Examen de Middleware/ROS

2h, documents autorisés

1 Téléchargement du package

L'examen est téléchargeable sous forme de package ROS avec la commande suivante :

```
cd ~/ros/src
git clone https://github.com/oKermorgant/ecn_ros2019.git
catkin build ecn_ros2019
gen_qtcreator_config    # sur les PCs de la salle P-robotique
```

2 Description du package

2.1 Le simulateur

Lancez `rviz.launch`:

```
roslaunch ecn_ros2019 rviz.launch
```

RViz est lancé avec quatre robots mobiles:

- Deux robots BB-8 et BB-9 qui suivent une trajectoire prédéfinie
- Deux robot D-0 et D-9 qui attendent vos instructions

Ces robots évoluent en (x, y, θ) et sont commandés avec une vitesse linéaire v et angulaire ω.

2.2 Nodes et topics

Le node `/simulator` rend compte du déplacement des robots. Celui de BB-8 et BB-9 sont imposés, les autres dépendent d'une consigne de vitesse.
Les poses de chaque robot sont disponibles via le topic `/tf`.

Pour plus d'information, les nodes et topics sont visualisables avec `rqt_graph` et accessibles via les commandes `rosnode` et `rostopic`.
Le simulateur ne prend en compte que les champs `linear.x` (vitesse linéaire dans le repère du robot) et `angular.z` (vitesse angulaire).

3 Objectifs

Les robots de type D aiment la compagnie et cherchent à suivre le robot allié (BB-8 pour D-0, BB-9 pour D-9). Inversement, ils sont également craintifs et essaient d'éviter le robot ennemi. Le travail demandé est d'écrire :

- Un node pour réaliser le mouvement d'un robot de type D
- Un launchfile pour exécuter deux fois le node ci-dessus

3.1 Écriture du node

Le node principal est à écrire en C++, en modifiant le fichier `main.cpp`. Ce node doit :

- Prendre trois paramètres privés:
  1. `robot` : le nom du robot contrôlé (`d0` ou `d9`)
  2. `friend`[1] : le nom du robot ami (`bb8` ou `bb9`)
  3. `foe` : le nom du robot ennemi (`bb9` ou `bb8`)
- Obtenir, via un TF Listener, la position relative des robots ami et ennemi
- Publier sur un topic de commande en vitesse (type `geometry_msgs/Twist`), nommé `cmd_vel`. Seuls les champs `linear.x` et `angular.z` sont considérés.
- Publier sur un topic de position articulaire (type `sensor_msgs/JointState`), nommé `jointe_states`.

On pourra commencer par écrire le node pour piloter uniquement un des robots D, avant de rendre le node générique.

3.2 Loi de commande

Via un TF Listener il est possible d'obtenir les positions relatives des robots ami (repère du nom du paramètre `friend`), ennemi (paramètre `foe`) par rapport au repère du robot piloté (paramètre `robot`).

On note (x_1, y_1) la position du robot ami dans le repère du robot piloté, et (x_2, y_2) la position du robot ennemi. La loi de commande est définie par une série d'instructions implémentées dans le fichier `control.h` sous la forme d'une fonction:

```
struct Cmd {double v, w;};
Cmd command(double x1, double y1,double x2, double y2);
```

Cette fonction est à utiliser pour transformer les positions (x_1, y_1, x_2, y_2) en une consigne sous la forme d'un message de type `geometry_msgs/Twist`, à publier sur `cmd_vel`.

[1] Attention en C++, `friend` est un mot-clé

3.3 Positions articulaires

Les robots de type D ont trois articulations: la roue, le torse et le cou. Ces articulations dépendent de la vitesse du robot et leurs positions sont définies comme suit :

Articulation	Nom	Position
Roue	`wheel`	$+3.7.v.dt$ (incrémental)
Torse	`torso`	$v.\pi/12$
Cou	`neck`	$\omega.\pi/12$

La position de la roue est définie de façon incrémentale, par rapport à la position à l'itération précédente. Les autres positions sont définies de façon absolue.
Les noms et positions des articulations doivent être écrits dans un message de type `sensor_msgs/JointState`, à publisher sur `joint_states`. Pour que le message soit pris en compte par la simulation, on doit préciser le time stamp avant de le publier:

```
msg.header.stamp = ros::Time::now();
```

3.4 Écriture du launchfile

Le node ci-dessus utilise des topics génériques, il convient donc de le lancer via un launchfile pour l'application qui nous intéresse.

Écrire un launchfile permettant à D-0 et D-9 de se déplacer. On utilisera des paramètres privés ainsi que la notion de namespace, permettant d'utiliser des noms de topics relatifs dans le code C++.

Pour simplifier le développement du node on pourra dans un premier temps écrire en dur les paramètres permettant à D-0 de suivre BB-8 en évitant BB-9.

4 Soumission du travail

Le package final est à zipper et à soumettre via ce formulaire en ligne.