ROB201 - Introduction à la robotique mobile

David Filliat - Elena Ivanova - Thibault Toralba
ENSTA Paris

8 mars 2023

Introduction

Ce document indique la trame générale prévue pour le développement du projet dans le cadre du cours ROB201. Les travaux prévus dans chaque séance sont indicatifs, il peuvent être améliorés si les travaux proposés sont terminés avant la fin, mais chaque séance doit être finie avant de passer à la suivante.

1 Séance 01 : Installation et prise en main

Nous vous conseillons d'utiliser l'environnement de développement Visual Studio Code (mais un autre est possible si vous êtes à l'aise avec). Installez l'extension Python qui fournit des outils utiles pour le développement python.

1.1 Installation du code de base

Commencez par créer votre propre copie du dépot github :

https://github.com/emmanuel-battesti/ensta-rob201

Pour cela vous devez posséder votre propre compte GitHub (créez le si besoin), puis, sur la page du dépot, sur le bouton `Fork`, sélectionnez `Create a new Fork`.

Clonez ensuite sur votre ordinateur le squelette de code que vous venez de dupliquer. Dans un terminal, tapez :

```
> git clone git@github.com:votre_login_github/ensta_rob201.git
```

Vous pouvez ensuite commencer à travailler sur le code téléchargé. Pendant toute la durée du cours, pensez à faire des `commit` fréquents, avec un message court et informatif. Pensez à faire des push réguliers vers votre déport github (à chaque fin de séance par exemple).

Suivez ensuite la procédure d'installation décrite dans le fichier `INSTALL.md`. Vous devrez adapter les commandes en fonction de votre version de python 3 (si ce n'est pas la 3.8). En particulier, ne sautez pas la phase de création de l'environnement virtuel et pensez bien à l'activer.

Une fois l'installation terminée, dans VSCode, ouvrez le répertoire `ensta_rob201`. Sélectionnez ensuite l'interpréteur python de l'environnement virtuel que vous venez de créer (avec `(Ctrl+Shift+P) Python: Select Interpreter`). Vous pouvez lancer le projet en exécutant le fichier `main.py`.
La fenêtre de simulation doit apparaître (Figure 1), et il ne se passe rien d'autre, c'est normal, la fonction de contrôle de votre robot est vide. Pour quitter la simulation, tapez `q` dans la fenêtre de la simulation.

1.2 Linter

Pour utilisez le linter, il faut le configurer en tapant `(Ctrl+Shift+P)` puis rechercher la commande `Python: Select Linter`. Sélectionnez `pylint`. A chaque sauvegarde de fichier python, le résultat de l'analyse apparaît dans l'onglet `Problems` (par défaut sous le code).

Pour vous entraîner, corrigez les problèmes relevés dans le fichier `main.y`.

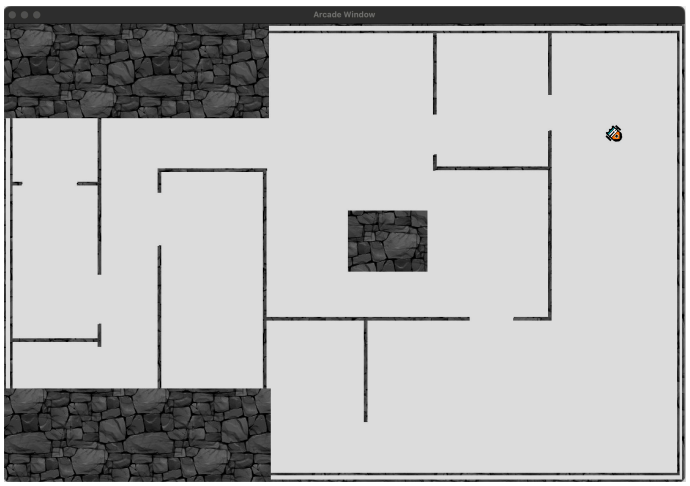

FIGURE 1 – Fenêtre du simulateur Place-bot

1.3 Profiler

Pour utiliser le profiler, utilisez d'abord le programme `cProfile` pour lancer votre script et enregistrer un fichier de statistiques :

```
python3 -m cProfile -o mon_script.prof mon_script.py
```

Utilisez ensuite l'interface graphique `snakeviz` pour visualiser plus facilement les résultats :

```
> python3 -m pip install snakeviz
> python3 -m snakeviz mon_script.prof
```

Au travers de l'interface graphique, vous pouvez alors déterminer les fonctions prenant le plus de temps d'exécution et tenter de les optimiser.

