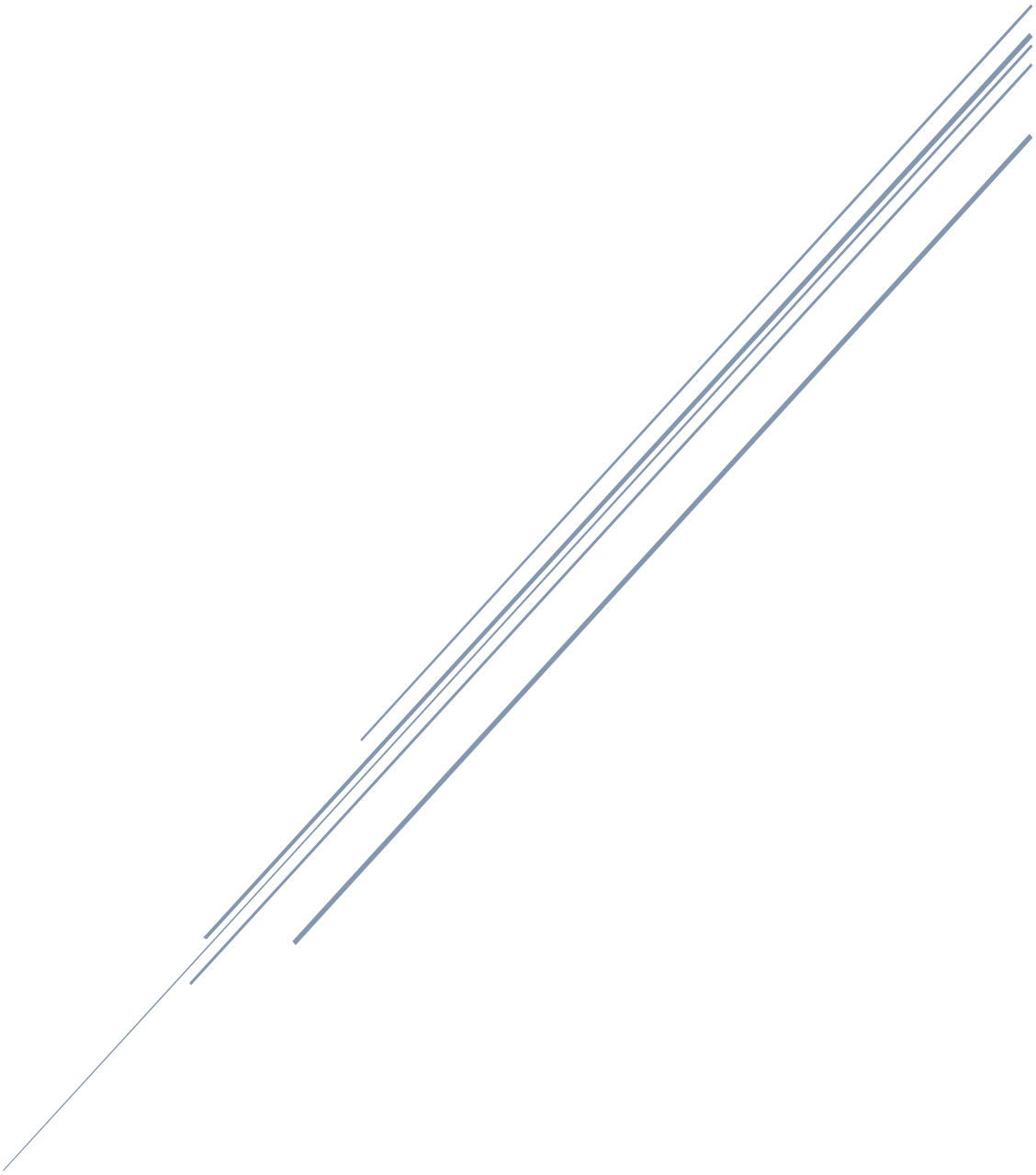
NAVIGATION D'INTERIEUR ET EN TEMPS REEL POUR ROBOTS

Projet GSE, dernière année du cycle ingénieur



Polytech Nice-Sophia – Section Electronique – Mineure SA
Hakan TASPINAR & Saad BARMAKI

Table des matières

Table des illustrations

I. Introduction :

A. Objectif du projet :

L'objectif du projet est d'implémenter en temps-réel un système de navigation pour véhicule robot et de mettre en place un démonstrateur complet, basé sur un robot existant fonctionnant sous l'environnement de travail ROS (Robot Operating System) : le Turtlebot3 Burger (cf. figure 1).

