateur avec des dispositifs d'entrée et de sortie sophistiqués.

L'idée plus acceptée du robot est associée à l'existence d'un dispositif de contrôle digital qui, grâce à l'exécution d'un programme enregistré dans la mémoire, dirige les mouvements d'un bras ou d'un système mécanique.

Le concept des machines automatisées provient de l'antiquité. Les automates, ou machines apparaissent sur les horloges des églises médiévales ou sur les horloges du siècle XVIII qui étaient connus par leurs ingénieuses créatures mécaniques.

Le contrôle par rétroaction, le développement d'outils spécialisés et la division du travail en tâches plus petites pouvant être exécutées par des ouvriers ou des machines ont été les ingrédients essentiels de l'automatisation des usines au 18e siècle. Au fur et à mesure que la technologie s'est améliorée, des machines spécialisées ont été développées pour des tâches telles que la pose de bouchons sur des bouteilles ou le versement de caoutchouc liquide dans des moules à pneus.

Les machines les plus proches de ce que nous comprenons aujourd'hui comme des robots étaient les "téléopérateurs", utilisés dans l'industrie nucléaire pour la manipulation de substances radioactives. Il s'agissait essentiellement de servomécanismes qui, au moyen de systèmes mécaniques, répétaient les opérations effectuées simultanément par un opérateur.

L'inventeur américain George C. Devol a mis au point en 1954 un dispositif de transfert programmé articulé (selon sa propre définition) ; un bras primitif qui pouvait être programmé pour effectuer des tâches spécifiques.

En 1958, Devol s'associe à Joseph F. Engelberger et, dans le garage de ce dernier, ils construisent un robot qu'ils appellent Unimate. Il s'agissait d'un dispositif qui utilisait un ordinateur et un manipulateur pour former une "machine" à laquelle on pouvait "enseigner" à effectuer diverses tâches automatiquement.

Au cours des années 1960, un nouveau concept est apparu par rapport aux développements précédents. En vue d'une plus grande flexibilité, le retour sensoriel devient nécessaire. En 1962, Tomovic et Boni ont mis au point une main dotée d'un capteur de pression pour la détection des objets, qui fournissait un signal de retour au moteur.

Les premières applications industrielles en Europe, des applications de robots industriels dans les chaînes de production automobile, datent des années 1970 et 1971. En 1971, Kahn et Roth ont analysé le comportement dynamique et la commande d'un bras manipulateur.

Au cours des années 1970, la recherche en robotique s'est largement concentrée sur l'utilisation de capteurs externes pour les tâches de manipulation. C'est également au cours de ces années que la présence des robots sur les chaînes de montage et dans les installations industrielles du monde entier s'est définitivement consolidée.

En 1982, le robot Pedesco est utilisé pour nettoyer une fuite de combustible dans une centrale nucléaire. L'accent est également mis sur les domaines de la vision artificielle, de la détection tactile et des langages de programmation.

En 2000, le robot ASIMO a été présenté à la société, un robot humanoïde capable de se déplacer de manière bipède et d'interagir avec les gens, créé par Honda Motor Co. Ltd, qui, après plusieurs mises à jour en 2011, a réussi à fonctionner à une vitesse de 9 km/h, à faire la différence entre 3 personnes parlant en même temps et à accomplir des fonctions telles que prendre un gobelet en papier et le remplir d'eau sans en renverser une seule goutte.

Selon leur chronologie, nous pouvons faire la suivante classifications des robots :

1ère génération.

Manipulateurs. Il s'agit de systèmes mécaniques multifonctionnels dotés d'un système de commande simple, soit manuel, soit à séquence fixe, soit à séquence variable.

2ème génération.

Les robots d'apprentissage. Ils répètent une séquence de mouvements qui a été préalablement exécutée par un opérateur humain. Cela se fait au moyen d'un dispositif mécanique. L'opérateur effectue les mouvements requis tandis que le robot les suit et les mémorise.

3ème génération.

Robots à commande sensorisée. Le contrôleur est un ordinateur qui exécute les ordres d'un programme et les envoie au manipulateur pour qu'il effectue les mouvements nécessaires.

4ème génération.

Des robots intelligents. Ils sont semblables aux précédents, mais ils possèdent également des capteurs qui envoient des informations à l'ordinateur de contrôle sur l'état du processus. Cela permet une prise de décision intelligente et un contrôle en temps réel du processus.

4 ROS (Robot Operating System) :

ROS (Robot Operating System) est un kit de développement logiciel à source ouverte pour les applications robotiques. ROS offre une plateforme logicielle standard aux développeurs de tous les secteurs d'activité, qui leur permettra de passer de la recherche et du prototypage au déploiement et à la production. C'est-à-dire, C'est une plateforme de développement open source pour les systèmes robotiques. Il fournit une gamme de services et de bibliothèques qui simplifient considérablement la création d'applications robotiques complexes.

Elle est similaire à d'autres plateformes de développement pour robots existant actuellement, et sa plus grande vertu est d'avoir réussi à combiner le meilleur de chacun de ces systèmes, en réunissant le tout en un seul système capable de communiquer aussi bien avec les robots les plus modernes qu'avec ceux déjà présents sur le marché.

Depuis sa création, ROS a été conçu pour faciliter l'échange de logiciels entre les amateurs et les professionnels de la robotique du monde entier grâce à son approche didactique et ouverte, qui a permis la constitution d'une vaste communauté de collaborateurs à travers le monde.

En matière de développement, ROS permet l'utilisation de différents langages de programmation. Officiellement, Python, C++ et Lisp sont supportés, ainsi que beaucoup d'autres comme Java (encore en phase expérimentale mais supporté par Google), Lua, etc.

ROS peut être exécuté sur des machines de type Unix, principalement Ubuntu et Mac OS X, bien que l'on puisse trouver dans la communauté un support pour d'autres plateformes telles que Fedora, Gentoo, etc.

ROS dispose d'une énorme communauté de développement, composée de chercheurs, d'amateurs, de fabricants de matériel et de sociétés de soutien telles que Willow Garage, ou même Google. Ainsi ROS dispose déjà d'un support natif pour un grand nombre de matériels tels que Nao, Lego Mindstorms, les aspirateurs Roomba, ou encore les robots de recherche de plusieurs milliers de dollars comme le PR-2.

Tout cela, associé à la grande quantité d'exemples, aux bibliothèques prêtes à l'emploi (avec des algorithmes de navigation, de vision artificielle, etc.) et à la communauté ROS en pleine expansion, constitue un excellent point de départ pour se lancer dans le monde passionnant de la robotique.

4.1 Objectifs du ROS :

Le but de ROS n'est pas d'être un système pionnier dans le domaine de la robotique, son objectif principal est de soutenir le code réutilisable dans la recherche et le développement en robotique. Le système est un cadre distribué de processus (également appelés nœuds ou classes) qui permet de concevoir des exécutables individuels et de les utiliser de manière flexible en temps réel. Ces processus peuvent être regroupés en paquets et en piles, qui peuvent être facilement partagés et distribués. ROS prend également en charge un système de référentiel, qui permet une collaboration à un niveau international, autorisant des décisions indépendantes sur le développement et la mise en œuvre, mais avec l'avantage de pouvoir s'appuyer sur des outils ou des fichiers créés par d'autres dans leurs projets.

L'objectif principal de ce projet est de partager et de collaborer, mais il existe plusieurs autres objectifs dans le cadre de ROS :

- Simplicité : ROS est conçu pour être aussi simple que possible afin que le code écrit pour ROS puisse être utilisé avec d'autres cadres logiciels pour robots. ROS est facile à intégrer à d'autres systèmes d'exploitation pour robots et a déjà été intégré à OpenRAVE, Orocos et Player.
- Modèle de bibliothèque : il s'agit du modèle de développement préféré, qui consiste à écrire une série de programmes avec des interfaces propres et fonctionnelles permettant une modification rapide.
- Indépendance du langage : le cadre ROS est facile à mettre en œuvre dans tout langage de programmation moderne. Il a été implémenté en Python, C++ et LISP, et des bibliothèques expérimentales existent actuellement en Java et Lua.
- Mode de test : ROS possède une commande appelée ROSTEST qui permet d'accéder facilement à un mode de test du système.
- Échelle : ROS est adapté aux grands systèmes et à l'exécution de grands processus.

4.2 Philosophie de ROS :

La philosophie de ROS se résume dans les 5 grands principes suivants : Peer to Peer, Basé sur des outils, Multi langages, Léger, Gratuit et open source.

I. Peer to Peer :

Les systèmes ROS sont composés de nombreux petits programmes (nœuds) qui se connectent les uns aux autres et échangent continuellement des messages.

Un robot suffisamment complexe est composé de plusieurs ordinateurs ou cartes embarquées reliées par Ethernet ainsi que parfois des ordinateurs externes au robot pour des tâches de calcul intensif. Une architecture peer to peer couplée à un système de tampon (buffering) et un système de lookup (un nom service appelé master dans ROS), permet à chacun des acteurs de dialoguer en direct avec un autre acteur, de manière synchrone ou asynchrone en fonction des besoins.

II. Basé sur des outils :

Il existe de nombreux petits programmes génériques qui effectuent des tâches telles que la visualisation, la journalisation, le traçage des flux de données, etc.

