École Centrale de Lille

Rapport Mini projet - Robot explorateur

Système de Transport Autonome

Auteurs: Hippolyte BERGERAULT - Thomas DESCORSIERS - Fernando FONSECA -
Yuchun LIU - Ignacio PÉREZ - Maxime RETUREAU - Léo RONGIERES

Tuteur: Wilfrid PERRUQUETTI

Villeneuve d'Ascq - Novembre 2022

I - Introduction

Ce projet final de l'électif STA consiste en la réalisation d'un prototype de système à déplacement autonome. Le but de ce projet est de créer un robot qui explore son environnement et en construit une carte. L'idée est que le robot se déplace dans différentes directions, réalise des acquisitions instantanés de son environnement à l'aide de capteurs LiDAR et les combine dans une carte. Plus tard, la carte peut être utilisée pour donner des instructions au robot pour se déplacer vers un certain point dans l'espace.
Pendant la phase d'exploration, le robot doit être capable de détecter et d'éviter les obstacles.

Pour aboutir à un prototype fonctionnel, l'équipe de travail s'est réparti les tâches en décomposant le projet selon les sous parties suivantes :

- L'implémentation de ROS
- L'asservissement des deux moteurs
- L'estimation de la position du robot
- l'intégration du LiDAR
- L'algorithme de SLAM (*Simultaneous Locating And Mapping*)
- L'algorithme de *Path-planning* et de *Path-tracking*
- L'algorithme de navigation en exploration autonome

Ces sous parties qui jalonnent le projet de robot explorateur vont être détaillées individuellement par la suite dans ce rapport.

II- Cahier des charges fonctionnelles :

Fonctions principales du prototype :

1/ Se déplacer dans un environnement plat
2/ Cartographier
3/ Naviguer de manière autonome

II.1 - Se déplacer dans un environnement plat

- Se connecter au robot en réseau (ssh)
- Pouvoir commander un déplacement linéaire (avec asservissement en vitesse)
- Pouvoir commander un angle de braquage
- Pouvoir arrêter le robot de façon stable avec un asservissement en position

II.2 - Cartographier

- Le robot doit être munis d'un capteur capable de faire des acquisition de son environnement à haute fréquence (LiDAR)
- Le robot doit grâce à ce capteur être capable de dresser une carte fiable de l'environnement dans lequel il se déplace
- Le robot doit connaître sa position sur cette carte en temps réel (SLAM)

- Le robot doit définir quels sont les points les plus intéressants à explorer

II.3 - Naviguer de manière autonome

- Le robot doit être capable d'éviter les obstacles de son environnement
- Le robot doit être capable de réaliser des déplacements seul sans entrer en collision avec les éléments de son environnement
- Le robot doit être capable de cartographier son environnement de façon autonome