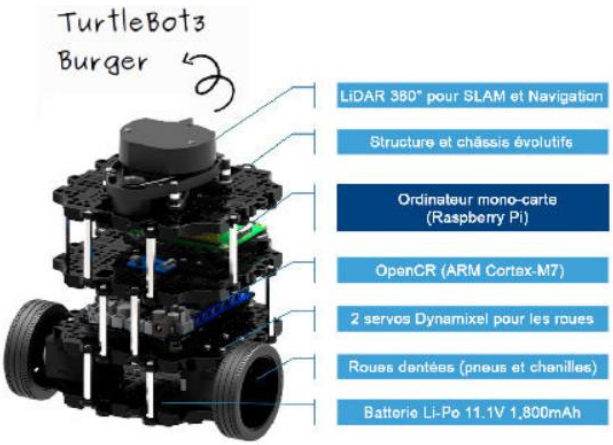


Figure 1 : Turtlebot3 Burger, composition du robot

B. Vue globale du projet :

La première étape consiste à l'implémentation du système de navigation. Pour cela, il faut utiliser un ordinateur portable utilisant le système d'exploitation Ubuntu (sous Linux donc) ou un de ses dérivés (exemple : Ubuntu MATE). En effet, le but étant de réaliser une navigation intérieure, il est nécessaire de mettre en place un poste de travail portatif. Sur ce dernier, ROS (Robot Operating System) sera installé et correctement configuré, tout comme sur le robot.

Par ailleurs, un Lidar qui est un scanner laser autonome portable, sera utilisé afin de générer une carte de navigation d'intérieur. L'environnement définit pour ce projet est l'un des bâtiments de l'école Polytech Nice-Sophia et plus particulièrement l'accueil de l'école. Le Lidar ainsi que ROS grâce à ses outils, permettent la collecte d'échantillon et la mise à jour en temps réel d'une cartographie d'un environnement. En effet, ce dispositif d'acquisition basé sur le capteur Lidar et contrôlé par un ordinateur mobile (robot embarqué ou ordinateur portable), communique via une connexion série en USB, ce qui permet de récupérer les données. Elles seront ensuite exploitées afin d'exécuter la navigation SLAM (Simultaneous Localization And Mapping) et sa visualisation via des outils fournis par ROS (exemple : RViz). Dans le but de mapper l'environnement et d'utiliser l'algorithme SLAM, plusieurs paquets intégrés dans ROS peuvent être utilisé : Gmapping, Hector Mapping ou encore Cartographer ROS (développé pour ce dernier par Google).

Enfin, l'envoi des commandes de navigation entre la station portable (Ubuntu) et le robot (Turtlebot3 Burger) se fait via une communication sur un même réseau. Ici, si la connexion le permet, le Wi-Fi de l'école Polytech Nice-Sophia sera utilisé. Dans le cas contraire, un partage de connexion de données mobiles sera effectué.

II. La partie matérielle (hardware) :

A. TurtleBot3 Burger :

Pour ce projet, le robot TurtleBot3 Burger est utilisé et est composé de 4 étages. Au 1ère étage (étage le plus bas), il y a de chaque côté une roue denté et un servo-moteur DYNAMIXEL afin de pouvoir bouger les roues. Également, se trouve une batterie Li-Po 11,1V de 1 800 mAh afin d'utiliser le robot sur batterie. Au 2nd étage, se trouve une carte OpenCR 1.0 relié notamment à la batterie et permettant l'alimentation de tout le système sur batterie. Cette carte permet également la communication entre la carte Raspberry (situé à l'étage du dessus) et les servos-moteurs. Au 3ème étage, se trouve donc la Raspberry Pi 3, branché en USB à l'OpenCR et en UART au Lidar (situé au sommet du robot). Suite à un problème de batterie, l'alimentation micro-USB de la Raspberry Pi 3 (branché au PC) a également été utilisé pendant la recharge de la batterie. Pour finir, donc, le RPLidar A1 qui permettra de se situer dans la carte qui aura été constitué, une fois la navigation autonome du robot lancé.