ROS a adopté un design qui utilise un grand nombre de petits outils pour faire la construction et l'exécution des différents composants ROS. Lorsque vous parcourrez les tutoriaux ROS, vous apprenez à vous servir de plusieurs commandes permettant de manipuler les nœuds et les messages. Chaque commande est en fait un exécutable. L'avantage de cette solution est qu'un problème sur un exécutable n'affecte pas les autres, rendant le système plus robuste et évolutif qu'un système basé sur un runtime centralisé.

III. Multi langages :

Les modules logiciels ROS peuvent être écrits dans n'importe quel langage pour lequel une bibliothèque client a été écrite. Il existe actuellement des bibliothèques clientes pour C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, etc.

ROS est neutre d'un point de vue langage et peut être programmé en différents langages. La spécification de ROS intervient au niveau message. Les connexions peer to peer sont négociées en XML-RPC qui existe dans un grand nombre de langages.

Léger :

Les conventions ROS encouragent les contributeurs à créer des bibliothèques/packages autonomes, puis à envelopper ces bibliothèques afin qu'elles puissent envoyer et recevoir des messages vers/depuis d'autres modules ROS.

Afin de lutter contre le développement d'algorithmes plus ou moins liés avec l'OS robotique et donc difficilement réutilisables ensuite, les développeurs de ROS souhaitent que les pilotes et autres algorithmes soient contenus dans des exécutables indépendants. Cela assure la réutilisabilité maximale et surtout le maintien d'une taille réduite. Ce mécanisme rend ROS facile d'usage, la complexité se trouvant dans les librairies. Cette organisation facilite en plus le test unitaire. Enfin, ROS utilise du code (pilotes et algorithmes) issus d'autres projets open source : OpenCV (Librairie de traitement d'image et de vision artificielle), OpenRave (Algorithme de planification).

IV. Gratuit et open source :

Nous avons déjà expliqué les raisons de ce choix avant. Notons toutefois que l'architecture choisie est cohérente avec ce choix : ROS passe des données grâce à de la communication interprocessus. De ce fait, les modules n'ont pas besoin d'être liés dans un unique process, facilitant ainsi l'usage des différentes licences.

Il existe ensuite de nombreux systèmes embarqués spécifiques à un robot en particulier. On sort ici du modèle multiplateforme.

4.3 Systèmes d'exploitation compatibles.

Actuellement, ROS ne fonctionne pleinement que sur les plateformes basées sur UNIX. Le logiciel ROS est principalement testé sur Ubuntu et Mac OS X, bien que la communauté ROS ait contribué au support de Fedora, Gentoo, Arch Linux et d'autres plateformes Linux.

En ce qui concerne Microsoft Windows, il s'agit d'un système sur lequel nous pourrons installer notre système ROS, bien qu'il y ait encore quelques bugs mineurs en cours de correction.

ROS est compatible avec un simulateur de monde 3D propre comme **gazebo**. Avec gazebo, vous pouvez prendre le modèle que vous avez construit et le placer dans un monde simulé afin de pouvoir le conduire, manipuler la gravité, etc. Il permet un test virtuel du modèle construit pour vérifier son bon fonctionnement avant de construire physiquement le robot.

4.4 Nouvelles bibliothèques :

Le système, les outils et les bibliothèques du noyau ROS sont créés et mis à jour régulièrement dans le cadre d'une distribution ROS. Cette distribution est similaire à une distribution Linux et offre un ensemble de logiciels compatibles que chacun peut utiliser. Sa nature Open Source vous permet de modifier n'importe quelle distribution pour l'adapter à vos besoins.

ROS étant un système open source, tout le monde peut contribuer au système ainsi qu'aux bibliothèques qui lui sont compatibles, et il existe une section spécifique à cet effet sur le site web du logiciel.

4.5 Niveaux du système ROS :

ROS comporte trois niveaux de concepts : le niveau du système de fichiers, le niveau computationnel et le niveau de la communauté. Ces niveaux et concepts sont résumés ci-dessous :

a) Système de fichiers (Filesystem level) :

Il s'agit des ressources qu'on peut trouver dans programme lui-même :

- Les paquets : Ils sont l'unité principale pour organiser le software dans le ROS. Un paquet peut contenir des processus exécutables (nœuds), une bibliothèque dépend, des ensembles de données, des fichiers de configuration, ou tout autre élément utile à une organisation commune.
- Manifestes : Ils fournissent des métadonnées sur un paquet, notamment sa licence et des informations sur les dépendances, ainsi que des informations spécifiques au compilateur.
- Piles : Une collection de paquets qui ont la même fonction.
- Manifeste de pile : fournit des données sur une pile, notamment ses informations de licence et ses dépendances vis-à-vis d'autres piles.
- Messages : définit les structures de données pour les messages envoyés dans ROS.
- Services : définissent les structures de données des demandes et des réponses des services requis par ROS.

b) Niveau computationnel (Computation Level) :

Au niveau du graphe, c'est le réseau ROS qui est responsable du traitement de toutes les données. Les concepts de base sont les nœuds, le maître, les messages et les sujets, qui fournissent les données de différentes manières :

- Nœuds : Les nœuds sont des processus qui effectuent des calculs. ROS est conçu pour être modulaire à l'échelle de base. Un système de commande de robot comprend généralement de nombreux nœuds. Par exemple, un nœud contrôle un télémètre laser, un nœud contrôle les moteurs des roues, un nœud effectue la localisation, un nœud effectue la planification de la trajectoire, un nœud fournit une vue graphique du système, etc.
- Master : Le Maître fournit l'enregistrement et la recherche de noms pour le reste de l'informatique graphique. Sans le maître, les nœuds ne seraient pas en mesure de trouver les messages des autres, d'échanger ou d'invoquer des services.
- Messages : les nœuds communiquent entre eux en se transmettant des messages. Un message est simplement une structure de données comprenant des types de champs. Les messages peuvent inclure des structures et des tableaux arbitrairement imbriqués (tout comme les structures C).
- Topics : les messages sont acheminés par un système de transport sémantique de type publication/abonnement. Un nœud envoie un message en publiant sur un certain sujet. Le sujet est un nom qui est utilisé pour identifier le contenu du message. Un nœud qui s'intéresse à un certain type de données s'abonne au sujet correspondant. Il peut y avoir plusieurs éditeurs et abonnés simultanés au même sujet, et un seul nœud peut publier et/ou s'abonner à plusieurs sujets. En général, les éditeurs et les abonnés ne connaissent pas l'existence de l'autre. Vous pouvez considérer un sujet comme un bus de messages. Chaque bus a un nom, et n'importe qui peut s'y connecter pour envoyer ou recevoir des messages, à condition qu'ils soient du bon type.

c) La communauté ROS (Communication Level) :

Les concepts ROS communautaires ROS sont des ressources qui permettent aux communautés de partager des logiciels et des connaissances. C'est-à-dire, Ce communauté facilite l'échange de logiciels et de connaissances entre les membres de la communauté. Ces ressources comprennent :

- Distribution : Ce sont des collections de piles versionnées qui peuvent être installées. Les distributions jouent un rôle similaire à celui des distributions Linux : elles facilitent l'installation d'un ensemble de logiciels, et maintiennent également des versions cohérentes dans un ensemble de logiciels.
- Dépôts : ROS est basé sur un réseau fédéré de dépôts de code, où différentes institutions peuvent développer et publier leurs propres composants logiciels du robot.
- Le wiki ROS : la communauté Wiki est le principal forum pour la documentation des informations sur ROS. Tout le monde peut créer un compte et contribuer à sa propre documentation, apporter des corrections ou des mises à jour, écrire des tutoriels, etc.
- ROS Answers : un site de questions-réponses pour répondre aux questions relatives aux ROS.
- Blog : le blog de Willow Garage (créateur de ROS) propose des mises à jour régulières, notamment des photos et des vidéos.

4.6 Fonctionnement du système :

Après une brève explication des différents niveaux du ROS, on va passer maintenant à expliquer plus en détail comment fonctionne ROS, sa hiérarchie et son manière d'opérer.

4.6.1 Structure :

La première chose que nous allons examiner est la structure du système d'exploitation, du plus global (les dépôts) au plus spécifique (nœud).

- Les répertoires : Ils sont des fichiers composés par une ou plus paquets ou piles, et qui nous permettent télécharger des fichiers nécessaires d'une forme plus simple et confortable.
- Les piles (Stack) : Il s'agit d'une collection de paquets qui partagent la même fonctionnalité, ce serait l'équivalent d'une bibliothèque dans d'autres systèmes de programmation, et au sein de ceux-ci nous pouvons trouver des fichiers avec des métadonnées qui nous fournissent les informations pour qu'ils fonctionnent correctement (dépendances, compilation...).
- Les paquets : C'est l'unité principale d'organisation dans ROS, elle contient tout ce qui est nécessaire pour la rendre fonctionnelle et est analogue à un paquet C.
- Les nœuds : Ce sont les processus exécutables qui sont inclus dans les paquets. En général, plusieurs nœuds sont utilisés dans les programmes.