III - Schéma bloc fonctionnel

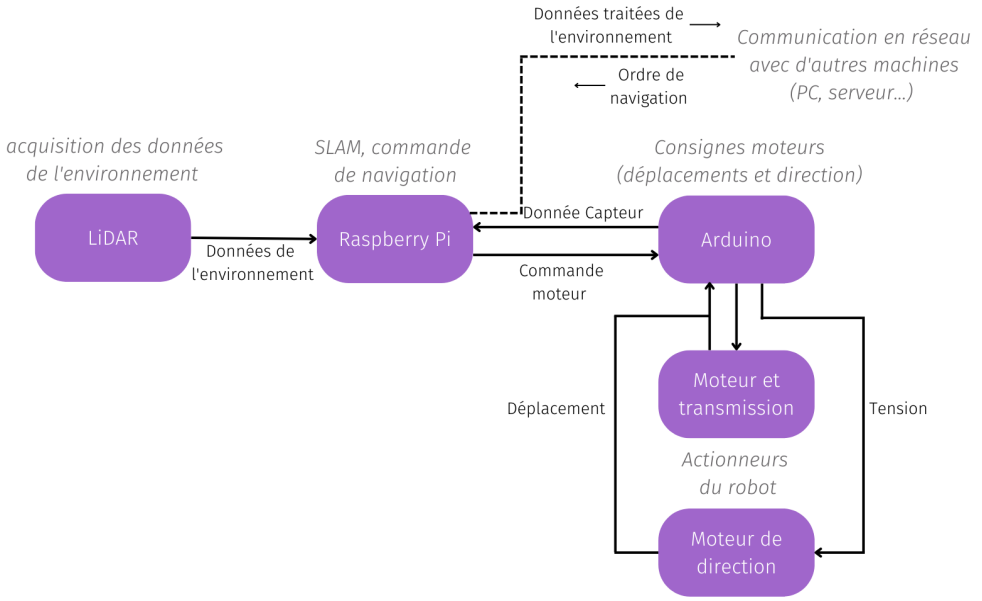


fig.0 : Schéma bloc fonctionnel du projet

IV - Schéma de mise en oeuvre

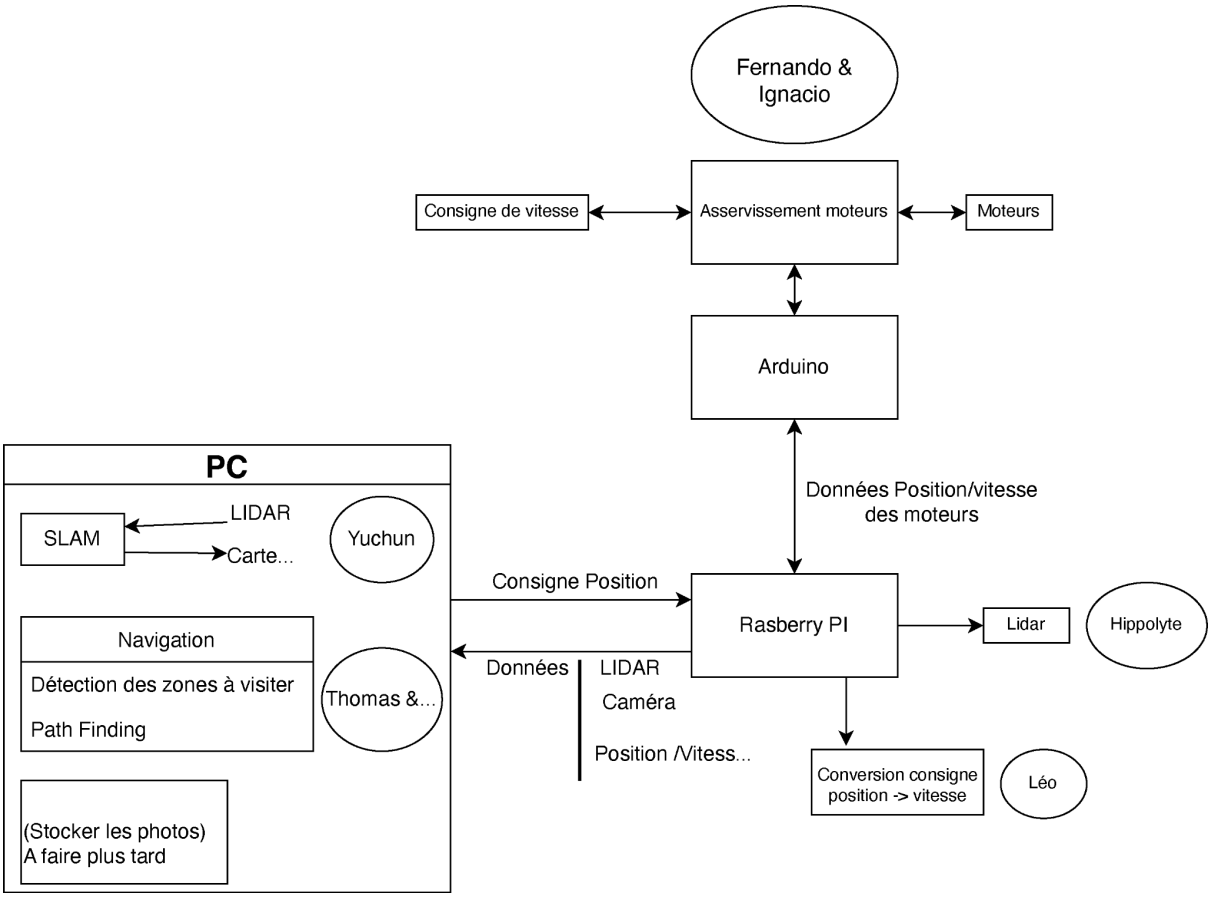


fig.1 : Schéma de mise en oeuvre du projet et rôle de chacun

V - ROS

Pour développer ce projet, nous avons décidé d'utiliser l'outil ROS (Robotics Operational System) qui est extrêmement connu et utilisé dans le contexte de la robotique. Avec ROS, il est possible d'exécuter différents codes (nœuds) qui sont exécutés en parallèle et aussi, si nécessaire, sur différentes machines, ces nœuds sont capables de communiquer via un système de " publishers " et " subscribers " de topiques. De cette façon, il est possible de développer l'ensemble du projet en parallèle, sans avoir besoin de créer un code principal qui gère l'ensemble du processus.

De plus, ROS étant un outil largement utilisé en robotique, la plupart des fonctions dont nous avons besoin ont déjà été développées et sont disponibles (par exemple gmapping, teb-planner, ros-serial, RRT), il suffit donc d'adapter ces paquets à notre type de robot.

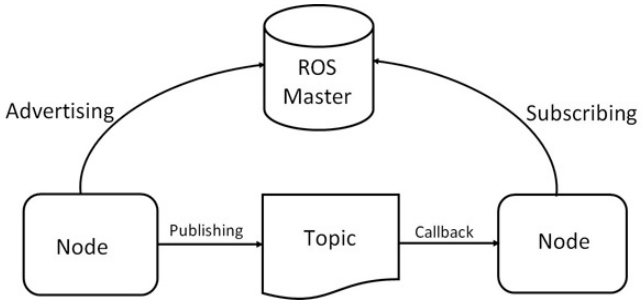


fig.1 : Schéma de la communication entre les nœuds (codes) avec ROS

Ainsi, comme nous disposons d'un Raspberry 3B+, qui supporte la version Ubuntu 20.04 et par conséquent, nous utilisons tout au long du projet la version Noetic de ROS. Enfin, les matériaux utilisés pour l'étude et l'apprentissage de cet outil figurent dans la XIV - Bibliographie.

VI - Asservissement des moteurs

Dans cette partie, on a commencé par estimer la dynamique de notre véhicule, donc on a fait une expérience en appliquant 5V pendant 1s sur les moteurs, puis on a pris les données expérimentales et estimé la fonction transfert en utilisant la fonction tfest() de matlab, qui fait le travail de trouver le modèle dynamique donné les données expérimentales. Avec cela, nous avons réglé 2 contrôles en utilisant la fonction pidTuner() : pour le moteur arrière (moteur linéaire) nous avons fait un contrôle de vitesse avec Tr = 0.5s et z = 0.7 et pour le moteur avant (direction) un contrôle de position avec Tr = 0.1s et z = 0.7. Rappelons que dans les deux cas, un dispositif anti-windup a été utilisé puisque la tension des moteurs ne peut pas dépasser 9V.

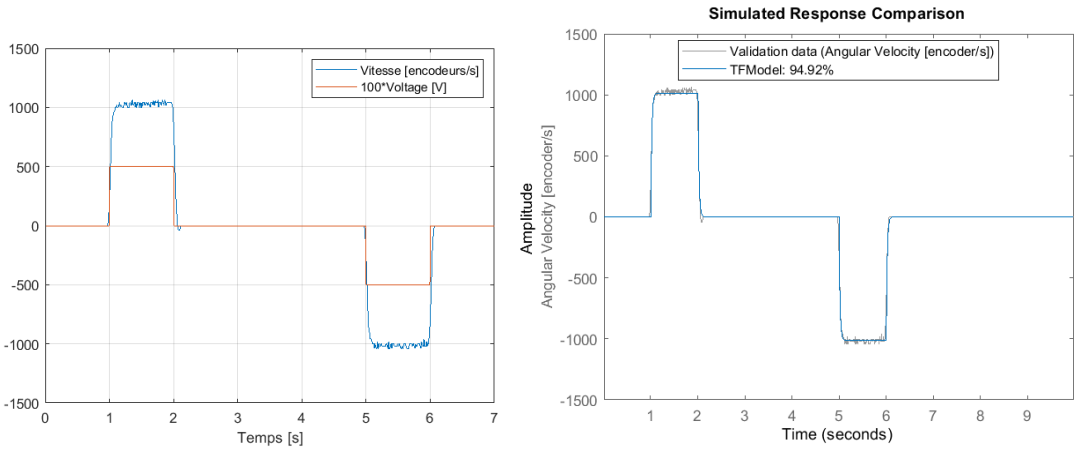


fig.2 : Résultat expérimental et résultat de l'estimation du modèle

Le contrôle théorique étant terminé, il est temps de le mettre en œuvre et de l'intégrer au ROS. Comme nous pouvons le voir dans le schéma ci-dessous, notre code passe par les étapes suivantes : nous recevons via une fonction de rappel la consigne pour les deux moteurs, puis nous obtenons la position et la vitesse estimées par les capteurs et effectuons le calcul PID, avec la sortie, nous appliquons la tension dans les moteurs qui les fera tourner, enfin, nous renvoyons au ROS la vitesse et les positions estimées par les capteurs, de manière à ce que l'estimation de la position du robot soit faite.