Figure 2 : ensemble du matériel constituant le robot TurtleBot3 Burger

Le montage du robot a été effectué par les différents binômes participant à ce projet GSE. Notre groupe a été chargé de constituer l'étage 3 concernant la carte Raspberry Pi 3 et son branchement avec l'OpenCR ainsi que le Lidar.

B. Autre matériel :

Avant la réception de tout ce matériel et afin d'avancer dans le projet et possiblement mapper une carte, il était possible d'utiliser un des 2 Lidars misent à notre disposition : un RPLidar A1 et un RPLidar A2. Entre ces 2 modèles, il y a peu de différence, hormis concernant l'angle du Lidar qui permet de prendre plus d'information pour la version la plus récente (RPLidar A2). Ainsi, pour cette partie, un RPLidar A2 a été utilisé en le branchant sur l'ordinateur portable ayant comme système d'exploitation Ubuntu.

Figure 3 : RP Lidar A2

Le RPLidar A2 (ou A1) utilise le principe de la méthode trigonométrique, comme indiqué dans la figure ci-dessous. Le laser émet une lumière laser et une fois l'objet irradié, la lumière réfléchie est reçue par le capteur CCD linéaire de la caméra. Étant donné que le laser et le détecteur sont séparés d'une certaine distance, selon le chemin optique, des objets de différentes distances seront imagés dans le capteur CCD, à un endroit différent. Calculée selon la formule du triangle, la distance de l'objet mesuré peut en être dérivée et obtenu. Désormais, il existe également des capteurs TOF, beaucoup plus récents, plus précis mais également beaucoup plus chère. Pour ce projet et l'utilisation de l'algorithme SLAM, ces capteurs RPLidar sont suffisants.

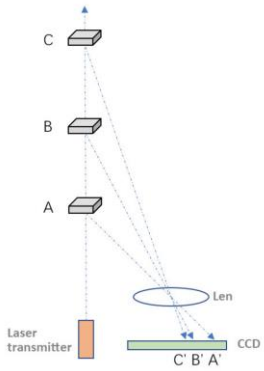


Figure 4 : le principe de la méthode trigonométrique

Enfin, une carte SD de 32 Go a été utilisé afin de pouvoir booter sur un environnement ROS, préconfiguré pour la Raspberry Pi du TurtleBot3 Burger (image fournie sur le site du concepteur ROBOTIS), et l'insérer dans l'emplacement qui lui est réservé.

III. La partie logicielle (software) :

A. Constitution de l'environnement (PC et Raspberry Pie 3) :

La station de travail utilisé pour ce projet a été un ordinateur portable utilisant Kubuntu 20.04 (dernière version LTS au moment du projet), avec la version Noetic de ROS (Robot Operating System, dernière version stable au moment du projet). Toutes les mises à jour (paquets ROS…etc.) ont été effectué au préalable afin d'essayer de garantir un maximum de compatibilité, voire de stabilité, avec les outils que nous devrions utiliser. Pour cela, le site du concepteur du TurtleBot3 Burger https://emanual.robotis.com/ a été utilisé. Parmi les dépendances de paquet ROS nécessaire, il est notamment utile de citer teleop, msgs, amcl, map-server, rviz, gmapping et navigation.

En effet, pour la localisation du robot, le package ROS d'AMCL (Adaptive Monte Carlo Localization) fonctionne bien. et est simple d'utilisation. AMCL ne fonctionne d'ailleurs qu'avec des balayages/scans laser et des cartes qui ont été réalisé avec un laser. La seule chose à laquelle il faut faire attention est qu'il nécessite un message d'odométrie (= mesure des distances parcourues). Il existe différentes façons de générer ce dernier. Les données d'odométrie peuvent être extraites d'encodeurs de roue, d'IMU (Odométrie Visuelle-inertielle), etc. qui peuvent être utilisées pour générer le message d'odométrie qui peut être fourni à AMCL. Une autre astuce consiste à utiliser la pose obtenue à partir de la cartographie Hector pour générer un message d'odométrie qui peut ensuite être fourni à AMCL. Si la pose de cartographie Hector est utilisée pour générer un message

d'odométrie, aucune odométrie externe n'est requise et le résultat est assez précis. C'est justement cette dernière technique qui sera utilisé dans le cadre de notre projet.