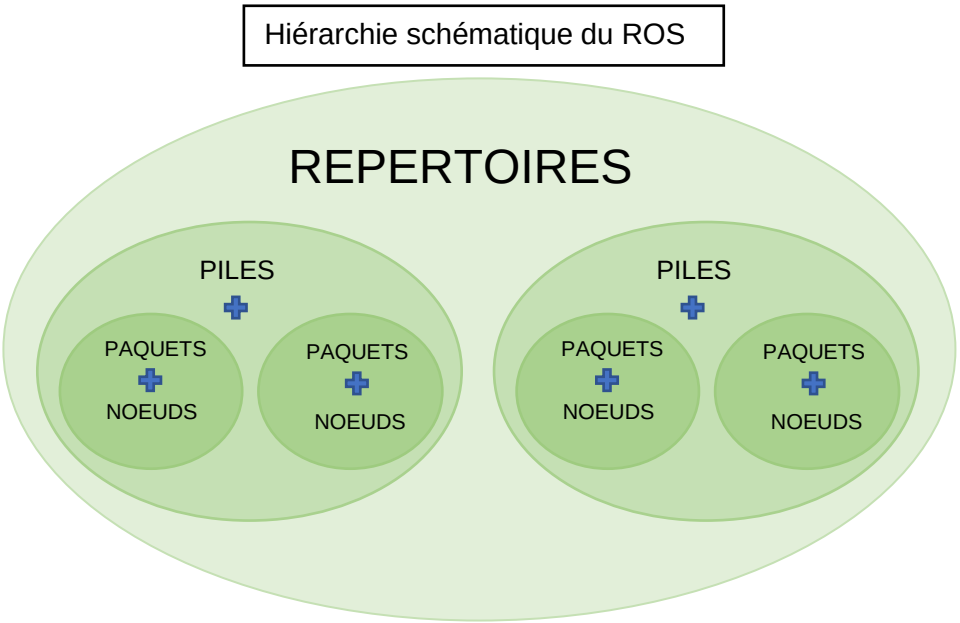

Figure 3 Hiérarchie schématique du ROS

4.6.2 Computation dans ROS :

Services :

Il s'agit du type d'architecture responsable de la communication entre les nœuds, qui utilise deux types de messages, l'un pour la demande, l'autre pour la réponse donnée par l'autre nœud à cette demande.

Dans les programmes, nous pouvons trouver des nœuds serveurs et des nœuds clients.

Après avoir fait la demande ou la requête, le nœud serveur reste en mode d'attente jusqu'à ce qu'il reçoive une réponse du nœud client. Si c'est le client qui fait la demande, le nœud serveur la traite et répond au client avec les informations demandées.

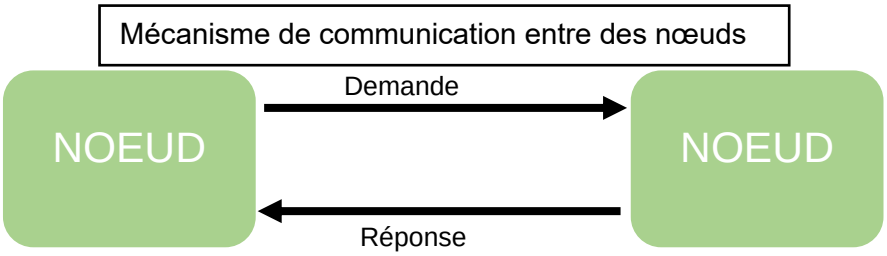

Figure 4 Mécanisme de communication entre des nœuds

Les Topics :

Les Topics ou thèmes sont les noms qui identifient le contenu d'un message, et ceux-ci sont acheminés de deux manières, un éditeur et un abonné.

Un nœud qui s'intéresse à un certain type de données s'abonne au sujet correspondant.

Il peut y avoir plusieurs éditeurs et abonnés simultanés au même topic, et un seul nœud peut publier et/ou s'abonner à plusieurs topics. En général, les éditeurs et les abonnés ne connaissent pas l'existence de l'autre.

Vous pouvez considérer un sujet comme un bus de messages. Chaque bus a un nom, et n'importe qui peut s'y connecter pour envoyer ou recevoir des messages, à condition qu'ils soient du bon type.

Les structures de données des requêtes et des réponses :

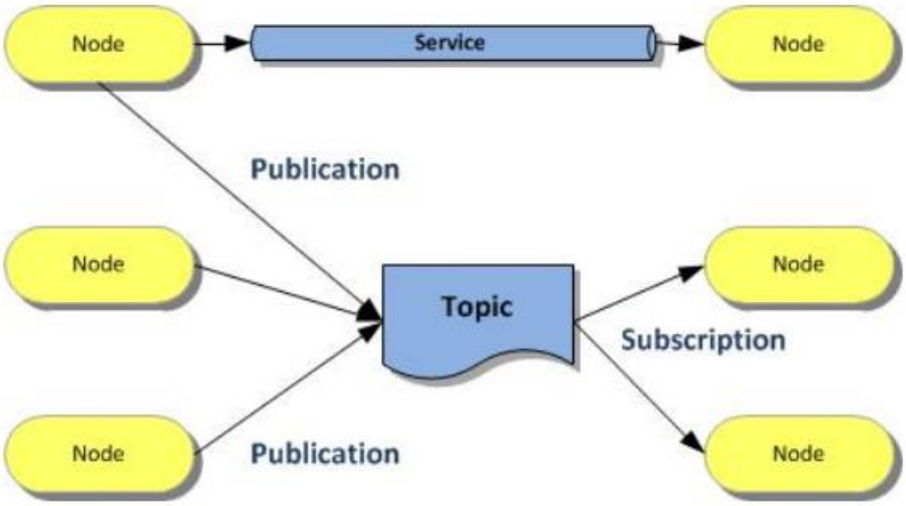

Figure 5 Structure de données des requêtes et des réponses

Messages :

Les nœuds communiquent entre eux en se transmettant des messages. Un message est simplement une structure de données, comprenant des types de champs. Les messages peuvent inclure des structures et des tableaux arbitrairement imbriqués (tout comme les structures C).

Le Master :

Le Master fournit l'enregistrement et la recherche de noms pour le reste des nœuds. Sans le maître, ils ne seraient pas en mesure de trouver les messages de l'autre, d'échanger ou d'invoquer des services, ce qui le rend totalement indispensable lors de l'exécution de tout type de programme.

Bags :

Les sacs sont un format de stockage et de relecture des données de messages ROS, qui nous permet de stocker une série de commandes et de les répéter séquentiellement. Les sacs constituent un mécanisme important pour le stockage de données, telles que les données de

capteurs, qui peuvent être difficiles à collecter, mais qui sont nécessaires pour développer et tester des algorithmes.

5 SLAM (Simultaneous Locating And Mapping) :

Le SLAM (Simultaneous Localisation And Mapping) est un processus par lequel un robot peut construire une carte d'un environnement donné et en même temps naviguer dans cet environnement. Pour ce faire, il faut identifier ou connaître à l'avance un certain nombre de points caractéristiques qui seront utilisés comme points de référence par le système.

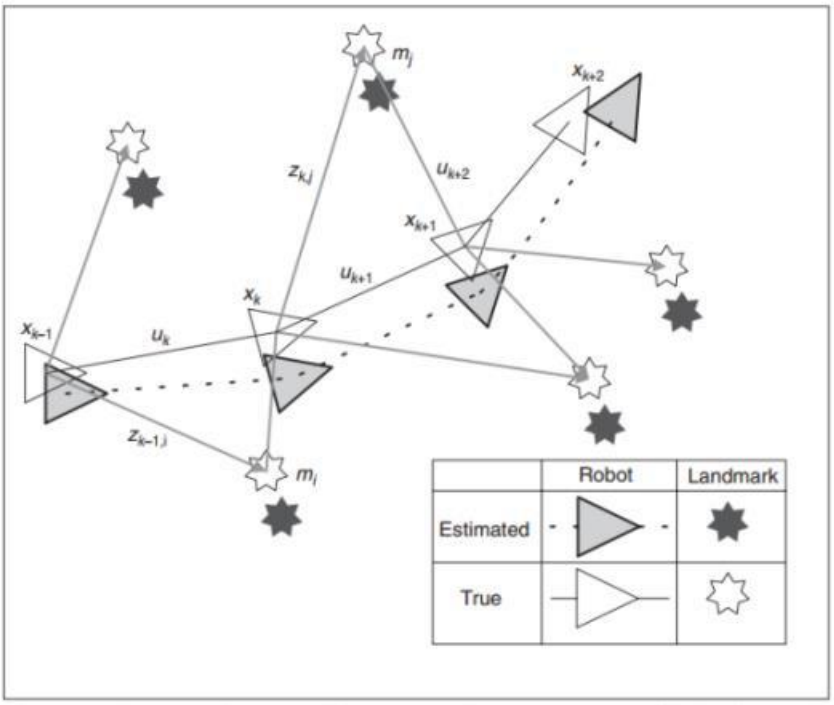

Figure 6 Fonctionnement basic du SLAM

La figure 1 montre rapidement le comportement réel d'un robot avec une certaine imprécision dans ses capteurs, ce qui génère une incertitude dans la position observée des points caractéristiques et donc dans l'idée que le robot se fait de sa position. Même si les capteurs étaient idéaux, cette incertitude existe toujours car la position de départ du robot n'est pas connue. Le fait de ne pas connaître cette position de départ est précisément ce qui distingue le SLAM des autres stratégies de localisation, car le fait de ne pas avoir besoin d'un protocole d'initialisation est à la fois un grand avantage et une difficulté supplémentaire.

Une autre des qualités les plus importantes du SLAM est que les estimations de la position des minuties s'améliorent au fur et à mesure que les minuties sont réobservées, ce qui rend l'algorithme plus efficace et plus fiable dans le temps.

Le SLAM sera appelé "le système" et non "l'algorithme", car le processus se compose de plusieurs étapes, chacune ayant ses propres algorithmes. La solution au problème du SLAM n'est donc pas unique, et de nombreuses approches différentes peuvent être adoptées pour trouver une solution, comme nous le verrons plus loin.