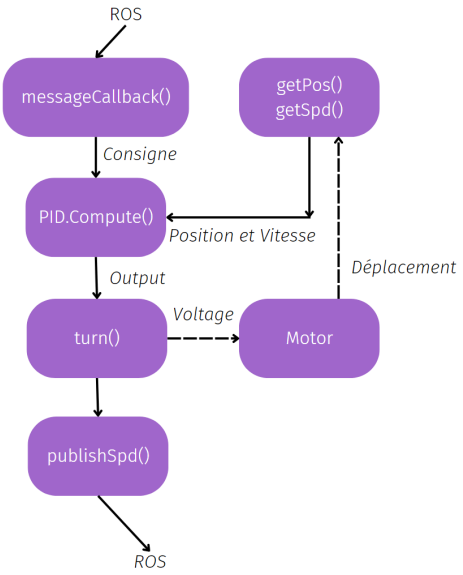


fig.3 : Diagramme du code dans l'Arduino

VII - Estimation de Position

Pour l'estimateur, nous avons fait un code simple qui reçoit via une fonction callback les données de l'arduino et calcule la position du robot via les équations suivantes :

$$\begin{cases} \omega_n = \frac{v_n}{l}\tan(\phi_n) \\ \Delta xy_n = v_n \Delta t \\ \Delta\theta_n = \omega_n \Delta t \\ \Delta x_n = \cos(\Delta\theta_n)\,\Delta xy_n \\ \Delta y_n = \sin(\Delta\theta_n)\,\Delta xy_n \\ \theta_n = \theta_{\{n-1\}} + \Delta\theta_n \\ x_n = x_{\{n-1\}} + \cos(\theta_n)\Delta x_n - \sin(\theta_n)\,\Delta y_n \\ y_n = y_{\{n-1\}} + \sin(\theta_n)\Delta x_n + \cos(\theta_n)\,\Delta y_n \end{cases}$$

éq.1 : Équations itératives pour le calcul de la position du robot

Où v est la vitesse linéaire, Φ est la position angulaire de la direction, l est la distance entre les axes des roues avant et arrière, ω est la vitesse angulaire du robot et (x,y,θ) la position du repère du robot dans le plan.

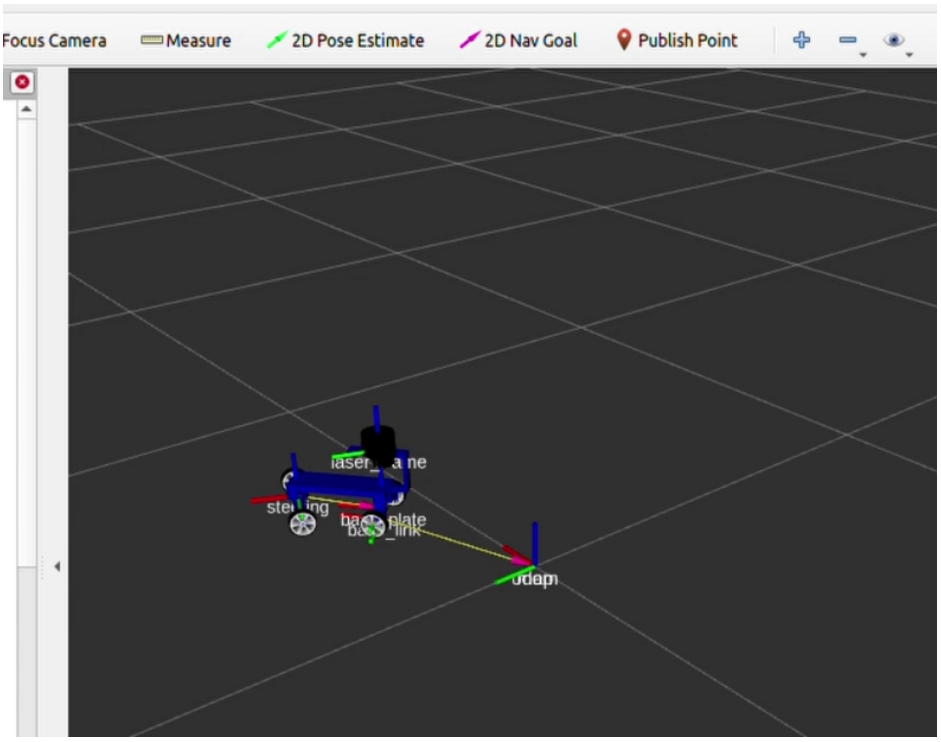


fig.4 : Visualisation du résultat de l'estimation de la position du robot avec Rviz

Il convient de mentionner que cet estimateur est loin d'être parfait, puisque nous n'avons utilisé que les données de l'odomètre, mais ce n'est pas un gros problème, puisque l'estimation réelle de la position sera faite par les données lidar, cette estimation est juste pour donner une "notion" de la position pendant que le nœud de cartographie fait l'estimation de la position qui est plus longue.

VIII - Intégration du LiDAR

Le but de la tâche est de permettre l'intégration d'un capteur LiDAR qui nous a été fourni dans le système pour permettre la détection d'obstacles et la cartographie de l'environnement du robot explorateur. Comme le choix a été fait d'utiliser ROS comme base de système d'exploitation du robot explorateur, l'intégration du LiDAR doit être réalisée dans cet environnement.

Le matériel fourni consiste en un LiDAR de modèle G4 rotatif à 360° de la marque YDLIDAR d'une portée théorique de 16 m.
La prise en main de ce LiDAR va se faire en deux temps.

La première piste explorée est une piste fournie par le constructeur (YDLIDAR) qui offre la possibilité d'installer un driver sur une machine et offre un outil de visualisation de l'environnement capturé par le LiDAR.

