Project SY15: Traversée d'un labyrinthe

11 avril 2024

1 Présentation du sujet

Cette année, l'objectif du projet de SY15 sera de commander un Turtlebot (figure 1) pour traverser un labyrinthe et atteindre une position cible donnée précédemment. Il faudra ensuite, de manière autonome, aller se garer à une bonne place inconnue à l'avance mais matérialisée par une cible réfléchissante (docking). On entend que par "autonome", aucune intervention d'un opérateur sera autorisée entre le début et la fin de la mission. Un affichage vue d'oiseau sur le PC de contrôle permettant de suivre l'évolution du robot pendant la mission sera nécessaire.

Les turtlebots utilisent ROS (Robot Operating System) pour fonctionner. Il s'agit d'un framework comprenant un ensemble de programmes et de bibliothèques (distribution) qui permettent d'abstraire le matériel, et en particulier les communications inter-processus. Dans ROS, plusieurs nœuds fonctionnent en parallèle, sur des processus différents. Ils communiquent entre eux via des sockets système, en utilisant une abstraction des messages permettant de facilement envoyer des structures complexes entre deux nœuds. Ces nœuds peuvent être écrits en C++ ou en Python nativement, mais des adaptations ont été faites, notamment pour pouvoir être utilisés avec MATLAB Simulink. Les ressources fournies pour ce mini-projet seront en Python, mais vous êtes libre du langage utilisé dans la réalisation du projet.

2 Prise en main

2.1 Connexion au robot

Tâche 2.1.1

Afin de pouvoir communiquer avec le robot et lui envoyer des ordres, connectez-vous à son wifi. Le robot émet un réseau wifi qui porte son nom (ex : michelangelo).

Tâche 2.1.2

Définissez l'adresse et le port du roscore tournant sur le robot :

```
export ROS_MASTER_URI=http://192.168.1.1:11311
```

Tâche 2.1.3

Définissez votre propre IP : récupérer l'ip de l'interface wifi via `ip a` :

```
export ROS_IP=192.168.1.104
```

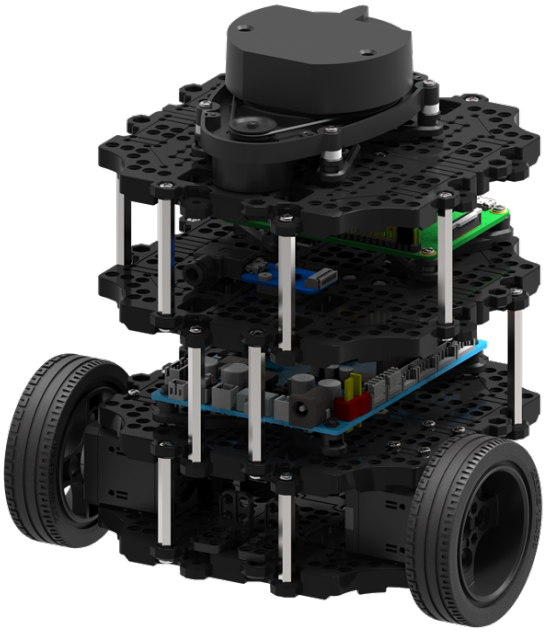

FIGURE 1 – Turtlebot3 "burger"

Tâche 2.1.4

Définissez le modèle du turtlebot (burger) :

```
export TURTLEBOT3_MODEL=burger
```

Ces commandes sont à faire à chaque fois que vous ouvrirez un terminal, vous pouvez donc les ajouter dans le fichier `.bashrc` (ou équivalent).

Note

S'il y a un problème de connexion, essayez un ping sur le robot :

```
ping 192.168.1.1
```

2.2 Installation et lancement de ROS

Pour installer ROS sur Ubuntu 20, vous pouvez suivre le tutoriel suivant : `https://wiki.ros.org/noetic/Installation/Ubuntu`

Si vous n'avez pas d'Ubuntu installé, vous pouvez soit installer ubuntu dans une VM/Conteneur en utilisant par exemple VirtualBox/Distrobox, soit utiliser WSL2 : `https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-11-with-gui-support#1-overview`. Si vous êtes sur Windows 11, WSL2 fonctionne normalement très bien pour les applications graphiques. Si vous êtes sous Windows 10, vous risquez d'avoir des problèmes, et il sera alors plus simple d'utiliser une VM.

Attention

Nous utilisons ROS noetic, qui est disponible que sous Ubuntu 20. N'installez pas Ubuntu 22 ! Si vous avez Ubuntu 18, vous pouvez installer ROS melodic.

Tâche 2.2.1

Vous pouvez créer un nouveau workspace :

```
mkdir -p workspace/src && cd workspace
catkin_init_workspace
```

puis décompressez l'archive présente dans moodle dans le dossier `src`.