Afin de créer les cartes, l'algorithme a besoin d'informations provenant des capteurs appropriés. Au minimum, il est nécessaire de disposer de données sur la profondeur et la commande de mouvement effectuée sur le robot. L'odométrie a été pendant de nombreuses années la méthode la plus courante parce que la moins chère, basée sur la lecture des révolutions des roues du robot et l'utilisation de capteurs inertiels pour estimer les virages. Aujourd'hui, cependant, le SLAM est un problème souvent résolu par la vision par ordinateur.

5.1 L’évolution du problème de SLAM :

La nécessité de résoudre ce problème découle du désir d'une plus grande autonomie des robots mobiles. Une fois qu'ils peuvent se déplacer de manière autonome dans un environnement donné, on peut compter sur eux pour des tâches de plus en plus complexes.

À partir de la décade des 90, les mondes de la robotique et de l'intelligence artificielle commençaient à fusionner, grâce à la première application des méthodes d'estimation probabiliste aux problèmes de robotique. Le plus grand risque lié à l'hypothèse d'une incertitude élevée est non seulement d'obtenir une fausse trajectoire, mais aussi de confondre des détails, ce qui peut conduire à des inexactitudes fatales.

Au cours des années suivantes, des études clés ont été publiées dans ce domaine, qui ont atteint l'objectif de manipuler l'incertitude géométrique de manière statistique, ce qui signifie que l'incertitude est réduite au fil des scans successifs. La solution la plus fréquemment adoptée est basée sur la modélisation avec le filtre de Kalman étendu (EKF).

La dernière décennie a vu une énorme croissance des performances des solutions SLAM qui ont été développées. Alors que la plupart des propositions visaient à améliorer l'efficacité des calculs en essayant de maintenir la précision de la cartographie, beaucoup d'autres ont trouvé la solution dans l'amélioration de la source de données. L'incorporation de l'odométrie visuelle dans le système, c'est-à-dire l'obtention de la variation de la position du point de vue à partir d'images successives, est particulièrement importante.

En général, une bonne mise en œuvre de l'odométrie visuelle est plus fiable que l'odométrie classique. Ce type de capteur a été mis en œuvre avec succès pour la première fois en 2003. Il est aujourd'hui le plus utilisé dans la recherche grâce à la réduction du prix des caméras et à l'apparition de capteurs avec lecture intégrée des couleurs et de la distance à un coût raisonnable.

D'un point de vue conceptuel et théorique, le SLAM est un problème qui peut être considéré comme résolu. Cependant, il existe encore des domaines qui peuvent être largement améliorés, comme la construction de la carte et son utilisation, ou le manque de polyvalence et d'adaptabilité dans le choix des points caractéristiques. Ce dernier point est particulièrement important, car il existe plusieurs critères pour les localiser. Logiquement, un robot entraîné à détecter des points de repère basés sur la couleur ne pourra pas s'orienter correctement dans une pièce monochromatique. Un autre problème pour lequel il n'existe pas de solution stable, même aujourd'hui, est la reconnaissance des changements de l'environnement.

5.2 Classification des différents types de SLAM :

Il existe une grande variété de caractéristiques permettant de classer les systèmes SLAM. Les critères que nous avons choisis pour classer les systèmes SLAM sont basés sur les types de sous-systèmes qui composent un système SLAM complet, la disponibilité des capteurs ou les fonctionnalités qu'il offre.

2D ou 3D :

Les systèmes de cartographie 2D SLAM ne sont utiles que dans les environnements plats et basent leurs observations sur des capteurs de distance ou de profondeur. L'algorithme le plus populaire pour résoudre ce problème est connu sous le nom de Grid Mapping (cartographie à grille ou quadrillé), qui divise la zone de travail en une grille de la résolution souhaitée, où chaque cellule a une probabilité de 0 à 1 d'être occupée. En combinant l'observation et les méthodes probabilistes, chacune de ces cellules est déchiffrée.

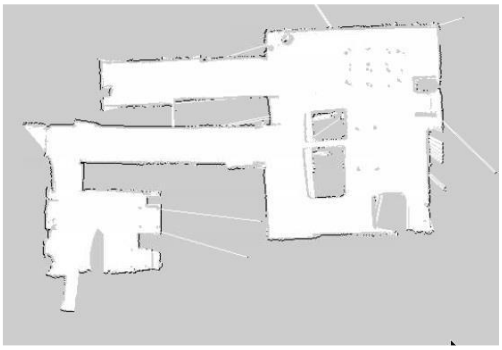

Figure 7 Exemple de SLAM 2D

Basé sur les points caractéristiques ou points directs :

Les points caractéristiques sont des références que le système utilisera pour se localiser. Il peut s'agir de données manuelles, de balises détectées par un certain capteur ou créées par le système. Ce dernier cas est le plus délicat et le moins robuste, car ces points crées sont déjà associés à une incertitude. Comme nous le verrons plus loin, il existe plusieurs critères visuels pour choisir les points caractéristiques, tels que la détection des coins, les formes ou les couleurs.

Selon le type de camera utilisée :

En SLAM 3D, nous aurons toujours une image en entrée, presque toujours en couleur. À partir de ces informations, il est encore possible d'aborder le problème de différentes manières.

Le SLAM monoculaire est connu comme un SLAM que grâce à la simplicité et le prix de son matériel le rendent idéal pour les applications où la robustesse n'est pas une priorité, même si la complexité de sa mise en œuvre sera beaucoup plus grande. Traditionnellement, il était essentiel d'utiliser un filtre de Kalman étendu pour suivre les minuties extraites, comme pour les prédictions de profondeur d'un système 2D. Cependant, l'utilisation de plusieurs images consécutives pour trianguler un point et prédire sa profondeur devient populaire. L'inconvénient majeur est l'erreur qui s'accumule dans leurs estimations, ainsi que le fait que plusieurs images consécutives sont traitées et fournissent peu de nouvelles informations.

À l'extrême opposé, on trouve l'utilisation de caméras stéréo. De la même manière que les yeux humains font la mise au point, ils sont capables de calculer la distance des objets à partir

des images. C'est pourquoi ils permettent des applications commerciales où il y a un intérêt à remplacer les humains, comme les voitures autonomes ou les drones.

Une dernière combinaison de capteurs, moins courante, consiste à utiliser une seule caméra assistée par un capteur inertiel. De cette manière, une estimation des tours est obtenue et l'incertitude des minuties est réduite de manière drastique.

Online/Offline :

Le SLAM Online, est celui qui fonctionne en temps réel, la pose du point de vue est principalement calculée à chaque instant. Le calcul de la trajectoire n'est pas quelque chose d'intéressant à la volée, car le but est de déduire comment la pose de la caméra évolue.

Cependant, dans les applications où l'objectif principal est d'enregistrer une séquence et de l'analyser ensuite pour voir si elle est optimale (hors ligne), l'objectif principal est généralement d'obtenir la trajectoire du robot comme produit final. Le grand avantage de la conception d'un système hors ligne est la liberté de mettre en œuvre des ressources qui le rendent plus puissant, puisqu'un temps d'exécution plus long n'est pas un problème.

5.3 Applications habituelles et potentielles :

Bien que depuis sa création, le SLAM ait été largement utilisé dans les robots mobiles pour accélérer et automatiser les tâches industrielles, au cours de la dernière décennie, les domaines d'application de cette technologie se sont multipliés grâce à l'évolution du matériel et à l'apparition sur le marché de nouveaux besoins, tant industriels que domestiques.

- **Cartographie 2D :** La variante la plus simple du SLAM, qui utilise le mappage 2D, est suffisante pour les applications de pièces plates où l'information de distance est la seule source nécessaire. Il s'agit d'une application relativement primitive qui est utile dans les robots domestiques tels que les aspirateurs autonomes ou dans les chaînes de montage où des chariots automatisés sont utilisés pour transporter des pièces.
- **La cartographie comme alternative aux autres méthodes de localisation :** Le potentiel de la SLAM dans les grands espaces est considérable si les bons capteurs sont en place. Il est donc possible d'automatiser des tâches pour lesquelles le GPS n'est pas une option, comme dans l'exploration des fonds marins, les mines, les tunnels et même sur d'autres planètes. Sa mise en œuvre dans des robots et des drones pour de telles tâches est déjà une réalité.
  D'autre part, et de manière similaire, il existe une variété d'expériences dans lesquelles le SLAM est appliqué à la voiture autonome. Les capteurs utilisés ne ressemblent guère à ceux que nous utiliserons dans des applications de recherche dans des espaces confinés, nécessitant plusieurs caméras, un LIDAR ou des capteurs similaires et le support d'un GPS et d'une odométrie.
- **Réalité augmentée :** les informations de profondeur sont utilisées pour localiser les éléments numériques, complétées par un algorithme SLAM pour garder le point de vue cohérent avec ces éléments.
  Ce type d'application commence à être utilisé par les entreprises des secteurs de la technologie médicale, de l'aérospatiale et de l'automobile pour former leurs travailleurs dans des scénarios semi-virtuels, car il permet de réaliser d'importantes économies de coûts et, surtout, de risques.
- **Applications en médecine :** Il existe de nombreuses procédures chirurgicales qui peuvent bénéficier du SLAM. En particulier celles dans lesquelles une

caméra examine l'intérieur d'une cavité. Dans ces cas, l'inclusion du SLAM permet de cartographier les cavités à examiner et de les étudier ultérieurement, réduisant ainsi le temps nécessaire à l'examen du patient.