Cette première piste est concluante, on parvient bien à visualiser l'environnement capturé, les paramètres de fréquence d'acquisition ainsi que différents filtres peuvent être modifiés ou ajoutés. L'outil propose également une exploitation des données capturées grâce à un enregistrement et une possibilité d'exportation mais cette piste n'est pas compatible avec le projet car elle n'est donc pas en temps réel.

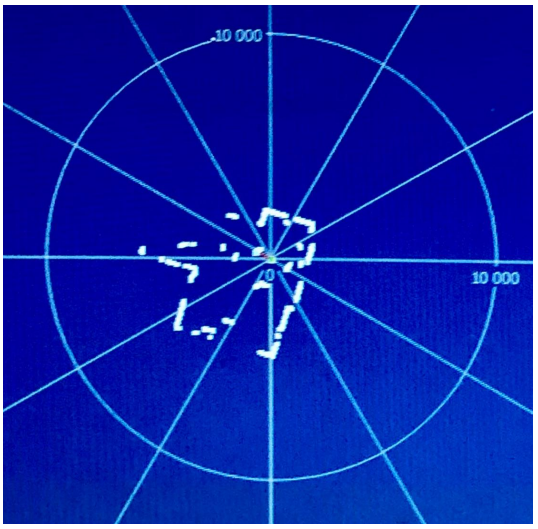


fig.5 : visualisation d'environnement par le LiDAR sur l'outil de visualisation de YDLIDAR

La deuxième piste, qui sera finalement exploitée dans le projet, est directement intégrée à l'environnement ROS. Une page github développée par YDLIDAR détaille les possibilités d'intégration de leur matériel dans ROS. (voir la bibliographie)

Cette méthode requiert finalement l'installation d'un nouveau driver adapté, le paquet ydlidar_ros_driver. Ce driver requiert l'utilisation d'une bibliothèque adaptée : YDLidar-SDK.

Une fois ce paquet et la bibliothèque installés, on peut simplement l'exécuter comme n'importe quel plugin sur ROS.

Dans ce paquet, YDLIDAR a également fourni un outil de visualisation, lidar_view. L'intégration de ce paquet à notre environnement ROS à permis la création d'un nouveau topic ROS appelé /scan qui permet de lancer un scan en continu de l'environnement par le LiDAR. Ce scan est visualisable grâce à l'outil View.

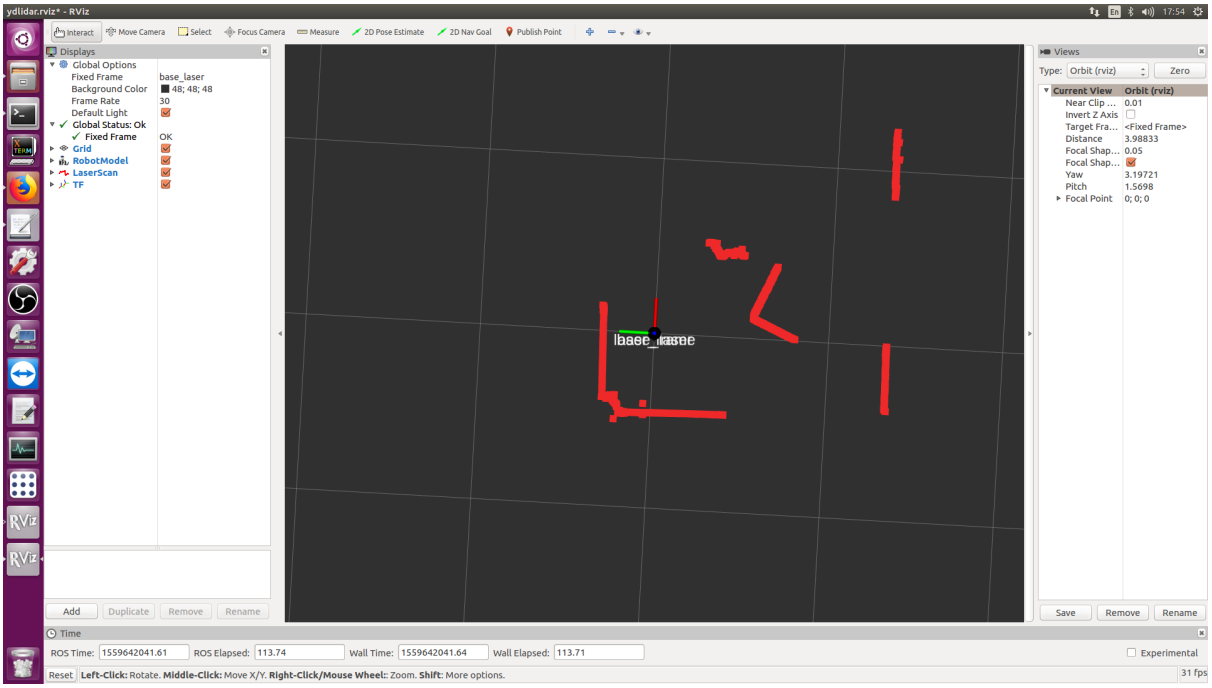


fig.6 : visualisation d'environnement par le LiDAR dans l'interface de ROS

Lors de la visualisation d'environnement, on observe un décalage angulaire entre le repère du LiDAR et le repère réel du robot. Un correctif de cet écart peut être apporté pour rendre les données capturées exploitables pour l'algorithme de SLAM. Dans le fichier de configuration du paquet, on peut modifier les paramètres de rotation de la base du robot vers le LiDAR (ici on ne modifie que z pour que les bases coïncident

