Université Clermont-Auvergne

École Universitaire de Physique et d'Ingénierie

Compte Rendu de TP

Contrôle de Robot Limo avec ROS2

Réalisé par :

Yannis LOUMOUAMOU

9 décembre 2024

Table des matières

1 Introduction

1.1 Contexte du projet

Ce projet fait partie d'un TP visant à initier aux concepts de base de ROS2 tels que les noeuds, topics, services et actions, afin de développer des applications simples.

Une compréhension des fondamentaux de la programmation en Python ainsi qu'un bon sens sont des pré-requis pour mener à bien ce projet. Une connaissance de ROS1 est bénéfique. Je tiens donc à préciser que n'ayant pas l'habitude de programmer en Python et ne disposant d'aucune connaissance de ROS1, il m'a été au début très difficile de saisir l'ensemble des concepts et de m'y adapter. Cependant, j'ai énormément utilisé la documentation mise à notre disposition ainsi que mes propres recherches sur internet pour pallier mes lacunes. Cela m'a permis de mieux comprendre et de surmonter les difficultés rencontrées tout au long de la réalisation du projet.

1.2 Objetif du projet

Le projet, qui fait suite à une introduction aux éléments fondamentaux de ROS2, vise à développer un système de contrôle pour le robot Limo, permettant de basculer entre un mode manuel et un mode autonome, en utilisant ROS2. Le robot Limo sera capable de réagir de manière sécurisée grâce à un LIDAR permettant de détecter les obstacles et d'agir en conséquence.

L'objectif est de base d projet est de contrôler le robot suivant deux modes distincts : un mode manuel, où le robot sera commandé via un joystick, et un mode autonome, où il devra sortir d'un labyrinthe en suivant le mur de gauche. Le mode manuel sera activé par un appui sur le bouton R1, tandis que le mode autonome sera enclenché par le bouton R2. Dans les deux modes, des procédures de sécurité seront mises en place pour garantir la sécurité du robot : si le robot se trouve à moins de 50 cm d'un mur en face de lui, il sera interdit d'avancer, et si la distance est inférieure à 30 cm, il reculera doucement à une vitesse de -0.2.

2 Travail préparatoire : planification du projet

Lors de la phase préparatoire, le cahier des charges a été analysé et ajusté pour mieux répondre aux contraintes techniques. Un plan détaillé de réalisation a été mis en place pour structurer le projet.

2.1 Description du plan de projet

Le plan de projet a connu beaucoup de changements au cours des différentes séances de TP du fait d'une meilleure compréhension des attentes et objectifs du travail demandé. Celui-ci est donc très différent de celui présenté lors de la première séance de TP. Cependant, les dates de validation des comportements restent inchangées.

2.1.1 Description des noeuds

- **Noeud numéro 1** :
  - **Nom** : controle_manuel
  - **Entrées** : commandes du joystick.
    Topic /joy : messages de type sensor_msgs/msg/Joy provenant du joystick pour transmettre les commandes utilisateur.
  - **Sorties** : vitesses du robot.
    Topic /cmd_vel : messages de type geometry_msgs/msg/Twist contenant les consignes de vitesse linéaire et angulaire pour le mouvement du robot.
  - **Fonction** : contrôler manuellement la vitesse du robot via le joystick
  - **Noeud créé par l'étudiant**
- **Noeud numéro 2** :
  - **Nom** : controle_autonome.
  - **Entrées** : données du LIADR.
    Topic /scan : messages de type sensor_msgs/msg/LaserScan provenant du LIDAR, contenant les données de distances autour du robot.
  - **Sorties** : vitesses du robot.
    Topic /cmd_vel_safe : messages de type geometry_msgs/msg/Twist contenant les commandes sécurisées pour piloter le robot.
  - **Fonction** : se déplacer de manière autonome en suivant un mur situé à gauche
  - **Noeud créé par l'étudiant**
- **Noeud numéro 3** :
  - **Nom** : securite.
  - **Entrées** : données du LIADR et consignes de vitesse :
    Topic /scan : messages de type sensor_msgs/msg/LaserScan .
    Topic /cmd_vel : messages de type geometry_msgs/msg/Twist .
  - **Sorties** : vitesses du robot.
    Topic /cmd_vel : messages de type geometry_msgs/msg/Twist .
  - **Fonction** : assurer la sécurité des déplacements du robot en vérifiant les obstacles situés dans la direction avant. , en ajustant les commandes pour éviter les collisions selon la distance de l'obstacle détecté.
  - **Noeud créé par l'étudiant**
- **Noeud numéro 4** :
  - **Nom** : choix.