Du côté du système utilisé sur la Raspberry Pi 3 du TurtleBot3 Burger, une image téléchargeable là-encore sur le site ROBOTIS est disponible. Celui choisi, a été la version Kinetic de ROS pour le robot, puisque c'était la seule version où l'environnement graphique était déjà disponible. En effet, malgré différentes tentatives, il n'a pas été possible de le faire sur une version Noetic de l'image fourni par le concepteur. L'environnement graphique permettant de configurer plus facilement l'ensemble de la suite des éléments et, une version différente de ROS par rapport à l'ordinateur portable n'étant pas une gêne pour l'exécution des commandes, le choix de ROS Kinetic pour la Raspberry Pi 3 a été effectué. Pour cela, le logiciel Raspberry Pi Imager ainsi qu'un lecteur de carte SD branché sur le PC a été utilisé. Une fois l'image télécharger, nous la formattons de sorte à pouvoir l'intégrer dans la carte SD, qui sera intégré avant l'alimentation et le démarrage du robot.

B. Configuration sur le même réseau :

Désormais, il faut configurer l'ordinateur portable et la Raspberry Pi 3 sur le même réseau afin de pouvoir envoyer des commandes depuis le PC et les recevoir par le TurtleBot3 Burger.

Figure 5 : configuration du réseau

Pour cela, il faut configurer dans le fichier ~/.bashrc des deux appareils (ordinateur portable et Raspberry Pi 3) l'adresse IP du maître et de l'esclave. Pour l'ordinateur portable, il doit se contenter d'envoyer des commandes et n'écouter en aucun cas le robot, il faut donc qu'il configure l'adresse IP du maître (master en anglais) ROS_MASTER_URI, comme étant sa propre adresse IP (cf. figure 5). Afin de relier l'ordinateur au robot, pour l'adresse IP du maître, il ne faut pas oublier d'ajouter 11311 pour les deux appareils. En ce qui concerne le TurtleBot3 Burger, il faut qu'il écoute les commandes de l'ordinateur portable et donc doit inscrire la même adresse IP que précédemment pour la ligne ROS_MASTER_URI. La ligne ROS_HOSTNAME concerne l'adresse IP de l'appareil lui-même. Pour connaître chacune des adresses IP, la commande ifconfig est utilisée.

Le Wi-Fi de l'école n'étant pas assez de bonne qualité, nous avons dû avoir recours à un partage de connexions (données mobiles).

C. Procédure de cartographie :

Dans un premier temps il a fallu cartographier l'environnement à naviguer. Pour ce faire, il faut établir la connexion avec le Lidar en effectuant l'installation d'Hector SLAM dans le répertoire catkin_ws en suivant le tutoriel (voir readme.md). Une fois cela terminé, il suffit alors de lancer la commande suivante : roslaunch hector_slam_launch tutorial.launch .

Ensuite, il reste à explorer l'espace à cartographier. Or, dans le but d'obtenir une carte optimale, il est nécessaire de se déplacer lentement et de maintenir le Lidar à la même hauteur. En outre, il faut à tout prix éviter les bordures transparentes car celle-ci peuvent créer des défauts de cartographie, et biaisent par conséquent les déplacements du robot.

Une fois la cartographie terminée, la carte est enregistrée dans un répertoire nommé maps (voir readme.md pour la procédure), afin de pouvoir la réutiliser plus tard.

D. Procédure de Simulation :

Avant de procéder à une simulation de navigation autonome, une simulation à l'aide du « téléopérations » a été réalisé. Le but de cette approche a été de s'assurer du bon fonctionnement du robot dans la carte créée précédemment, où nous avons pu parcourir à l'aide des touches du clavier.