IX - SLAM avec *Deeplearning*

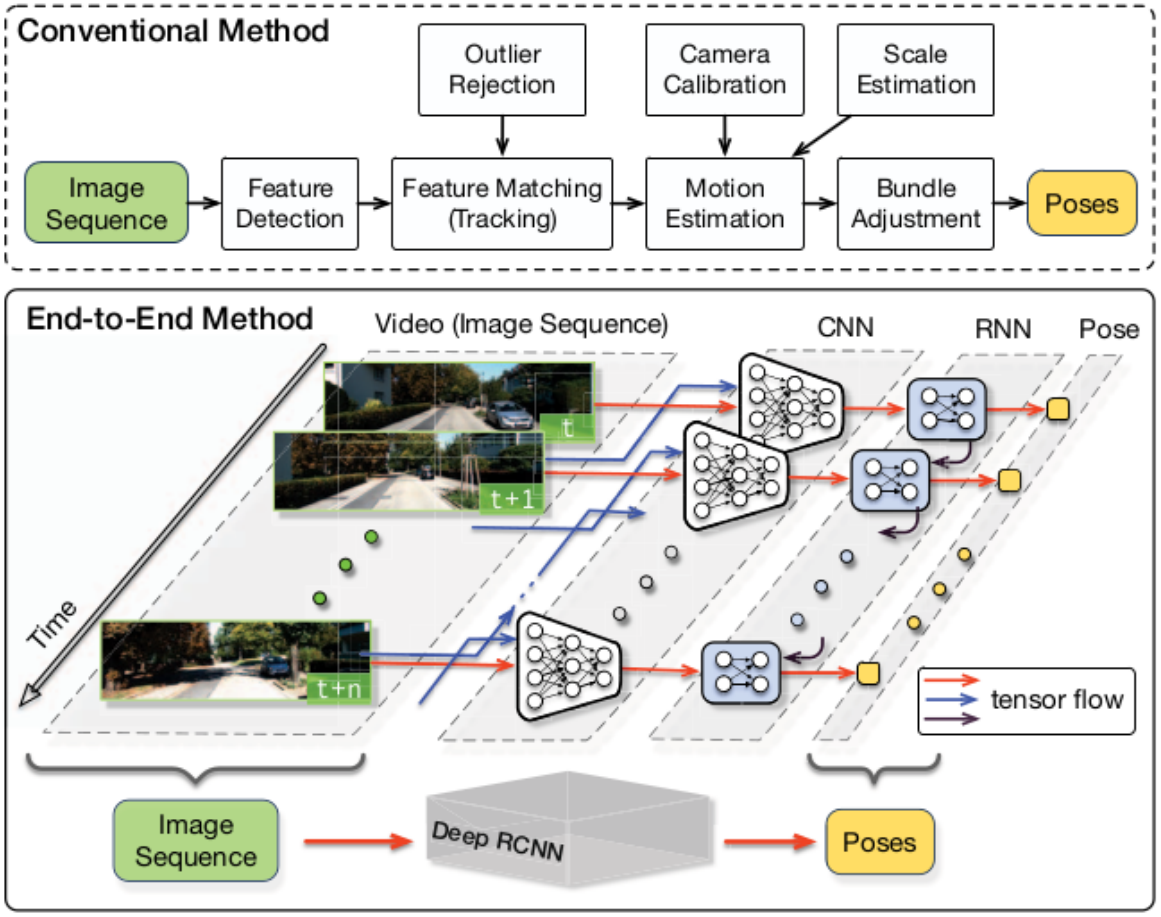
IX.1 - État du (SLAM)

SLAM=Simultaneous localization and mapping
La majorité des systèmes SLAM partagent plusieurs composants communs :

- Un détecteur de caractéristiques qui trouve un point d'intérêt dans l'image (caractéristiques)/
- Un descripteur de fonctionnalité qui correspond aux fonctionnalités de suivi d'une image à l'autre.
- Un backend d'optimisation qui utilise lesdites correspondances pour construire une géométrie de la scène (carte) et trouver la position du robot.
- Un algorithme de détection de fermeture de boucle qui reconnaît les zones précédemment visitées et ajoute des contraintes à la carte.

IX.2 - Contexte et introduction

- Visual odometry (VO) est une des techniques les plus essentielles pour l'estimation de la pose et la localisation du robot.
- Un algorithme VO devrait modéliser la dynamique du mouvement en examinant les changements et les connexions sur une séquence d'images plutôt que de traiter une seule image. Cela signifie que nous avons besoin d'un apprentissage séquentiel. RCNN est un bon choix.
- LSTM détermine quel précédent caché états à supprimer ou à conserver pour la mise à jour du etat.



Architectures de la VO monoculaire basée sur les caractéristiques conventionnelles et la méthode de bout en bout proposée. Dans la méthode proposée, RCNN prend une séquence d'images RGB (vidéo) en entrée et apprend les caractéristiques de CNN pour la modélisation séquentielle basée sur RNN afin d'estimer les poses.

IX.3 - Layers of Net

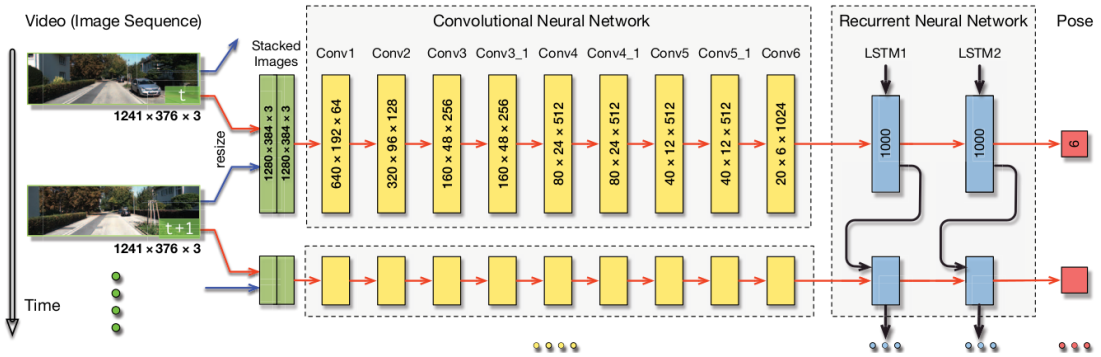


Fig. 2. Architecture of the proposed RCNN based monocular VO system. The dimensions of the tensors shown here are given as an example based on the image size of the KITTI dataset. The CNN ones should vary according to the size of the input image. Camera image credit: KITTI dataset.

Dans cette section, le cadre RCNN profond réalisant le VO monoculaire de bout en bout est décrit en détail. Il est principalement composé d'extraction de caractéristiques basées sur CNN et de modélisation séquentielle basée sur RNN.
L'architecture du système VO de bout en bout proposé de bout en bout est présentée à la figure 2. Il prend en entrée un clip vidéo ou une séquence séquence d'images monoculaire en entrée. A chaque pas de temps, l'image RVB est prétraitée en soustrayant les valeurs RVB moyennes de l'ensemble de de l'ensemble d'apprentissage et, éventuellement, en redimensionnant à une nouvelle taille dans un multiple de 64. Deux images consécutives sont empilées sont empilées pour former un tenseur permettant au RCNN profond d'apprendre à comment extraire les informations de mouvement et estimer les poses. De manière spéciale, le tenseur d'images est introduit dans le CNN pour produire une caractéristique caractéristique efficace pour la VO monoculaire, qui est ensuite passée par un RNN pour un apprentissage séquentiel. Chaque paire d'images donne une estimation de la pose à chaque pas de temps dans le réseau. Le système Le système VO se développe au fil du temps et estime de nouvelles images sont capturées.