  - **Entrées** : commandes du joystick :
    Topic /joy : messages de type sensor_msgs/msg/Joy.
  - **Sorties** : lancement de processus ROS : basculement entre les modes en exécutant des commandes ros2 launch.
  - **Fonction** : basculer dynamiquement entre différents modes de contrôle du robot en fonction des boutons du joystick.
  - **Noeud créé par l'étudiant**

- **Noeud numéro 5** :
  - **Nom** : joy.
  - **Entrées** : aucune entrée externe, communique directement avec un périphérique de joystick (manette) connecté au système.
  - **Sorties** : commandes du joystick :
    Topic /joy : messages de type sensor_msgs/msg/Joy.
  - **Fonction** : sert d'interface entre un périphérique joystick physique et le système ROS2.
  - **Noeud existant fourni par le paquet joy**

2.1.2 Description des étapes

- **Étape 1** :
  - **Description du comportement** : comportement simple : Mise en place du mode manuel permettant de contrôler le robot Limo à l'aide d'un joystick
  - **Noeuds mis en oeuvre** : joy_node et control_manuel.
  - **Validation prévue à la séance** : 2
  - **Validé à la séance** : 3
  - **Numéro de commit** : 147
  - **Schéma** : Joystick –> [joy_node] –> /joy –> [controle_manuel] –> /cmd_vel –> Robot Limo

- **Étape 2** :
  - **Description du comportement** : comportement simple amélioré : Mise en place du mode manuel sécurisé.
  - **Noeuds mis en oeuvre** : joy_node , control_manuel et securite.
  - **Validation prévue à la séance** : 2
  - **Validé à la séance** : validé après les séances de TP
  - **Numéro de commit** : 206

  - **Schéma** : Joystick –> [joy_node] –> /joy –> [controle_manuel] –> /cmd_vel_raw –> LIDAR –> /scan –> [securite] –> /cmd_vel –> Robot Limo

- **Étape 3** :
  - **Description du comportement** : comportement autonome : Implémentation du mode de contrôle autonome, où le robot suit un mur à une distance prédéfinie.
  - **Noeuds mis en oeuvre** : controle_autonome.
  - **Validation prévue à la séance** : 2
  - **Validé à la séance** : validé après les séances de TP
  - **Numéro de commit** : 254
  - **Schéma** : LIDAR –> [controle_autonome] –> /cmd_vel –> Robot Limo
- **Étape 4** :
  - **Description du comportement** : comportement autonome amélioré : Intégration du mode autonome avec un mécanisme de sécurité.
  - **Noeuds mis en oeuvre** : control_autonome et securite.
  - **Validation prévue à la séance** : 2
  - **Validé à la séance** : validé après les séances de TP
  - **Numéro de commit** : 254
  - **Schéma** : LIDAR –> [controle_autonome] –> /cmd_vel_raw –> [securite] –> /cmd_vel –> Robot Limo
- **Étape 5** :
  - **Description du comportement** : permet à l'utilisateur de basculer entre différents modes (manuel, automatique, manuel amélioré, autonome amélioré) en appuyant sur les boutons correspondants du joystick.
  - **Noeuds mis en oeuvre** : choix.
  - **Validation prévue à la séance** : 3
  - **Validé à la séance** : validé après les séances de TP
  - **Numéro de commit** : 256
  - **Schéma** : Joystick (choix) –> [choix] –> Choix du mode (manuel, automatique, etc.)

2.2 Méthodologie utilisée

Dans le cadre de cet exercice, nouveau paquet loumouamou_limo a dû être créé, compte-tenu des consignes de validation du projet.

