Subsomption

Revu dernièrement par SD

Version du 11 octobre 2018

Vous pourrez retrouver ce document à cette URL : `http://pages.isir.upmc.fr/~doncieux/public/IAR_2018/`

L'objectif de ce TME est de doter un robot de comportements simples et d'étudier la complexité émergeant de la dynamique issue de l'interaction de ce robot et de ses règles de fonctionnement avec un environnement lui aussi très simple. Il consistera également une introduction à ROS par la pratique.

1 Dans la salle de TME

ROS ayant de nombreuses dépendances, il a été décidé, pour simplifier son installation, de passer par une machine virtuelle contenant une installation complète. Ce choix a limité la puissance disponible pour ROS, nous avons donc utilisé un simulateur très simple. Si vous avez accès à une machine puissante sur laquelle vous pouvez installer ROS, vous pourrez tester le même TME sur des robots plus sophistiqués, comme un turtlebot ou le PR2 en passant par le simulateur 3D Gazebo.

Pour lancer la machine virtuelle, dans le terminal, taper :

```
Vbox IAR_2018
```

(puis entrez votre mot de passe)

ATTENTION : à partir de maintenant, toutes les commandes qui vous seront indiquées devront être lancées dans un terminal de la machine virtuelle.

Pour avoir accès à internet, pensez à configurer le proxy de votre navigateur (dans la machine virtuelle) : nom : proxy port : 3128 (à utiliser pour tous les protocoles).

2 Préparation de l'environnement

Les instructions suivantes expliquent comment installer les sources nécessaires au fonctionnement du TME. Ces opérations ont déjà été réalisées dans la machine virtuelle mise à votre disposition dans les salles de TME. Elles vous sont données pour que vous puissiez continuer hors des salles de TME si vous le souhaitez. Vous pouvez passer directement à la section suivante.

Créer les répertoires `catkin_ws/src` :


```
mkdir -p ~/catkin_ws/src
```

Initialisez l'environnement catkin :

```
cd ~/catkin_ws/src
catkin_init_workspace
```

2.1 Récupération des sources et compilation

Les codes sources utiles pour le TME sont téléchargeables à cette URL : `http://pages.isir.upmc.fr/~doncieux/public/IAR_2018/`.

Vous aurez besoin de :

— `libfastsim.tar.gz`, la bibliothèque de simulation simple de robot. Vous pouvez aussi le récupérer avec `git` :
`git clone https://github.com/jbmouret/libfastsim`

— `ros_fastsim.tar.gz`, le noeud ROS permettant de s'interfacer avec cette bilbiothèque de simulation. Vous pouvez aussi le récupérer avec `git` :
`git clone https://github.com/jbmouret/ros_fastsim`

— `subsomption.tar.gz` : les noeuds que vous allez compléter pendant ce TME.

2.2 Compilation des bibliothèques fournies

2.2.1 `libfastsim`

Décompressez l'archive depuis la racine de votre compte et lancez les commandes suivantes dans le répertoire obtenu :

```
./waf configure
./waf
```

2.2.2 `ros_fastsim`

Allez dans le répertoire `catkin_ws/src`, décompressez l'archive à partir de ce répertoire. Depuis `catkin_ws`, lancer les commandes :

```
source devel/setup.bash
catkin_make
```

ATTENTION : vous devrez exécuter la première commande dans chaque nouveau terminal dans lequel vous voulez exécuter un noeud ROS.

3 ROS

3.1 Guide de survie sous ROS

ROS est un framework logiciel permettant de faciliter le développement d'applications robotiques. Il permet notamment d'échanger des messages entre des noeuds, de définir des services, etc. Il permet d'utiliser des noeuds développés pour un robot particulier sur un autre robot (du moins du point de vue informatique, rien ne garantit

que le noeud marchera correctement avec le nouveau robot...). ROS permet aussi de passer très facilement du robot simulé au robot réel (là encore il est question de facilité "technique", mais ce n'est pas parce qu'un noeud peut facilement se connecter au robot réel qu'il marchera comme prévu...).

Chaque noeud de ROS est exécuté en parallèle des autres. Comme chaque noeud est susceptible de générer ses propres messages, il faut le lancer dans un terminal séparé. L'utilisation de ROS implique donc d'avoir de nombreux terminaux ouverts en même temps. Il est recommandé de les organiser, par exemple en ouvrant de nouveaux onglets plutôt que des fenêtres séparées.

3.1.1 Préparation de l'environnement

3.1.2 ROSCORE

ROS s'appuie sur un serveur qui est le chef d'orchestre et gèrera les échanges entre les différents noeuds, c'est `roscore`. Avant toute utilisation de ROS, il faut donc lancer ce serveur en tapant, dans un terminal la commande :

```
roscore
```

Cette commande doit toujours être active pendant le TME.