IX.4 - LSTM

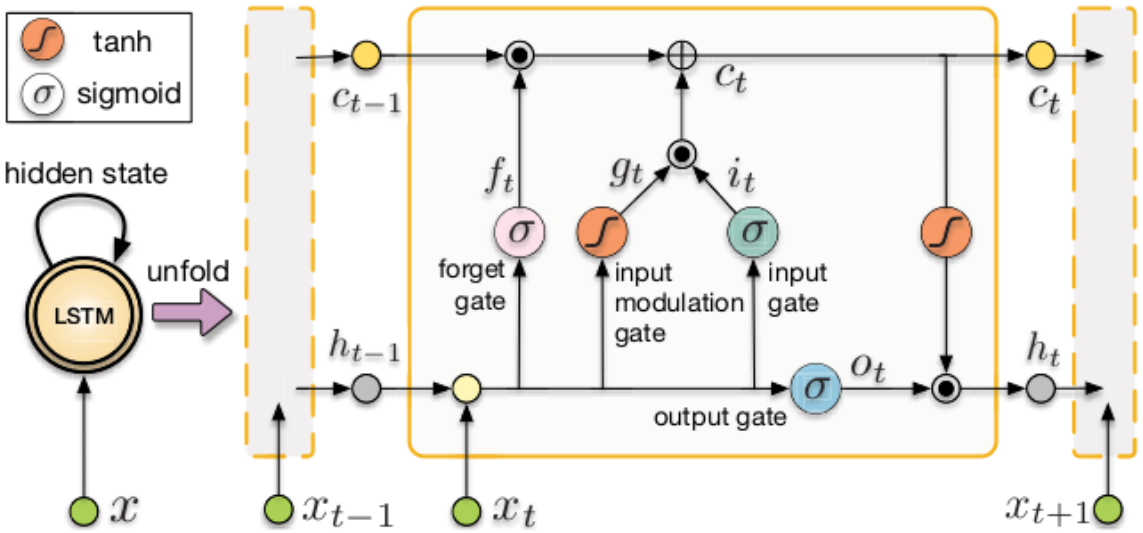


Fig. 3. Folded and unfolded LSTMs and internal structure of its unit. $\odot$ and $\oplus$ denote element-wise product and addition of two vectors, respectively.

0 < forget gate < 1

Afin de pouvoir findir et exploiter les corrélations entre les images prises dans des trajectoires longues, la mémoire à long terme (LSTM), qui est capable d'apprendre les dépendances à long terme en introduisant des portes et des unités de mémoire [26], est utilisée comme notre RNN. Elle détermine explicitement quels états cachés précédents doivent être rejetés ou conservés pour mettre à jour l'état actuel, étant censée apprendre le mouvement pendant l'estimation de la pose. La Fig. 3 montre le LSTM replié et sa version dépliée au fil du temps, ainsi que l'état interne du LSTM. temps sont illustrées à la figure 3, ainsi que la structure interne d'une unité LSTM. d'une unité LSTM.

IX.5 - Code

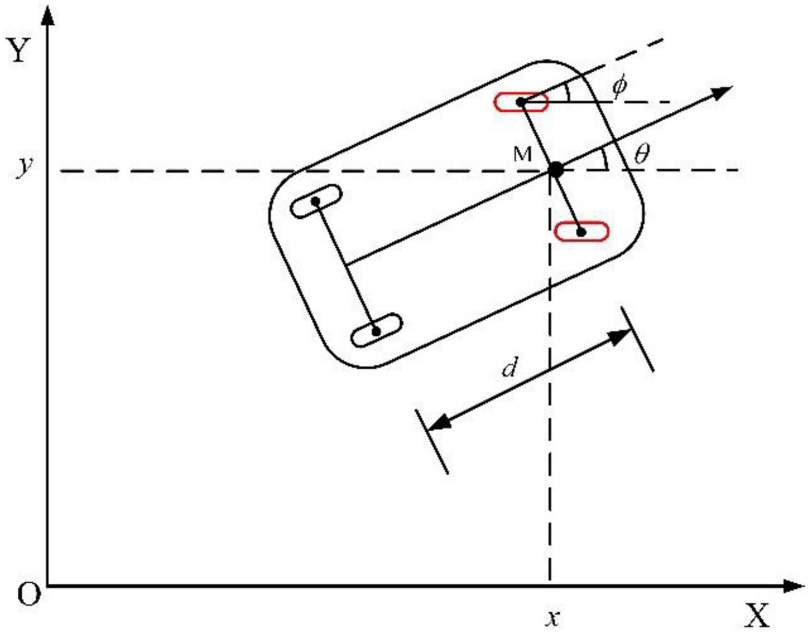
https://colab.research.google.com/drive/1Z8wg7M-vEDh1-ZWqALco8Bj3Oi1Hz5O_?hl=en#scrollTo=XS5Dvxr6fUei

IX.6 - Problèm des Images réelles et données Lidar

Les modèles existants sont basés sur de vraies images. Mais en mini-projet, on utilise Lidar pour collecter des informations environnementales. On doit convertir le type de données.

X - Path planning - Path tracking

Notre robot est un type voiture, c'est-à-dire qu'il a deux roues directrices à l'avant et deux roues motrices à l'arrière.