Les critères suivants ont été appliqués tant bien que mal pour chaque comportement développé :

- être implémentée dans un fichier launch,
- ne pas perturber les comportements plus simples existants,
- faire l'objet d'un commit dans le dépôt

Pour implémenter mon projet, j'ai donc commencé par créer un package ROS2 spécifique pour mon robot avec mon nom. Après la création du package, j'ai ajouté les dépendances nécessaires dans le fichier package.xml, ce qui m'a permis d'assurer que tous les modules nécessaires soient accessibles pour les nœuds que j'allais développer.

Ensuite, j'ai écrit chaque nœud individuellement, en les testant au fur et à mesure afin de m'assurer que chaque fonctionnalité fonctionnait correctement avant de passer à l'intégration suivante. Une fois chaque nœud validé, je les ai ensuite intégrés dans des fichiers de lancement (launch), afin de faciliter leur exécution et leur gestion. Sans oublier des commits réguliers pour suivre l'avancement du développement et faciliter le suivi des modifications.

Enfin, pour que chaque nœud soit correctement reconnu par le système, j'ai ajouté une entrée point dans le fichier setup.py, ce qui a permis d'enregistrer chaque nœud en tant que module exécutable au sein du package.

3 Architecture et conception

L'architecture des nœuds et des fichiers de lancement a été conçue pour offrir une gestion claire des responsabilités, ainsi qu'une communication efficace entre les différents composants du système. Les fichiers launch permettent un démarrage rapide des nœuds nécessaires pour chaque mode de fonctionnement, tout en intégrant les **remappages** lorsque j'ai jugé cela nécessaire.

3.1 Architecture des nœuds

Chaque nœud a été conçu pour être indépendant, ce qui permet de tester et déboguer chaque composant isolément avant de les combiner. Par exemple, les nœuds de contrôle manuel, de sécurité, et autonome sont séparés afin que chaque fonction puisse être améliorée ou modifiée sans affecter les autres :

- **controle manuel** : gère l'entrée du joystick pour déplacer le robot manuellement. Il reçoit les commandes via le joystick et publie les vitesses appropriées sur le topic /cmd_vel_raw.,
- **controle_autonome** : met en œuvre la logique pour que le robot suive un mur. Il analyse les données LIDAR pour ajuster sa trajectoire en fonction de la distance du mur.,

- **securite** : surveille la distance des obstacles détectés par le LIDAR et applique des mesures de sécurité en interrompant ou en ajustant la commande du robot pour éviter les collisions.,

- **choix** : permet de basculer entre différents modes de fonctionnement via des boutons du joystick, en utilisant un processus pour chaque mode. Pour ce noeud, j'ai choisi dutiliser des processus externes via **subprocess.Popen** pour lancer les différents modes, ce qui permet de garantir une transition fluide entre les modes et évite de redémarrer l'ensemble du système à chaque changement de mode. .

3.2 Architecture des launchs

Les fichiers launch sont utilisés pour démarrer et configurer l'ensemble des nœuds nécessaires à l'exécution de chaque mode ou comportement. Ils ont été organisés comme suit :