Tâche 2.2.2

Compilez le workspace :

```
catkin_make
```

Tâche 2.2.3

Ajoutez dans votre `.bashrc` (ou équivalent) les lignes suivantes :

```
source /opt/ros/noetic/setup.bash
source /chemin/vers/votre/workspace/devel/setup.bash
```

2.3 Visualisation des données

La visualisation des données sous ROS se fait notamment grâce à RViz.

Tâche 2.3.1

Lancez RViz dans un terminal en tapant la commande `rviz`.

Vous pouvez ajouter des topics ROS via le bouton "Add", en b as à gauche. Allez dans l'onglet "By topic" pour sélectionner le topic à visualiser.

Vous pouvez sauvegarder votre configuration via **File** > **Save Config As**. Pour l'ouvrir, tapez la commande `rviz -d [chemin vers votre config]`.

Tâche 2.3.2

Affichez les données lidar en ajoutant le topic `/scan`.

2.4 Contrôle manuel du robot

Nous allons envoyer des commandes à votre robot. Il existe pour cela le package ros 'turtlebot3_teleop' qui permet de contrôler le robot via les touches du clavier.

Tâche 2.4.1

Dans un terminal, lancez

```
rosrun turtlebot3_teleop turtlebot3_teleop_key
```

puis contrôlez le robot.

Attention, les touches sont configurées en QWERTY.

2.5 Gazebo : Un simulateur sous ROS

Gazebo est un simulateur fonctionnant sous ROS, avec un environnement 3D. Un module pour fonctionner avec les turtlebots existe, via le package ROS 'turtlebot3_gazebo'.

Un fichier ROS launch est présent dans l'archive téléchargée précédemment.

Tâche 2.5.1

Lancez Gazebo en utilisant le fichier launch suivant : SY15/launch/gazebo_simulation.launch

```
roslaunch SY15 gazebo_simulation.launch
```

Une fois Gazebo lancé, une fenêtre permettant de visualiser l'environnement s'ouvrira, et un labyrinthe sera présenté devant le robot. Dans ROS, les topics suivants ont été créés :

- `/scan` publie les scans lidar pour être exploité.
- `/cmd_vel` qui écoute les commandes à envoyer au robot pour le commander.
- `/odom` publie les données d'odométrie du robot, sous la forme d'une vitesse linéaire et d'une vitesse angulaire.
- `/imu` publie les données inertielles du robot, sous la forme d'une orientation (via les magnétomètres), d'une accélération linéaire et d'une vitesse angulaire.

Note

Par défaut, le simulateur Gazebo est lancé à une vitesse de 0.1x pour ménager l'ordinateur. Vous pouvez changer cette valeur dans le fichier launch `src/SY15/launch/gazebo_simulation.launch`, en changeant le paramètre `use_sim_time` à la ligne 10.

2.6 Estimation d'état

Le mini-projet vous demandera très certainement de pouvoir estimer la position et l'orientation du robot afin de rejoindre une cible. Pour cela, le filtre de Kalman étendu (EKF, version non linéaire du filtre de Kalman) est généralement utilisé. Le principe de l'EKF est de linéariser les modèles de prédiction et de correction autour de l'estimation actuelle.

Quelques informations concernant les turtlebots que vous utiliserez :

- Tout d'abord, ces robots se comportent comme des chars : ils ont 2 roues commandées, et il est possible de commander la vitesse de ces deux roues indépendamment. Ainsi, pour tourner, la roue extérieure au virage doit aller plus vite. Il est possible de faire un demi-tour sur place lorsque les roues tournent à des vitesses opposées.
- Les commandes envoyées, via le topic `/cmd_vel` sont sous la forme d'une vitesse longitudinale et d'une vitesse de rotation, c'est le robot qui transforme ces vitesses en vitesse de roues.
- Les robots mesurent des données odométriques, qui sous deux formes :
  - L'IMU (Inertial Measurement Unit) qui fourni l'accélération linéaire sous forme d'un vecteur en 3D dans le repère du robot, et la vitesse angulaire, qu'on ne pourra considérer que sur l'axe Z (le lacet).
  - Les données de vitesse, calculées à partir de mesure des vitesses des roues, sous la forme d'une vitesse longitudinale et d'une vitesse angulaire.

2.7 Définition des modèles

Tâche 2.7.1

Définissez le **modèle de prédiction** du turtlebot.

- De quelles variables est composé l'état du système ?
- Les vitesses longitudinale et angulaire doivent-elles être dans l'état du véhicule ?
- Quelle dérivée de la position considérez-vous nulle ?
- Donnez les équations d'évolution de votre état, en temps discret.

Tâche 2.7.2

Définissez le **modèle de correction** du turtlebot. Vous pourrez utiliser les données issues du robot : odométrie (vitesse longitudinale v et vitesse angulaire ω_{odom}) et/ou IMU (accélération linéaire a et vitesse angulaire ω_{IMU}).