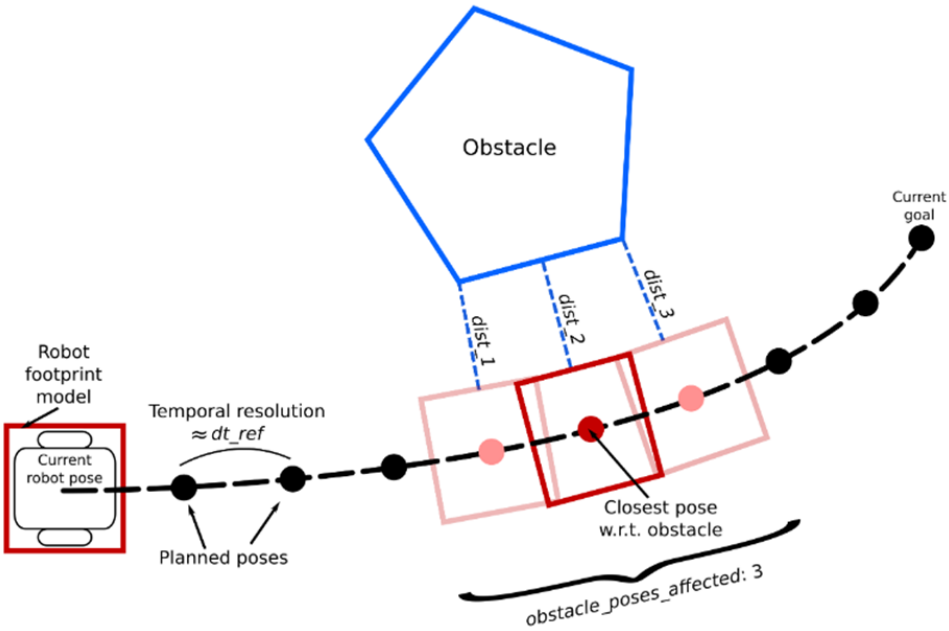


Notre robot n'allant pas très vite (quelques centimètres par seconde), on suppose alors que le modèle cinématique est suffisant pour le path planning et tracking. On a donc les équations:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos(\theta + \phi) \\ \sin(\theta + \phi) \\ \sin\phi/d \\ 0 \end{bmatrix} \nu + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \omega$$

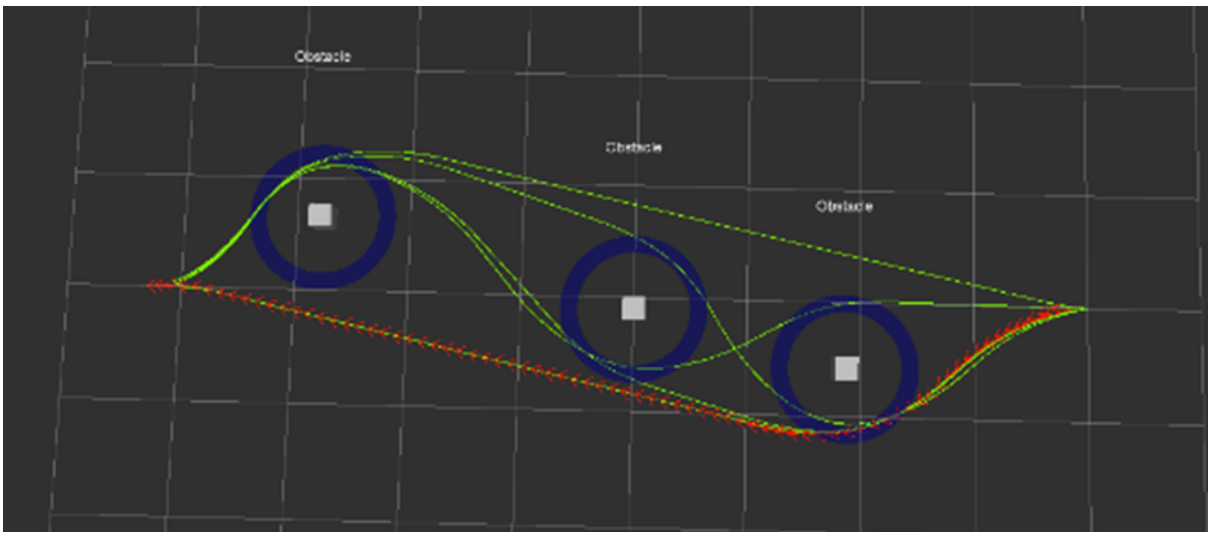
X.1 - Path planning

L'objectif de cette partie est de générer la trajectoire optimale que le robot doit emprunter pour atteindre un point précis en objectif. La trajectoire doit bien sûr prendre en compte la cinématique du robot et les obstacles détectés par le LIDAR qu'il doit éviter.



Nous avons adapté un nœud qui permet de générer cette trajectoire à partir des paramètres du robot (angle de braquage maximal, distance entre les deux trains, vitesse) et de la distance de sécurité fixée entre le robot et les obstacles. Le programme génère une liste de coordonnées, correspondants aux points que robot doit atteindre à chaque période dt_ref. Il quadrille la zone et teste toutes les trajectoires possibles (en respectant les contraintes du robot) dans l'objectif de trouver les plus rapides. Il utilise une fonction quadratic curve pour créer des courbes, en prenant en compte le rayon de braquage maximal du robot.

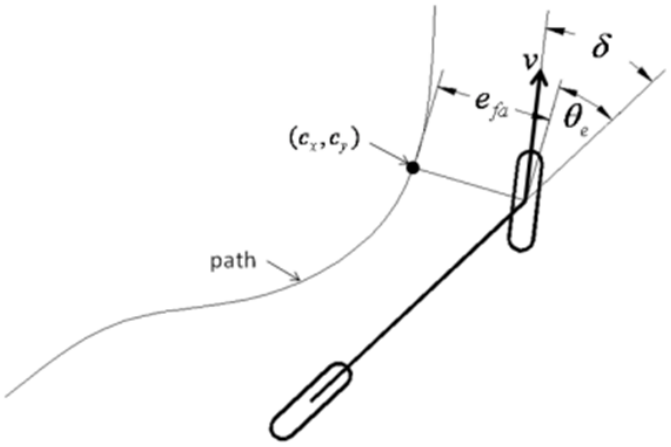
Sur rviz on simule le programme et affiche la trajectoire :



En vert, on a toutes les trajectoires possibles contournant les obstacles différemment. En rouge, c'est la trajectoire la plus rapide, c'est elle qui choisie.

X.2 - Path tracking

Une fois la trajectoire choisie, il faut que le robot suive cette dernière. C'est le rôle du path tracking.



Dans le cas parfait, il suffirait que les roues soient tangentes à la trajectoire pour que le robot la suivant. Or, dans la réalité, il y a des erreurs dues à plusieurs phénomènes (temps de réaction, patinage des roues,...). Le programme calcule l'angle **δ**, il est la somme de l'angle **θ** qui correspond à l'angle pour que les roues soient tangentes à la trajectoire, et de l'angle nécessaire pour rectifier la trajectoire du robot. Cet angle est calculé à partir de l'écart (efa) à l'instant t entre la position du robot estimée et le point de la trajectoire où il devrait être normalement. Cet angle dépend des coefficients du controller permettant d'influer sur des paramètres comme les oscillations, la vitesse que mettra le robot à revenir sur la trajectoire,...