- **Scannage d'objets en 3D :** Cela a plusieurs applications possibles dans le monde de la réalité augmentée, car cela permet d'introduire dans l'univers virtuel des objets préalablement scannés par l'utilisateur, par exemple en mettant le visage de l'utilisateur sur un personnage de jeu vidéo. Une technique similaire a été utilisée pour inclure des bâtiments réels dans des cartes virtuelles telles que Google Maps ou dans des effets spéciaux vidéo.

5.4 Extraction des points caractéristiques :

Dans les systèmes SLAM basés sur les points caractéristiques, il est courant que les points doivent être identifiées à la volée. Cela sera compliqué par une complexité croissante, d'où l'importance de reconnaître ces points une fois qu'ils ont été créés.

Par conséquent, les approches adoptées pour résoudre ce problème comprennent toujours une approximation calculée à l'aide de détecteurs et de descripteurs. Un détecteur est simplement un algorithme qui détecte des formes, des textures ou des couleurs dans une image, en renvoyant des coordonnées 2D. Idéalement, un détecteur est complété par un descripteur, qui analyse également la zone autour du point détecté afin d'optimiser l'échantillonnage ou de reconnaître l'orientation du point.

Une autre façon d'aborder le problème consiste à détecter des arêtes au lieu de points. Les algorithmes de Canny, Sobel, Harris ou Prewitt appliquent de manière similaire une réduction gaussienne, en recherchant la direction du gradient de chaque pixel et en filtrant ainsi les zones non texturées.

5.5 Calcul des trajectoires :

Scan Matching :

La façon la plus populaire de calculer ces trajectoires est en utilisant Scan Matching qui consiste à trouver une transformation rigide de translation et de rotation qui aligne deux images successives.

Afin d'analyser ces images, les points caractéristiques visibles dans les deux images sont pris comme données d'entrée. Soit donc deux ensembles de points :

$$X = \{x_1, ..., x_n\}$$
$$P = \{p_1, ..., p_n\}$$

Figure 8 Ensemble de points

La procédure est d'appliquer les minimes carrés pour trouver la ligne reliant les cadres. Dans ce cas, nous voulons obtenir la rotation R et la translation t telles que l'expression des moindres carrés soit minimisée :

$$E(R,t) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||x_i - Rp_i - t||^2$$

Figure 9 Expression de Reduction par des minimes carrés

Où xi et pi sont le même point dans les deux images.

Cette méthode est simple à mettre en œuvre mais est très sujette aux erreurs de précision, car elle prend en compte des points qui peuvent ne pas être pertinents pour la transformation. Cela peut influencer radicalement le résultat si trop de fausses références sont détectées.

Pour résoudre ce problème, il existe plusieurs façons de filtrer les points que l'on ne veut pas prendre en compte. Le plus populaire est l'algorithme RANSAC (RANdom SAmple Consensus).

RANSAC est une méthode itérative pour estimer les paramètres d'un modèle mathématique d'un ensemble de données observées contenant des valeurs aberrantes. Il s'agit d'un algorithme non déterministe dans le sens où il ne produit un résultat raisonnable qu'avec une certaine probabilité, qui augmente avec le nombre d'itérations autorisées.

Filtro de Kalman :

Le filtre de Kalman est un outil probabiliste. Son but dans le SLAM est, étant donné une observation et une action de mouvement, de fournir la nouvelle position des points de référence et la trajectoire qui a été suivie. En d'autres termes, si nous définissons l'état du robot comme sa position par rapport aux points caractéristiques, nous obtiendrons les états successifs.

Pour ce faire, une stratégie de prédiction plus correction est suivie. C'est-à-dire que lorsque nous observons un point caractéristique, nous voulons être sûrs de ne pas le confondre avec un autre, donc une prédiction est faite et c'est cette incertitude qui est corrigée.

En résumé, l'algorithme suit 5 étapes :

- Prévision de l'État
- Prédiction des mesures
- Observer la mesure
- Associer les données
- Mettre à jour la prédiction avec l'observation.

1: $\mathbf{Kalman\_filter}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: $\quad \bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$

3: $\quad \bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$

4: $\quad K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$

5: $\quad \mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$

6: $\quad \Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$

7: $\quad return\ \mu_t, \Sigma_t$

Figure 10 Algorithme du filtre de Kalman

Dans l'algorithme de la figure 20, μt est l'état et zt l'observation. On voit apparaître la matrice de covariance Σ, qui joue un rôle fondamental puisqu'elle comporte des informations sur l'incertitude. Kt est un paramètre qui prend en compte les incertitudes et les observations pour quantifier le degré de certitude du robot quant à ses conclusions. Plus le K est élevé, plus l'incertitude est faible.

6 LiDAR :

C'est l'acronyme de "Light Detection and Ranging", c'est-à-dire la détection par la lumière et la distance. Il s'agit d'un système laser qui nous permet de mesurer la distance entre le point d'émission de ce laser et un objet ou une surface. Le temps que met le laser pour atteindre et revenir de sa cible est ce qui nous permet de calculer la distance entre les deux points. L'un des avantages de ce capteur par rapport à d'autres qui peuvent remplir une fonction similaire car ils ont la capacité de recevoir des mesures simultanées dans plusieurs directions en même temps avec un taux d'échantillonnage élevé. L'utilisation de cette technologie est de plus en plus mise en œuvre dans les voitures autonomes.

La réception étant basée sur le laser, elle est beaucoup plus rapide et moins sujette au bruit, ce qui en fait une option beaucoup plus précise que le sonar.

Toutes les données sont transmises par USB au Raspberry Pi. Le type de sortie des données est un tableau qui comprend les valeurs de distance détectées autour du LiDAR.

7 Raspberry Pi :

La Fondation Raspberry Pi elle-même le définit comme un PC à faible coût, de la taille d'une carte de crédit, qui se connecte à un écran ou à un téléviseur et s'utilise avec un clavier et une souris. Quel que soit ton âge, il te permet d'explorer le monde de l'informatique et d'apprendre les langages de programmation Scratch et Python.

Le Raspberry Pi est un petit ordinateur peu coûteux qui tient dans la paume de votre main, mais vous pouvez y connecter un téléviseur et un clavier pour interagir avec lui comme n'importe quel autre ordinateur.

On peut attendre d'un Raspberry Pi la même utilisation qu'un PC : naviguer, utiliser Microsoft Office, lire des vidéos, jouer à des jeux vidéo (légers), etc. Ce sont des appareils qui ont connu un grand succès grâce à leur faible coût et à leur grande accessibilité, ainsi qu'à la communauté qui se cache derrière le monde des Raspberry Pi.

Le Raspberry Pi est une simple carte informatique composée d'un SoC, d'un CPU, de RAM, de ports d'entrée et de sortie audio et vidéo, d'une connectivité réseau, d'un slot SD pour le stockage, d'une horloge, d'une prise d'alimentation, de connexions pour les périphériques de bas niveau, de... c'est à peu près la même chose que si vous regardez l'arrière d'une tour d'ordinateur, car le Raspberry est un ordinateur. Il n'y a pas d'interrupteur pour l'allumer ou l'éteindre, cependant.

Pour le rendre opérationnel, nous devons connecter des périphériques d'entrée et de sortie pour pouvoir interagir avec lui, comme un écran, une souris et un clavier, et enregistrer un système d'exploitation pour Raspberry Pi sur la carte SD. Il ne reste plus qu'à le connecter à l'alimentation électrique et nous sommes prêts à partir.

De quoi a-t-elle besoin pour fonctionner ?

La Fondation Raspberry elle-même explique très facilement en cinq étapes ce dont un Raspberry Pi a besoin pour fonctionner :

1. Insérez une carte SD comme mémoire de stockage avec un système d'exploitation
2. Connexion de la carte mère à un moniteur ou à un téléviseur

3. Connecter un clavier et une souris
4. Connectez-vous au réseau via Ethernet (ou WiFi - ceci est optionnel).
5. Connectez le Raspberry Pi à l'alimentation électrique.

8 Les Méthodes de SLAM :

8.1 Gmapping :

L'un des algorithmes les plus connus pour la mise en œuvre du SLAM est connu sous le nom de Gmapping. Ce logiciel met en œuvre une adaptation de la bibliothèque OpenSLAM pour la rendre compatible avec l'environnement ROS.

Le programme reçoit le système de coordonnées associé à la base du robot et le système de coordonnées associé à l'odométrie du robot. Suite à l'application du filtre à particules, un nouveau système de coordonnées lié à la carte sera généré.

Il sera également nécessaire de fournir au nœud les informations recueillies par le capteur. Cela se fait en indiquant le sujet contenant les messages de type sensor_msgs::LaserScan.

Concernant la carte, la position de l'odométrie indiquera la position initiale où le robot est apparu et l'image de la base sera l'estimation de la position du robot. Le système de coordonnées odométriques est généré par le robot lui-même à partir des commandes de vitesse qui lui sont envoyées. En l'absence de carte, sa relation avec la base sert à estimer sa position. En résumé, on peut inclure que, conformément à la théorie, le système de coordonnées de la carte est une correction d'image purement odométrique dans l'estimation de la pose du robot.

Gmapping est l'algorithme SLAM le plus intégré qui puisse être utilisé dans ROS aujourd'hui. Il utilise un nœud appelé slam_gmapping pour créer une carte d'occupation 2D au format grille, grâce aux informations du laser et à la position accumulée du robot mobile.

Les algorithmes SLAM intègrent souvent des filtres d'optimisation et le cas du Gmapping ne fait pas exception. Cette méthode utilise le filtre à particules RaoBlackwellisé.