Lors de la simulation autonome, les fonctionnalités du logiciel Rviz sont utilisées et permettent d'établir une « position initiale 2D » (2D pose estimate), ainsi qu'une « destination finale » (2D Nav Goal). Le robot parcourt cette distance en recalculant constamment sa trajectoire et évitant d'éventuels obstacles.

E. Procédure de Navigation :

Avant de procéder à la navigation nous avons procéder à quelques vérifications. Tout d'abord, il faut s'assurer que le « firmware » de la carte OpenCR était bien configuré notamment grâce à cette manipulation :

- Appuyez sur la touche PUSH SW 1 et maintenez-la enfoncée pendant quelques secondes pour ordonner au robot d'avancer de 30 centimètres (environ 12 pouces).

- Maintenez la touche PUSH SW 2 enfoncée pendant quelques secondes pour ordonner au robot de pivoter de 180 degrés sur place.

Après s'être assuré de la bonne configuration du firmware, à l'image de la simulation, il est alors possible d'effectuer une navigation manuelle à l'aide des touches du clavier (téléopérations). Pour ce faire, il a fallu s'assurer de la connexion en entre le robot et l'ordinateur notamment en vérifiant la liste des topics.

Ensuite, alors vient la navigation autonome, en vérifiant également la connexion entre l'ordinateur et le robot en affichant la liste des différents topics (cf. tableau 1).

F. Remarque(s) / difficulté(s) rencontrée(s) :

Nous avons pu rencontrer des problèmes lors de la simulation ainsi que la navigation, car certaines zones étaient mal cartographiées, dû à la limitation matérielle et logicielle, et donc le robot pouvait avoir un peu de mal en fonction du scénario (s'il était trop proche du mur, dans une zone mal cartographiée). Il a fallu notamment re-cartographié et éviter de rester exposé notamment aux zones où les rayons du soleil pouvaient facilement rentrer (exemple : baies vitrées de l'accueil). Nous avons fourni des exemples de ces deux cas de figures dans notre GitHub.

G. Topics :

Voici la liste des topics obtenue pour la navigation autonome :

Tableau 1 : topics après roscore et bring up, pour la navigation autonome

Topics	Explanation
/battery_state:	Contains battery voltage and status
/cmd_vel	Control the translational and rotational speed of the robot unit in m/s, rad/s
/diagnostics:	Contains self diagnostic information
/firmware_version:	Contains the Turtlebot3 hardware, firmware,and software information.
/imu:	Topic that includes the attitude of the robot based on the acceleration and gyro sensor
/magnetic_field:	Measurement of the Magnetic Field vector at a specific location
/joint_states:	This package publishes sensor_msgs/JointState messages for a robot. The package reads the robot_description parameter from the parameter server, finds all of the non-fixed joints and publishes a JointState message with all those joints defined.
/odom:	Contains the Turtlebot3's odometry information based on the encoder and IMU
/rosout:	Standard ROS topic for publishing logging messages.
/rosout_agg:	Aggregated feed of messages published to /rosout.
/motor_power:	Dynamixel Torque On/Of
/sounds_state:	Output Beep Sound
/tf:	Contains the coordinate transformation such as base_footprint and odom

III- Conclusion :

En conclusion, ce projet nous aura permis de découvrir la technologie, ainsi que les différentes étapes permettant d'obtenir un moyen de navigation autonome intérieur pour robot. En effet, il nous a fallu dans un premier temps comprendre les caractéristiques et les composantes de notre Lidar, expérimenter différentes pistes, afin d'initier le processus de cartographie étant donné que SLAM Gmapping ne semblait pas fonctionner. Puis, engager différentes simulations dans différents environnements y compris nos cartes mappées. Enfin, prendre en main la Raspberry Pi et configurer correctement notre environnement dans la carte SD, dans le but de pouvoir naviguer avec le robot. En plus, du fait d'avoir découvert ce procédé passionnant qu'est la navigation autonome, ce projet nous a permis d'aiguiser notre sens de la recherche, de la réflexion mais aussi de développer notre esprit d'équipe et de la communication. Toutes ces qualités et ces connaissances acquises durant ce projet nous seront utiles dans notre vie ultérieure en tant qu'ingénieur que ce soit d'un point de vue technique ou humain.