Le test sur le robot a été concluant, la trajectoire est bien calculée, et le robot atteint l'objectif.

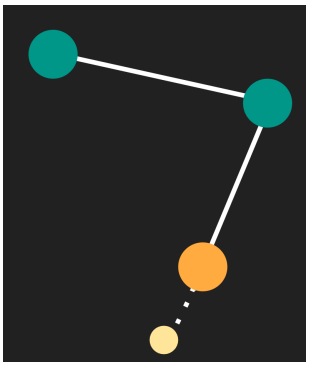
XI - Exploration

Le but de cette partie était à partir d'une carte donnée par le Lidar déterminer les points à explorer pour avoir le maximum d'informations sur l'environnement dans lequel évolue le robot.

XI.1 - Exploration de la carte

Tout d'abord ayant la carte brute avec la position du robot, nous devions explorer virtuellement la carte avant de pouvoir établir des points à explorer. Pour cela, nous avons étudié plusieurs algorithme de détermination de chemin optimal tel que Dijkstra, A ou A* pour au final choisir de prendre RRT (Rapidly-exploring random tree) en effet sachant que le chemin généré n'avait pas à être le plus optimisé possible car il sert juste à générer des points, nous avons privilégié la vitesse de calcul. Le principe de cet algorithme est qu'il va générer des points

aléatoires sur la carte et va essayer de les rejoindre avec le point le plus proche dans l'arbre déjà créé comme sur l'exemple suivant ou le nouveau point est en jaune et les points existants en bleu.

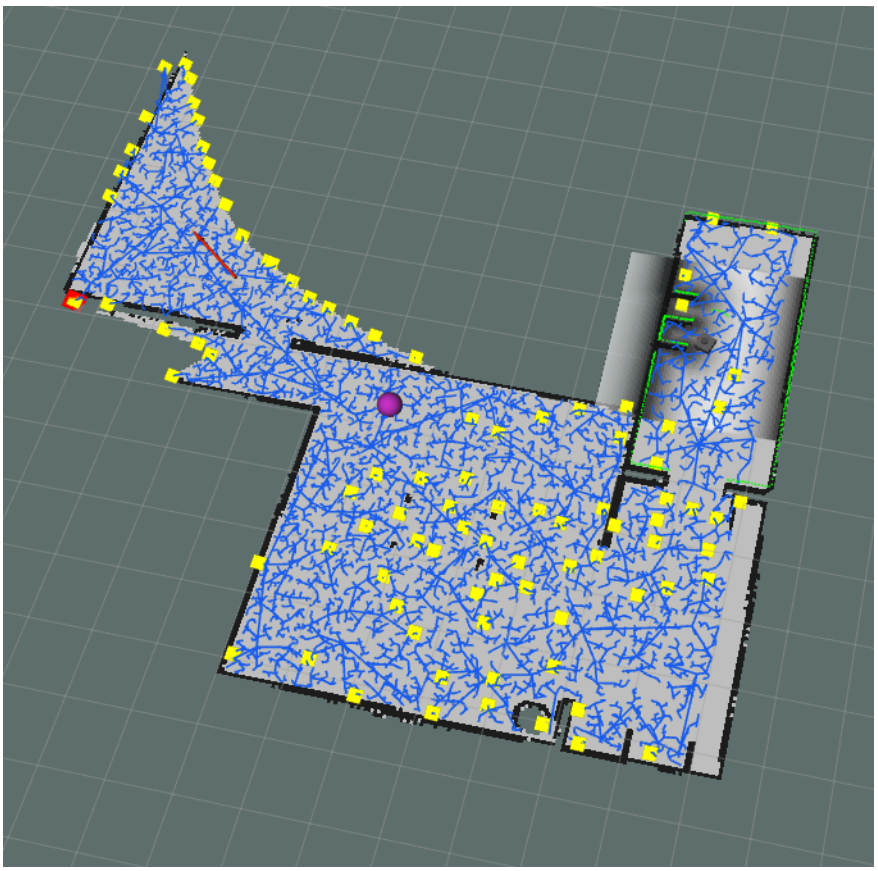


XI.2 - Filtrage des points

Ensuite avec les points qu'on a obtenu avec cet algorithme on va isoler les points qui sont sur la frontière des zones déjà explorées et éliminer tous les autres points sur des zones déjà connues. On va ensuite regrouper les points qui sont proches pour éviter d'avoir émis deux points à visiter, sachant que en explorant le premier la zone autour du deuxième point sera révélée.

XI.3 - Détermination des points à explorer

Pour déterminer les points qui seraient les plus intéressants à explorer, on va attribuer un score à chaque point. Ce score se calcule avec le coût du point qui correspond à la distance avec le robot et le revenu de l'exploration de ce point c'est-à-dire la quantité d'information qu'on peut obtenir en explorant ce point. Ensuite, on va donc classer ces points et attribuer au robot le point le plus efficace c'est-à-dire avec le meilleur score qui va ensuite utiliser le module de navigation pour calculer sa trajectoire.



XIII - Annexe

1. Project's main repository: https://github.com/FernandoBFonseca/STA_robot_explorateur
2. ROS Repository: https://github.com/FernandoBFonseca/STA_robot_explorateur_ros

XIV - Bibliographie

1. Sujet Robot Explorateur - https://moodle2223.centralelille.fr/pluginfile.php/20363/mod_folder/content/0/Sujet_RobotExplorateur.pdf
2. ROS: Robot Programing - https://community.robotsource.org/t/download-the-ros-robot-programming-book-for-free/51
3. Formation control for car-like mobile robots using front-wheel driving and steering - https://www.researchgate.net/publication/325421132_Formation_control_for_car-like_mobile_robots_using_front-wheel_driving_and_steering
4. LiDAR implementation in ROS - https://www.daslhub.org/unlv/wiki/doku.php?id=using_ros_to_read_data_from_a_hokuyo_scanning_laser_rangefinder https://github.com/YDLIDAR/ydlidar_ros_driver https://github.com/YDLIDAR/YDLidar-SDK
5. Autonomous Slam - https://github.com/fazildgr8/ros_autonomous_slam
6. Nox Robot - https://github.com/RBinsonB/Nox_robot/tree/master/nox
7. Autonomous Mobile Robot - https://github.com/bandasaikrishna/Autonomous_Mobile_Robot
8. Planning for car-like robots - http://wiki.ros.org/teb_local_planner/Tutorials/Planning%20for%20car-like%20robots