Rao-Blackwellised suppose que, étant donné la trajectoire de l'observateur, chaque particule peut porter une carte individuelle de l'environnement. Le nombre de particules est réduit par calcul grâce à une distribution probabiliste qui prend en compte le mouvement du robot et les observations les plus récentes fournies par les capteurs. Par conséquent, l'incertitude sur la position du robot est considérablement réduite.

Gmapping utilise l'algorithme Split-and-merge, une technique de traitement d'images. La procédure est illustrée dans l'image ci-dessous.

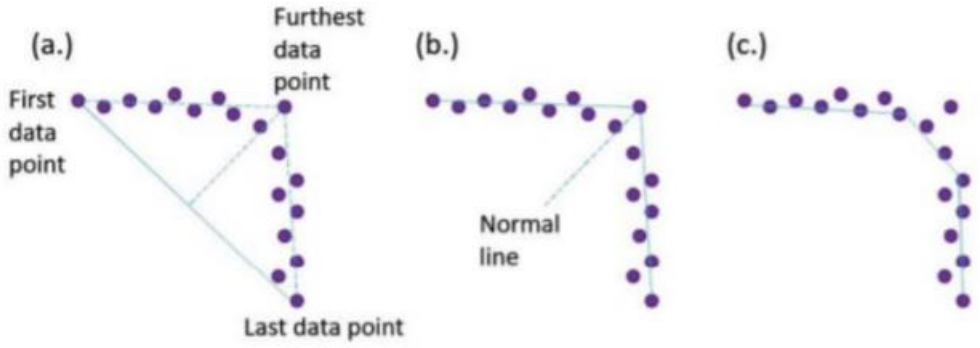

Figure 11 Split-and-merge algorithme de Gmapping

Tout d'abord, une ligne d'estimation reliant le point de départ au point d'arrivée est établie. Ensuite, le point le plus éloigné est joint à la ligne reliant le point de départ et le point d'arrivée, si la longueur de la ligne est supérieure à un seuil prédéfini, la ligne est divisée en deux, reliant le premier et le dernier point au point le plus éloigné. Cette procédure est répétée jusqu'à ce que la longueur de la ligne normale soit inférieure au seuil donné.

- Avantage du Gmapping:
  - Documentation suffisante sur les sites Web des développeurs.
  - Le paquetage est facile à configurer et à utiliser.
  - Odométrie suffisamment précise.
- Desavantages :
  L'utilisation de ce package est limitée uniquement aux cas où le robot dispose de données d'odométrie. Comme nous pouvons seulement générer une carte en 2D ceci va supposer un problème pour la mise en œuvre ultérieure dans le véhicule autonome, qui vise à utiliser les informations Lidar en 3D pour saisir davantage d'informations sur un environnement complexe.

8.2 Hector SLAM :

L'algorithme Hector de localisation et de cartographie simultanées Hector est un système SLAM 3D, qui combine un balayage lidar 2D avec des informations provenant d'un IMU à l'aide d'un filtre de Kalman (KF).

Il est composé de trois modules différents :

- hector_mapping : le nœud SLAM.
- hector_geotiff : où sont stockées la carte et la trajectoire du robot.
- hector_trajectory_server : stocke les transformées de Fourier (FFT) des trajectoires.

Le système n'est pas conçu pour nécessiter des informations d'odométrie, il s'appuie donc uniquement sur les données fournies par le capteur laser et un IMU. En combinant la position avec l'unité de tangage/roulement, le laser est capable de se stabiliser, ce qui rend le système capable de travailler dans des environnements qui ne sont pas nécessairement plats.
L'un des nœuds du paquet fusionne le filtre de Kalman étendu (EKF) avec les informations de l'IMU et l'erreur de position du laser 2D. Le filtre est basé sur un modèle de mouvement générique pour les véhicules terrestres et est amélioré par l'IMU, sans avoir à se fier aux

données d'odométrie qui peuvent normalement être erronées en raison des irrégularités du terrain, ou inexistantes comme dans le cas des véhicules aériens.

Avec le nuage de points formé par le laser, une carte d'élévation de type grille 2D est générée qui stocke la valeur de hauteur correspondant à chaque cellule. Enfin, un autre nœud fusionne la carte d'élévation 2,5 D avec la carte d'occupation de la grille 2D.

Les positions par lesquelles le robot est passé sont stockées dans un des nœuds et échantillonnées en fonction de leur distance les unes par rapport aux autres en les ajoutant à une liste. Un algorithme de transformation appliqué à cette liste, ainsi que la recherche de la cellule ayant la valeur la plus élevée, génère un point suffisamment éloigné et sûr pour atteindre le robot.

Désavantages :
Hector SLAM est un paquet très complet qui offre de multiples options de travail, mais il n'a pas été décidé de le faire passer à la phase de mise en œuvre. Cette méthode, bien qu'elle obtienne dans ce cas une carte 3D, le fait en utilisant un lidar 2D combiné à un IMU. Pour ce projet, nous recherchons un algorithme qui utilise le lidar 3D, qui contient beaucoup plus d'informations sur l'environnement.

8.3 Karto :

C'est l'algorithme commercial de SLAM basé sur le graphique de SRI International, disponible en open-source sur les dépôts ROS. Il n'y a pas beaucoup d'information ou de documentation disponible à son sujet sur ROS ou sur le site web de Karto. Nous avons constaté que chaque nœud du graphe conserve les informations de pose ainsi que les mesures des capteurs. Normalement, avec un grand nombre de points de repère, plus de mémoire est nécessaire. Mais comme il s'agit d'un SLAM basé sur un graphe qui ne conserve que les informations de pose par nœud, il est très économe en mémoire.

Ce paquet utilise la méthode propre aux auteurs appelée Sparse Pose Adjustment (SPA). Elle est basée sur l'optimisation de graphes de localisation, ces derniers étant compris comme l'ensemble des localisations d'un robot, reliées par des contraintes non linéaires, obtenues à partir des observations faites par les capteurs. Pour l'optimisation, le problème est supposé être un problème de moindres carrés non linéaires et la méthode Levenberg-Marquardt est utilisée pour le résoudre.

Cette optimisation rend la fermeture de boucle avec cette méthode très efficace, et de plus, comme elle provient d'une méthode graphique, les résultats sur les grandes cartes sont assez bons. Cependant, comme pour la plupart des méthodes graphiques SLAM, le coût de calcul est élevé.

8.4 Cartographer SLAM :

C'est un paquet open source de Google. Il s'agit d'un système qui permet la localisation et la cartographie simultanées en temps réel dans des environnements extérieurs et intérieurs. Cartographer offre la possibilité d'obtenir deux types de cartes (2D ou 3D) en fonction du capteur laser utilisée.

C'est un système qui fournit une localisation et une cartographie simultanées en temps réel (SLAM) en 2D et 3 D sur plusieurs plateformes et configurations de capteurs. Ce projet fournit l'intégration ROS de Cartographer. Le système comporte de nombreux paramètres, dont beaucoup s'influencent mutuellement.

Cartographer peut être considéré comme deux sous-systèmes distincts, mais liés. Le premier est le SLAM local (parfois aussi appelé frontend ou local trajectory builder). Son rôle est de construire une succession de sous-cartes.
L'autre sous-système est le SLAM global (parfois appelé le backend). Il fonctionne en arrière-plan et sa tâche principale est de trouver les contraintes de fermeture de boucle. Pour ce faire, il fait correspondre des scans (rassemblés dans les nœuds) à des sous-cartes. Il intègre également d'autres données de capteurs pour obtenir une vue de plus haut niveau et identifier la solution globale la plus cohérente. En 3D, il essaie également de trouver la direction de la gravité.

Local SLAM insère un nouveau balayage dans la construction de sa sous-carte actuelle par correspondance de balayage en utilisant une estimation initiale de le “pose extractor”. L'idée derrière le “pose extractor” est d'utiliser les données de capteurs autres que le télémètre pour prédire où le prochain scan doit être inséré dans la sous-carte.

Deux stratégies de correspondance de scan sont disponibles :

- Le CeresScanMatcher prend la supposition initiale comme antécédent et trouve le meilleur endroit où la correspondance de balayage s'ajuste la sous-carte. Il le fait en interpolant la sous-carte et en alignant le scan au sous-pixel. Cette méthode est rapide, mais ne peut pas corriger les erreurs qui sont beaucoup plus grandes que la résolution des sous-cartes. Si la configuration de votre capteur et le timing sont raisonnables, utiliser uniquement le CeresScanMatcher est généralement le meilleur choix à faire.

- Le RealTimeCorrelativeScanMatcher peut être activé si vous n'avez pas d'autres capteurs ou si vous ne leur faites pas confiance.Il utilise une approche similaire à la manière dont les balayages sont comparés aux sous-cartes dans la fermeture de boucle (décrite plus loin), mais il les compare à la sous-carte actuelle. La meilleure correspondance est alors utilisée comme antécédent pour le CeresScanMatcher. Ce matcheur de scan est très coûteux et va essentiellement écraser tout signal provenant d'autres capteurs que le télémètre, mais il est robuste dans les environnements riches en fonctionnalités.

SLAM Global: Pendant que le SLAM local génère sa succession de sous-cartes, une tâche d'optimisation globale (généralement appelée "problème d'optimisation" ou "ajustement de pose clairsemé") s'exécute en arrière-plan. Son rôle est de réarranger les sous-cartes entre elles afin qu'elles forment une carte globale cohérente. Par exemple, cette optimisation est chargée de modifier la trajectoire actuellement construite pour aligner correctement les sous-cartes en ce qui concerne les fermetures de boucle.