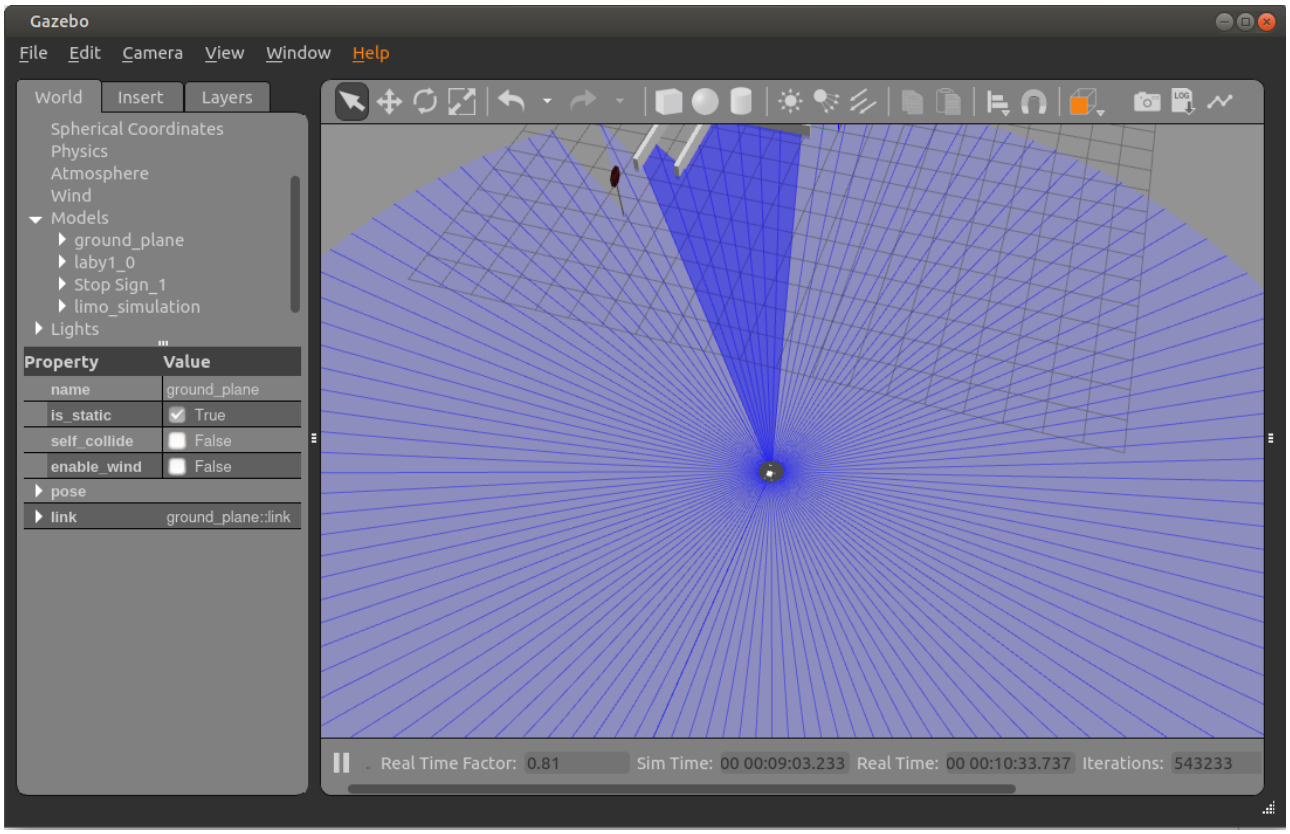
- Chaque fichier launch est dédié à un mode ou **un comportement spécifique**, comme l'exige le cahier des charges

- Dans les fichiers de lancement, certains topics sont **remappés** pour permettre aux différents nœuds de communiquer via des topics intermédiaires ou sécurisés. Par exemple : /cmd_vel est remappé en /cmd_vel_raw dans certains launchs pour que les commandes brutes et sécurisées soient traitées de manière isolée avant de passer à l'exécution finale, le nœud de sécurité remappe les topics de commandes et de capteurs pour appliquer des restrictions de sécurité.
  Ce remappage permet une configuration flexible des communications entre les nœuds et facilite l'adaptation aux exigences de chaque mode de fonctionnement sans modifier le code source des nœuds.

4 Tests et validation

La majorité des tests ont été réalisés dans un environnement simulé via Gazebo, en raison du manque de temps pour effectuer les tests en conditions réelles avec le robot. Cela a permis de tester l'ensemble des comportements du robot dans un cadre contrôlé et de simuler différents scénarios. Seul le nœud de contrôle manuel a pu être testé en conditions réelles pour valider son bon fonctionnement.

En général, les résultats des tests en simulation ont été satisfaisants. La plupart des nœuds et launchsont bien fonctionné et produit les comportements attendus, en particulier ceux relatifs au mode manuel manuel du robot, ainsi que le mode autonome avec les conduition de sécurité qui a permis au robot de sortir du labyrinthe tout seul, comme illustré ci-dessous.



5 Difficultés rencontrées et solutions

Cependant, d'importantes difficultés ont été constaté :

5.1 Mode autonome

D'abord avec le mode autonome de suivi de mur, qui ne fonctionne pas comme prévu. Le robot a parfois dévié de sa trajectoire. Lorsque le robot est supposé avancer de manière rectiligne, il dévie légèrement vers la gauche ou la droite. Ce phénomène semble est peut-être dû à une mauvaise configuration des axes du joystick, ce qui a conduit à un contrôle moins précis du robot. Il pourrait également être lié à une gestion incorrecte des données du LIDAR ou à une erreur dans le calcul des vitesses de rotation.