- Quelle donnée issue du robot est directement compatible avec votre état, sans dériver ni intégrer ? (Si vous n'en trouvez aucune, il faudra peut-être revoir votre état)
- Donnez les équations du modèle de correction, permettant de prédire l'observation en fonction de l'état.

2.8 Linéarisation autour de l'estimation

Maintenant que vous avez défini les modèles de prédiction et de correction, vous constatez qu'il y a des équations non-linéaires dans vos modèles, il faut donc linéariser les au point estimé. Pour cela, l'EKF utilise les matrices Jacobiennes des modèles (dérivées partielles).

L'étape de prédiction se fait comme suit :

$$X_{k+1|k} = f(X_{k|k}) \tag{1}$$

$$P_{k+1|k} = F_k P_{k|k} F_k^T + Q \tag{2}$$

où :

- f est le modèle de prédiction, qui dépend de l'estimation issue de la correction $X_{k|k}$,
- F_k est la Jacobienne de f suivant X, au point $X_{k|k}$,
- Q est la matrice de covariance du bruit de modèle, très importante lorsqu'on linéarise.

Note

Une Jacobienne est une matrice contenant les dérivées partielles premières des différentes équations. Dans le cas de F, la Jacobienne de f suivant la variable vectorielle X, cela donne :

$$F = \begin{bmatrix} \frac{\partial f_1}{\partial X_1} & \cdots & \frac{\partial f_1}{\partial X_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial X_1} & \cdots & \frac{\partial f_n}{\partial X_n} \end{bmatrix}_{X_{k|k}} \tag{3}$$

Pour information, la matrice contenant les dérivées secondes est appelée Hessienne, mais nous n'en auront pas besoin ici.

Tâche 2.8.1

Donnez la Jacobienne F en fonction du modèle de prédiction défini précédemment.

Normalement, votre étape de correction n'implique que des équations linéaires, vous pouvez donc utiliser les équations du filtre de Kalman linéaire :

$$X_{k|k} = X_{k|k-1} + K_k(Z - CX_{k|k-1}) \tag{4}$$
$$P_{k|k} = (I - K_k C)P_{k|k-1} \tag{5}$$
$$K_k = P_{k|k-1}C^T(\underbrace{CP_{k|k-1}C^T + R_k}_{S_k})^{-1} \tag{6}$$

où :

- Z est le vecteur d'observation,
- C est la matrice représentant le modèle de correction linéaire,
- K est le gain de Kalman,
- R est la matrice de covariance de l'observation (à régler).

Tâche 2.8.2

Donnez la matrice C en fonction du modèle de correction défini précédemment.

2.9 Implémentation

Tâche 2.9.1

Complétez le nœud présent dans le fichier `src/SY15/scripts/state_estimation.py` avec les équations déterminées aux questions précédentes.

Vous pouvez visualiser votre estimation dans RViz en ajoutant le topic `estimation`.

3 Le mini-projet

3.1 Objectif 1 : Atteindre la position demandée

Le premier objectif du projet est d'atteindre une position demandée, exprimée de manière relative à la position de départ du robot. Cependant, entre la position de départ et la position d'arrivée, des obstacles se présenteront. Lors de la démonstration finale, il faudra traverser un labyrinthe. Le labyrinthe aura forcément un chemin valide.

Attention, il est interdit de toucher les murs avec le robot !

Vous êtes libre de la stratégie à adopter, mais vous ne connaîtrez pas la disposition du labyrinthe en avance. La vitesse d'exécution de l'objectif sera prise en compte.

3.2 Objectif 2 : Positionnement relatif à une cible (docking)

Le deuxième objectif consiste à aller garer le robot immédiatement après avoir atteint la cible de l'objectif 1, à une place inconnue à l'avance, mais signalée par un panneau très réfléchissant. Ce panneau sera très facilement détectable au lidar. Il faudra positionner le robot sur la normale du panneau, à 20 cm. Le robot devra être orienté vers le panneau.

3.3 Livrable

Attention

Le projet se décompose en deux objectifs. Essayez d'avancer sur les deux objectifs en même temps plutôt que de vous concentrer sur un seul.

Un petit rapport est attendu à la fin du projet (à remettre dans le Moodle du cours), ainsi qu'une démonstration avec une vidéo (faite avec votre smartphone à remettre aussi dans le Moodle du cours).

Dans le rapport, vous tâcherez de détailler les méthodes utilisées pour chaque tâche. N'hésitez pas à aussi décrire des méthodes qui n'ont pas fonctionné, si vous les avez testées, en expliquant les problèmes rencontrés. **Les réponses aux questions posées dans la partie 2.6 sont à inclure dans votre rapport.**