Il s'agit essentiellement d'une optimisation du graphe de pose qui fonctionne en construisant des contraintes entre les nœuds et les sous-cartes, puis en optimisant le graphe de contraintes résultant. Les contraintes peuvent être considérées intuitivement comme de petites cordes reliant tous les nœuds entre eux. L'ajustement de pose clairsemé attache ces cordes ensemble. Le réseau résultant est appelé le "graphe de pose".

- Les contraintes non globales sont construites automatiquement entre les nœuds qui se suivent de près sur une trajectoire. Intuitivement, ces "cordes non globales" maintiennent la structure locale de la trajectoire cohérente.
- Les contraintes globales sont régulièrement recherchées entre un nouveau sous-map et les nœuds précédents qui sont considérés comme "suffisamment proches" dans l'espace (faisant partie d'une certaine fenêtre de recherche) et comme une correspondance forte (une bonne correspondance lors de l'exécution de la

correspondance par balayage). Intuitivement, ces "cordes globales" introduisent des nœuds dans la structure et rapprochent fermement deux brins.

Pour limiter la quantité de contraintes (et de calculs), Cartographer ne considère qu'un sous-ensemble de tous les nœuds proches pour la construction des contraintes.

9 Configuration et lancement du robot

9.1 Description du fonctionnement daemon de ROS

Le but de notre installation est d'exécute ROS Noetic sur le turtlebot « host » en en prenant le contrôle depuis un ordinateur « master ». Ceci est possible via une connexion ssh et des connexions lancées en daemon (processus en arrière-plan) sur ubuntu. Penchons-nous de plus près sur cette architecture.

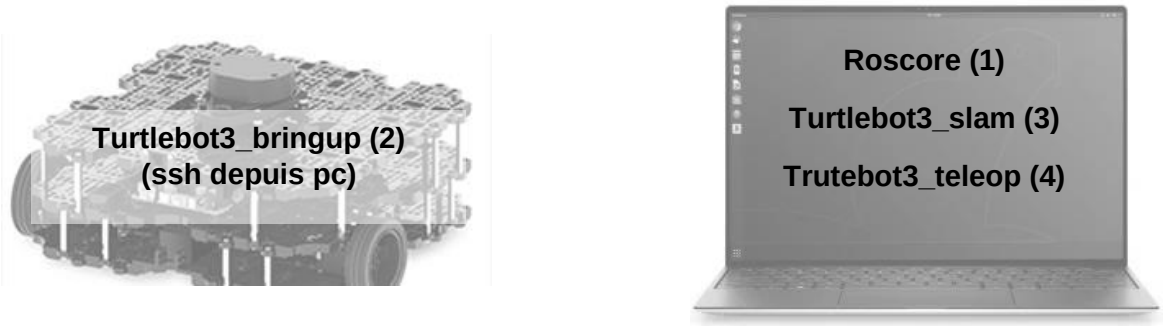

Figure 12 ROS installation

Processus exécutés sur les différentes machines

Lorsque l'on veut exécuter ROS afin d'utiliser la technique de SLAM, il faut donc lancer successivement les processus « daemon » :

- **Roscore :**

```
$ roscore
```

Nœud essentiel de ROS, il est exécuté sur le PC et assure le démarrage d'un Master ROS, d'un serveur de paramètres ROS et d'un nœud journal « rosout »

- **Bringup :**

```
$ ssh pi@{IP_ADDRESS_OF_RASPBERRY_PI}
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Instruction donnant au robot la consigne d'initialiser la connexion au Master afin de pouvoir en lire les consignes. Cette instruction est exécutée sur le robot depuis le pc, par l'intermédiaire d'une connexion SSH (protocole permettant de prendre le contrôle d'une machine depuis une autre, via le réseau auquel sont connectées les deux machines)

- **Slam :**

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
```

Cette instruction, exécutée sur le PC, permet d'exécuter l'utilitaire R-VIZ, qui exploite les données du capteur LiDAR afin de dresser une carte de son environnement.
Le daemon SLAM exécuté en arrière-plan permet à l'utilitaire R-VIZ de recevoir les données d'odométrie et du LiDAR. Ceci rend possible la création d'une carte en utilisant le principe de Simultaneous Localization And Mapping. Il est possible de spécifier le type de SLAM que l'on souhaite exécuter à ce moment.

- **Teleop :**

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Cette instruction, exécutée par le PC, permet d'utiliser les touches du clavier Z,Q,S,D,X pour contrôler le déplacement du robot. L'interaction avec le robot est assurée par la commande Roscore exécutée précédemment.

9.2 Installation de ROS et des modules de SLAM

Dans le cadre de notre projet, nous avons eu à installer la distribution « Noetic » de ROS, qui assure une meilleure compatibilité avec les systèmes d'exploitation récents, puisqu'il utilise Python 3, là où ROS Kinetic utilise Python 2.

Nous avons suivi la procédure conseillée par le manuel en ligne de Turtlebot. Ceci nous a permis d'installer sans encombre la méthode de SLAM intitulée Gmapping. Nous avons cependant rencontré un obstacle considérable lorsqu'il a fallu installer la méthode « cartographer », qui nous empêchait de lancer Roscore.

9.3 Configuration réseau du PC et du robot

La spécification des adresses IP respectives du master et du host s'effectue dans le fichier accessibke par la commande « nano ~/.bashrc ». Elle doit respecter la configuration décrite sur l'image suivante :

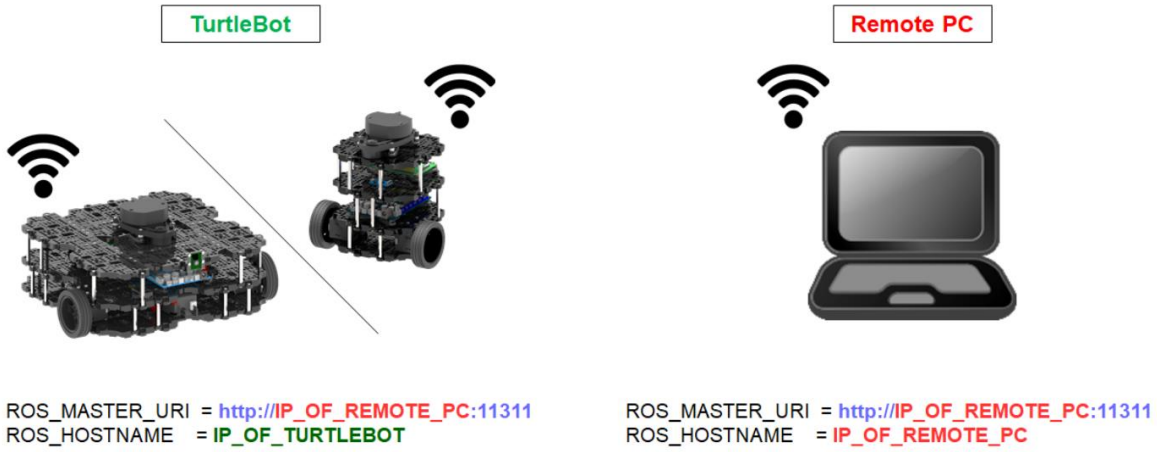

Figure 13 Configuration réseau du PC et du robot

Une des principales difficultés rencontrées lors de la réalisation de notre projet a été la configuration réseau des appareils utilisés. En effet, lors de la première installation, le robot a besoin d'une connexion respectant les critères suivants :

1) L'adresse IP du robot est connue et spécifiée dans le fichier /.bashrc du robot seul
2) L'adresse IP du PC est connue et spécifiée dans le fichier /.bashrc du robot et du pc
3) Le routeur wifi est capable de se connecter à internet
4) Le routeur wifi est joignable par un SSID et une clé WPA2 spécifiés dans l'un des fichiers de la carte SD

Le réseau WIFI-ENSAM ne respecte pas le critère 4) puisqu'il demande un identifiant et un mot de passe après connexion. Nous avons donc du dans un premier temps utiliser le partage de connexion d'un téléphone pour nous permettre d'accéder à internet tout en garantissant la communication du pc et du robot.

Ceci a entrainé un problème de respect des conditions 1) et 2) , puisque le portable utilisé ne pouvait pas gérer le DHCP (processus d'attribution des adresses IP) statique. Il a donc fallu modifier manuellement les fichiers /.bashrc à chaque début de séance.

L'utilisation d'un routeur wifi fixe capable de gérer des adresses statiques était également exclue dans un premier temps, puisque non connectable au réseau extérieur de l'école sans réaliser une procédure complexe auprès du service informatique de l'école.

Ce problème a finalement pu être réglé ***<u>en dernière séance de PJT</u>***, grâce à l'aide de Mr Romain DESCHODT. Ce dernier a réalisé la démarche pour autoriser la présence du routeur sur le réseau de l'école et m'accès de ce dernier à internet, permettant par la même occasion l'adressage statique du PC et du robot.

DHCP RESERVATIONS LIST

Enable	Host Name	MAC Address	IP Address
☑	IIWA	90:1B:0E:DA:5B:D8	172.31.1.148
☑	li-florit-02	18:db:f2:56:cc:1b	172.31.1.100
☑	li-florit-02_wifi	34:f3:9a:a1:65:e9	172.31.1.101
☑	li-lispen-12	30:24:32:e0:de:45	172.31.1.102
☑	LI-LISPEN-11	7c:7a:91:15:64:45	172.31.1.103
☑	Turtlebot	[illegible]	172.31.1.105
☑	Turtlebot Computer	5c:ba:ef:0b:8d:11	172.31.1.106

Figure 14 Direction IP

À l'avenir, ce problème ne devrait plus se produire, les adresses statiques ayant été définies, et le routeur ayant été connecté à internet. La seule consigne consistera à modifier l'adresse mac du pc « Turtlebot computer », en tapant « ifconfig » ou « hostname -a » dans un terminal, et en saisissant l'adresse ainsi obtenue dans l'interface du routeur wifi. Un document expliquant la procédure sera rendu disponible pour faciliter le travail des futurs groupes.

9.4 Mise en œuvre d'un algorithme de SLAM

Les dernières séances de PJT ont été l'occasion d'expérimenter l'algorithme de Gmapping. Celui-ci produit les résultats suivants, lors d'un passage aux ateliers :

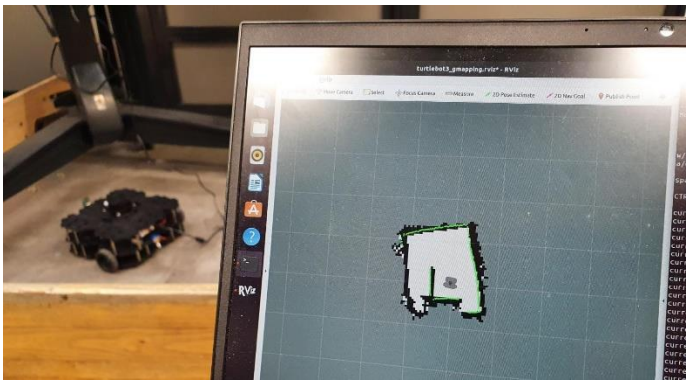

Figure 15 réalisation d'un premier slam dans la salle de robotique

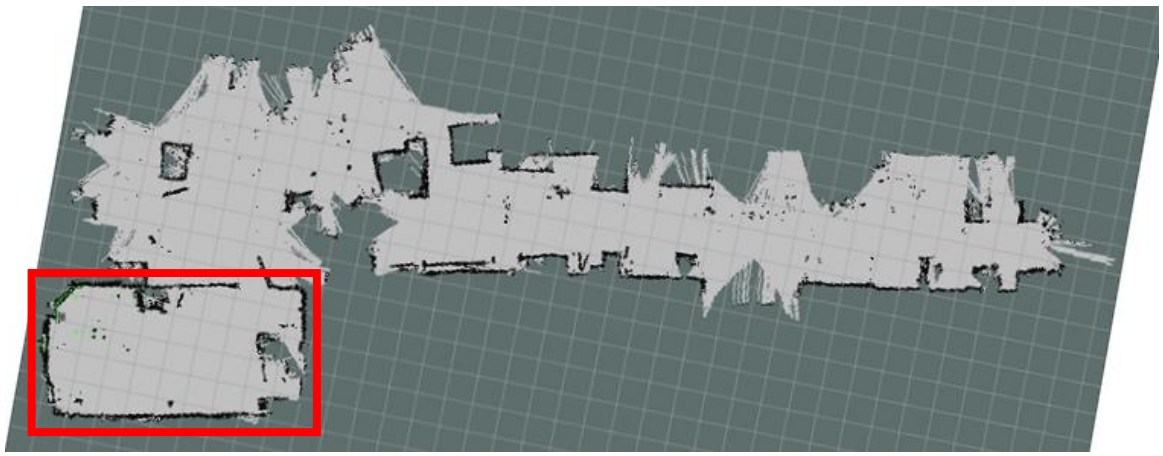

Figure 16 carte de la salle robotique et des ateliers

On remarque que l'odométrie entraîne une distorsion de la carte générée, due principalement au glissement des roues sur le sol.

Nous avons pu constater que certains défauts rendaient la carte complètement illisible. Par exemple, lorsque le robot croit avancer en se basant sur le mouvement de ses roues, alors qu'il se retrouve en réalité bloqué sans contact avec le sol. La carte alors générée vient se superposer sans aucun contrôle à la carte générée précédemment.

Lors d'un échange avec Mr Vilson Wenis BELLE, nous avons cependant pu déterminer que l'algorithme qu'il conviendrait d'utiliser sur de telles distances était l'algorithme « Hector », qui ne tient compte que des données du lidar, donc que de la position réelle du robot.

Le résultat obtenu peut être exploité au format « carte d'occupation », communément utilisé dans les algorithmes de slam. Ce dernier peut alors servir à créer une navigation dans l'espace.

10 Comparaison des résultats pour trois types de SLAM différents

La carte créée par le package Gmapping décrit le mieux décrit le mieux l'environnement réel. Les murs de l'environnement rendus ont le moins de bruit et la surface de la carte n'est en aucun cas déformée. Ce type de carte est tout à fait suffisante pour une localisation et la navigation.

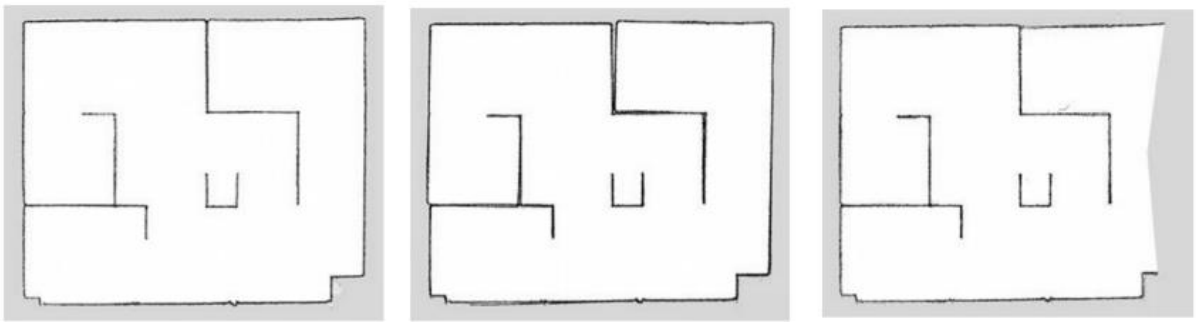

Figure 17 comparaison des methodes Gmapping, Karto et Hector

Gmapping utilise la méthode du filtre à particules. Cette méthode nécessite des modèles stochastiques de capteurs et d'odométrie. Le site valeurs prédéfinies de ces modèles ne décrit pas précisément le comportement du capteur sur le robot. Par conséquent, il était nécessaire de créer un modèle des capteurs utilisés pour améliorer les résultats du processus de cartographie.

Pour l'application gmapping dans ROS, une valeur spécifique est à définir. Cette valeur caractérise le modèle stochastique et elle sera représentée par l'argument d'entrée. Cette valeur a été déterminée pour une distance de 2,5 m, car en termes de multiplicité, cette valeur était la plus élevée dans l'environnement donné. Valeur était la plus élevée dans l'environnement donné. L'écart de référence pour cette distance est de 0,0041 m.

11 Conclusion

Nous l'avons vu, la technologie exigée par le slam est complexe. Nous avons réussi à la mettre en place malgré les difficultés techniques, ce qui permettra aux futurs groupes travaillant sur le sujet d'approfondir sur des bases saines, fonctionnelles. Nous avons cependant été freinés par des difficultés aussi bien techniques qu'organisationnelles :il nous est arrivé d'avoir des cours pendant les heures de projet, ce qui n'a pas aidé à aller au bout de toutes nos missions.

L'élargissement du sujet serait d'arriver à généraliser les algorithmes du turtlebot à d'autres robots, et de généraliser les algorithmes d'autres robots au turtlebot. Une automatisation du parcours réalisé en SLAM serait également étudiable en acquérant les connaissances nécessaires pour programmer un script exécutable sur ROS.

12 Bibliographie :

www.wikipedia.org

https://www.ros.org/

https://www.generationrobots.com/blog/en/ros-robot-operating-system-2/

wiki.ros.org

https://youtu.be/yWtGUk3PBms

Introduction to ROS – Robot Operating System – Daniel Serrano

http://lsi.vc.ehu.es/pablogn/investig/ROS/ROS_an_introduction.pdf

SLAM: Simultaneous Localization and Mapping - Wolfram Burgard, Cyrill Stachniss, Kai Arras, Maren Bennewitz

An Introduction to Robot SLAM (Simultaneous Localization And Mapping) - Bradley Hiebert-Treuer

Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms Hugh Durrant-Whyte, Fellow, IEEE, and Tim Bailey

https://youtu.be/wVsfCnyt5jA

Riisgaard, S., & Blas, M. R. (2003). SLAM forutorial Approach to Simultaneous Localization and Mapping, 22(1-127)

Sarah Kudan (2016). An Introduction to Simultaneous Localisation and Mapping.

Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping:
part I. IEEE Robotics Automation Magazine.

CONCEPTION ET INTEGRATION D'UN CAPTEUR LIDAR 3D POUR LA NAVIGATION AUTONOME DES ROBOTS MOBILES EN TERRAIN INCONNU JEAN-PHILIPPE ROBERGE

https://www.raspberrypi.org/

https://www.jmest.org/wp-content/uploads/JMESTN42353033.pdf
https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html
https://buildmedia.readthedocs.org/media/pdf/google-cartographer-ros/latest/google-cartographer-ros.pdf
https://linklab-uva.github.io/autonomousracing/assets/files/SLAM.pd