Malgré plusieurs tentatives pour résoudre ce problème, une solution efficace n'a pas su être trouvée.

5.2 Noeud de choix des modes

Autre problème, cette fois-ci rencontré lors de l'implémentation du noeud de sélection des modes (choix) dans le launch princial, a été l'incapacité du système à appliquer correctement le mode sélectionné après une première sélection suite à l'appui sur un bouton (R1, R2, L1 ou

L2). Ce dysfonctionnement n'a pas été résolu faute de temps mais semble être dû à plusieurs facteurs potentiels :

- conflit dans l'exécution des processus : lorsqu'un mode est changé, le processus précédent n'est peut-être pas correctement arrêté avant de lancer un nouveau mode, ce qui empêche le changement de mode de s'appliquer correctement,
- mauvaise gestion des remappings de topics dans les fichiers launch

L'ajout des logs pour suivre l'exécution des processus et vérifier la gestion des remappings dans les fichiers launch ne m'a pas parmis de détecter la cause de ce comportement indésirable.
J'ai également tenter de passer par un **service** pour gerer la sélection des modes, comme on peut le constater à partir du commit 212. Cette approche n'ayant pas été abordée en TP et nécessitant de modifier le document package.xml m'a en fin de compte paru trop compliquée.

5.3 Simulation gazebo et commits sur svn

Encore un autre problème rencontré au cours du développement a été la simulation, qui a présenté de nombreux bugs. Bien que la simulation dans Gazebo soit généralement un outil pratique et efficace pour tester le comportement du robot, elle n'a pas toujours fonctionné comme prévu, avec des problèmes de synchronisation et des erreurs dans la gestion des capteurs ou des mouvements du robot. Ces dysfonctionnements ont entravé les tests et la validation des différents modes de fonctionnement, obligeant à passer plus de temps pour tenter de comprendre et de résoudre les problèmes qui ne dépendaient en fait pas de moi. La solution a parfois été de fermer complétement la machine virtuelle.
De plus, j'ai eu des difficultés initiales à faire des commits réguliers, ce qui a entraîné quelques incohérences dans le suivi du développement. En effet, je n'avais pas bien établi une routine de commits fréquents au début (des oublis très fréquents en fin de séances de TP), ce qui a compliqué le processus de gestion du versionnement et la traçabilité des changements effectués. Cela a été une source de frustration et constitue un point à corriger pour mes futurs projets en groupe.

6 Conclusion et perspectives

En somme, bien que des difficultés aient marqué ce projet, les objectifs de base ont été atteints et une bonne base a été posée pour les développements futurs de projet sous ROS2. La configuration et mise en place d'un environnement de travail pour développer avec ROS2 a été relativement bien maîtrisé. Avec un peu plus de temps et de ressources, il aurait été possible de perfectionner les comportements du robot, de résoudre les problèmes techniques en cours et d'améliorer la robustesse du système dans son ensemble.

References

[1] Formulaire ROS2, Moodle UCA, https://moodle2024.uca.fr/pluginfile.php/582530/mod_resource/content/3/FormulaireRos2.pdf

[2] Ostasse, A., *Poly copie ROS2*, LAAS-CNRS, https://homepages.laas.fr/ostasse/Teaching/ROS/2022/anf-2rm-ros2-2022-polycopie.pdf

[3] Documentation ROS2 en français, ROS Discourse, https://discourse.ros.org/t/documentation-en-francais/28896/4

[4] Documentation Python, bibliothèque subprocess, https://docs.python.org/fr/3/library/subprocess.html

[5] StackOverflow, How to use subprocess.Popen Python, https://stackoverflow.com/questions/12605498/how-to-use-subprocess-popen-python

[6] ROS2 Documentation, Understanding ROS2 Services, https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

[7] YouTube, *Introduction à ROS2*, https://www.youtube.com/watch?v=0aPbWsyENA8&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy