Projet ROS 2 : Contrôle de robot mobile (Wheeled Robot)

Auteur

GITHUB ROLLN7DRKTAYAU

LINKEDIN RCT

Ce projet contient différents comportements autonomes pour un robot mobile à roues simulé dans ROS 2. Chaque comportement est implémenté comme un nœud ROS 2 en C++. Le robot utilise notamment des capteurs tels que le laser pour percevoir son environnement.

Arborescence

```
.
├── exploring-ros2-with-wheeled-robot
│   ├── CMakeLists.txt
│   ├── launch/
│   ├── package.xml
│   ├── src/
│   └── worlds/
├── readme.md
└── tp_ros_etudiant_v2.pdf
```

Pré-requis

- ROS 2 humble
- colcon
- Un environnement de simulation (comme Gazebo et TurtleSim)

Mise en place

```
# Dans le répertoire Home/TD_ws
mkdir -p ~/TD_ws/src
cd ~/TD_ws/src
git clone https://bitbucket.org/theconstructcore/exploring-ros2-with-wheeled-
robot.git
cd ..
colcon build --symlink-install --packages-select my_package

# Configurer TurtleBot3 et Gazebo
source /opt/ros/humble/setup.bash
export TURTLEBOT3_MODEL=burger
```

```
export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:$(ros2 pkg prefix
turtlebot3_gazebo)/share/turtlebot3_gazebo/models/
```

Tests

- **Lancer le simulateur Gazebo** :

```
export TURTLEBOT3_MODEL=burger
ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

- **Exécuter le noeud d'évitement** :

```
source install/setup.bash
ros2 run my_package obstacle_avoidance -ros-args -p base_speed:=0.2 -p gain:=1.0
```

Compilation

```
cd exploring-ros2-with-wheeled-robot
colcon build
source install/setup.bash
```

Lancement des comportements

Chacun des fichiers `.cpp` dans `src/` est un nœud indépendant que vous pouvez lancer via `ros2 run`.

---

1. **Obstacle Avoidance**

- **Fichier** : `obstacle_avoidance.cpp`
- **Commande** :

```
ros2 run my_package obstacle_avoidance -ros-args -p base_speed:=0.2 -p
gain:=1.0
```

- **Description** : Le robot évite les obstacles détectés avec le laser. Réagit à la proximité d'objets.

---

2. **Braitenberg Avoidance**

- **Fichier** : `braitenberg_avoidance.cpp`
- **Commande** :

```
ros2 run my_package braitenberg_avoidance -ros-args -p base_speed:=0.2 -p
gain:=1.0
```

- **Description** : Inspiré des véhicules de Braitenberg, le robot réagit aux obstacles de façon comportementale (non déterministe).

---

3. **Wall Follower**

- **Fichier** : `wall_follower.cpp`
- **Commande** :

```
ros2 run my_package wall_follower -ros-args -p base_speed:=0.2 -p gain:=1.0
```

- **Description** : Le robot suit un mur à une distance constante grâce au capteur laser.

---

4. **Wall Follow Equilibrium**

- **Fichier** : `wall_follow_equilibrium.cpp`
- **Commande** :

```
ros2 run my_package wall_follow_equilibrium -ros-args -p base_speed:=0.2 -p
gain:=1.0
```

- **Description** : Variante plus fine du wall following, maintenant une position équilibrée entre plusieurs surfaces.

---

5. **Reading Laser**

- **Fichier** : `reading_laser.cpp`
- **Commande** :

```
ros2 run my_package reading_laser -ros-args -p base_speed:=0.2 -p gain:=1.0
```

- **Description** : Affiche les données brutes du capteur laser sur le terminal. Utile pour le debug.

---

6. **Moving Robot**

- **Fichier** : `moving_robot.cpp`

- **Commande** :

```
ros2 run my_package moving_robot -ros-args -p base_speed:=0.2 -p gain:=1.0
```

- **Description** : Envoie des commandes constantes de mouvement au robot. Utilisé pour tester les déplacements simples.

---

7. **Speed Controller**

- **Fichier** : `speed_controller.cpp`
- **Commande** :

```
ros2 run my_package speed_controller -ros-args -p base_speed:=0.2 -p
gain:=1.0
```

- **Description** : Contrôle la vitesse du robot selon des règles internes ou des paramètres dynamiques.

Pour répondre à la **question 4** :

☑ À quelle vitesse le robot doit-il rouler pour éviter correctement les obstacles ?

La vitesse du robot doit être **modérée**, généralement dans une plage de **0.1 à 0.3 m/s**, en fonction de :

- **La densité d'obstacles** dans l'environnement.
- **La fréquence de mise à jour des capteurs** (par exemple, la fréquence de publication du `laser_scan`).
- **Le temps de réaction de l'algorithme d'évitement** (traitement + commandes moteur).

Aller trop vite pourrait faire dépasser l'obstacle entre deux cycles de détection, surtout si les obstacles sont proches ou si les scans sont lents.

🔍 Paramètre "caché" à prendre en compte

Le paramètre souvent négligé mais **crucial** est le **temps de latence global** du système, qui inclut :

- La fréquence de publication des capteurs (par exemple, 10 Hz = un scan toutes les 100 ms).
- Le délai de traitement dans le nœud de contrôle (algorithme d'évitement).
- Le temps d'exécution de la commande sur le robot (moteurs, simulateur).

Ce temps détermine combien de **distance** le robot parcourt **avant** de réagir à un obstacle. Il faut s'assurer que :

```
distance_avant_réaction = vitesse × latence_totale
```

Donc **nettement inférieure** à la distance minimale entre le robot et un obstacle détecté pour laisser le temps à une manœuvre d'évitement.

⚠ En résumé

- Démarre à une vitesse de **0.1 m/s**, puis augmente prudemment si les performances le permettent.
- Tient compte du **temps de réaction complet** pour assurer une marge de sécurité suffisante.
- Ajuste dynamiquement la vitesse en fonction de la **proximité des obstacles** si possible (stratégie réactive).

Souhaites-tu un petit calcul d'exemple avec des valeurs concrètes ?

8. **Light Follower**

- **Fichier** : `light_follower.cpp`
- **Commande** :

```
ros2 run my_package light_follower -ros-args -p base_speed:=0.2 -p gain:=1.0
```

- **Description** : Le robot suit une source lumineuse simulée. Utilise un topic de lumière simulée (à configurer dans le monde).

---

9. **Swarm Follower**

- **Fichier** : `swarm_follower.cpp`
- **Commande** :

```
ros2 run my_package swarm_follower -ros-args -p base_speed:=0.2 -p gain:=1.0
```

- **Description** : Comportement de robot suiveur dans un scénario de robotique en essaim.

Lancement avec des fichiers de lancement (`launch/`)

Exemple : Lancer le wall follower avec les bons remappages

```
ros2 launch my_package do_something.launch.py
```

Ce fichier lance le nœud `wall_follower` avec les remappings suivants :

- `/laser_scan` → `/dolly/laser_scan`
- `/cmd_vel` → `/dolly/cmd_vel`

Simulation

Les fichiers dans le dossier `worlds/` sont des mondes pour simuler le robot avec Gazebo :

- `dolly_walls.world` : Monde avec des murs, idéal pour tester les followers.
- `map.world` : Monde plus ouvert ou avec d'autres configurations.

License

- Roland Cédric TAYO – Projet ROS 2 - INEM
- Basé sur les ressources pédagogiques fournies dans `tp_ros_etudiant_v2.pdf`