Utilisez le profiler sur le fichier `main.py` fourni (tapez `q` dans la fenêtre de la simulation pour l'arrêter au bout de quelques secondes), déterminez les fonctions les plus chronophages. Parmis celles-ci, certaines sont dans les bibliothèques utilisée et ne peuvent pas être modifiées. Déterminez celles auxquelles vous avez accès et tentez de les optimiser.

1.4 Prise en main du simulateur

Nous utilisons dans ce cours un simulateur simple développé à l'ENSTA : Place-bot (il a déjà été installé avec le code du cours, il n'est pas utile de le réinstaller vous-même).

Pour utiliser ce simulateur, il faut créer :

— une classe qui dérive de la classe `RobotAbstract` pour définir votre robot, en particulier une fonction `control(self)` qui va donner les commandes à votre robot en fonction des données capteurs,

— une classe qui dérive de la classe `WorldAbstract` pour définir l'environnement de simulation avec votre robot en paramètre,

— un simulateur dérivant de la classe `Simulator` avec votre monde en paramètre.

L'exécution de la fonction `Simulator.run()` va ensuite simuler le fonctionnement de votre robot avec sa fonction de contrôle. Il est également possible de contrôler le robot avec le clavier en passant l'argument `use_keyboard=True` au simulateur.

L'ensemble de ces étapes a été fait dans le fichier `main.py`, avec les classes définies dans les fichiers `my_robot_slam.py` et `worlds/my_world.py`.

Programmez ensuite un premier comportement pour le robot. Pour cela, adaptez la fonction `control(self)` dans la classe `MyRobotSlam` afin de faire un évitement d'obstacles simple. Pour une bonne organisation du code, vous pouvez mettre votre code dans une ou plusieurs fonction dans le fichier `control.py`, et importer les fonctions utilse dans le fichier `my_robot_slam.py`

Vous pouvez accéder aux données :

— du **télémètre laser** grâce à la fonction `self.lidar()` qui renvoie un object de type `Lidar`. Cet objet a une méthode `get_sensor_values()` qui renvoie les distances mesurées par le

télémètre laser sous forme d'un `array` Numpy, et une méthode `get_ray_angles()` qui renvoie la direction (angle en radians, sens trigonométrique) de chaque rayon du télémètre laser par rapport à l'avant du robot. Le champ de vision du télémètre est de 2π.

— de **l'odométrie** avec la fonction `self.odometer_values()` qui renvoie un `array` avec la position $[x, y, \theta]$ estimée dans le repère odométrie initialisé à $[0, 0, 0]$ au lancement de la simulation.

Votre fonction doit renvoyer une commande de vitesses de translation et de rotation, dans l'intervalle $[-1, 1]$, sous forme d'un dictionnaire python :

```
command = {"forward": 0,
           "rotation": 0}
```

Implémentez un comportement qui avance en ligne droite quand il n'y a pas d'obstacles devant le robot, et qui tourne d'un angle aléatoire quand un obstacle est présent.

1.5 Extensions possibles

Vous pouvez ensuite améliorer le comportement d'évitement d'obstacles, par exemple en gérant un historique des directions de rotation pour éviter de tourner toujours du même coté, ou en choisissant la direction de rotation en fonction de l'angle de l'obstacle (pour éviter de tourner face à un mur). Vous pouvez également essayer de réaliser un algorithme de suivi de mur, par exemple en vous inspirant du TP 'Wall Follow' de F1TENTH[1]. Essayez de faire en sorte que le robot explore tout l'environnement.

Vous pouvez également définir un nouvel environnement de simulation en remplacement de celui fournit dans le fichier `worlds/my_world.py`.

1. Disponible sur https://f1tenth-coursekit.readthedocs.io/en/latest/assignments/labs/lab3.html