3.1.3 Lancement d'un noeud

Le lancement d'un noeud ROS se fait avec la commande `rosrun` :

```
rosrun <package> <executable>
```

Lorsqu'il faut lancer plusieurs noeuds en parallèle ou spécifier beaucoup d'arguments, il est possible de passer par la commande `roslaunch` qui prend en argument un fichier au format XML spécifiant les différents noeuds à lancer et la valeur des paramètres à utiliser. Nous présenterons plus loin comment l'utiliser.

3.1.4 Topics

Sous ROS un topic est un canal de communication entre des noeuds émetteurs et des noeuds récepteurs. Un topic est défini par un nom et un type de message.

Lister les topics disponibles :

```
rostopic list
```

Afficher des détails sur un topic (type de message, émetteur(s), récepteur(s)) :

```
rostopic info <nom de topic>
```

Afficher les messages envoyés sur un topic :

```
rostopic echo <nom de topic>
```

Pour avoir le descriptif d'un message :

```
rosmsg show <nom du message>
```

3.1.5 Autres

Pour faire des tests avec un simulateur de tortue simple (pilotée par les flèches du clavier) :

```
rosrun turtlesim turtlesim_node
rosrun turtlesim turtle_teleop_key
```

Pour voir le graphe des noeuds et des topics qui les relient :

```
rosrun rqt_graph rqt_graph
```

Pour aller plus loin : `http://wiki.ros.org/`.

4 Package `subsomption`

Le package `subsomption` contient le squelette du code que vous aurez à compléter. Il contient également le fichier .launch à utiliser avec fastsim. Le noeud `ros_fastsim` donne accès à un simulateur d'un robot circulaire, de type e-Puck, avec une gestion simplifiée de la physique et des collisions.

Vous devrez d'abord décompresser l'archive `subsomption.tar.gz` dans le répertoire `/catkin_ws/src/`, puis exécuter les commandes :

```
cd ~/catkin_ws
catkin_make install
```

Pour lancer fastsim sur l'environnement défini pour le TME, exécutez les commandes suivantes :

```
cd ~/catkin_ws/src/ros_fastsim/envs
roslaunch subsomption.launch
```

Vous allez à présent créer votre architecture de subsomption. Elle s'appuiera sur le programme en python `subsomption_architecture.py`. Ce noeud ROS, qui prend en argument le nombre n de comportements à définir, écoute les topics `/subsomption/channel`*i*, *i* étant un entier compris entre 0 et n. Sur ce topic sont envoyé des message selon le format suivant :

```
$ rosmsg show subsomption/channel
bool activated
float32 speed_left
float32 speed_right
```

Le noeud de subsomption envoie à fastsim les commandes `speed_left` et `speed_right` venant du topic `/subsomption/channel`*i* avec *i* la valeur la plus haute pour laquelle `activated` vaut `True`.

Exercice(s)

Vous devrez donc définir des comportements et les connecter aux différents canaux du noeud de subsomption. Le comportement actif qui sera connecté au canal d'indice le plus élevé sera celui qui aura la main sur le robot.

Le fichier `example.py` contient un exemple de noeud de comportement à remplir. Il vous est suggéré de partir de ce fichier pour créer vos propres comportements. Pour exécuter le comportement `comportement.py`, vous pouvez lancer :

```
rosrun subsomption comportement.py
```

(`comportement.py` doit alors être exécutable), ou plus simplement :

```
python comportement.py
```

Le noeud contenant l'architecture de sumbsomption devra tourner en parallèle :

```
python subsomption_architecture.py n
```

où n est le nombre de comportements souhaités.

L'évaluation se basera sur une démonstration de vos comportements. NE PARTEZ PAS DE LA SALLE DE TME SANS AVOIR FAIT CETTE DEMO.

Vous devrez également envoyer un email à `stephane.doncieux@upmc.fr` avec, en attachement, les modules python que vous avez développés pour chaque question. Vous les accompagnerez de quelques commentaires expliquant vos choix.

Suivi de mur

1. Écrivez un premier comportement permettant au robot d'avancer tout droit et lancer son exécution sur le canal de priorité la plus faible.
2. En cas de collision (détectée par les bumpers), écrivez un comportement permettant du robot de reculer et de faire demi-tour. Connectez-le au canal de priorité 2.
3. Afin d'éviter les collisions, écrivez un comportement utilisant les lasers pour tourner avant de toucher un obstacle, vous lui donnerez la priorité 1. Vous pouvez changer la couverture des lasers en modifiant le fichier `envs/example.xml`.
4. Définissez un ou plusieurs comportements permettant au robot de suivre les murs. Vous pourrez vous appuyer sur les comportements précédents ou en définir de nouveaux.

Aspirateur

Vous allez maintenant programmer un robot aspirateur ROUMEBA pour qu'il se déplace aléatoirement dans une pièce, ceci dans le but de la nettoyer.

1. Définissez un ensemble de modules permettant de reproduire le comportement observé sur la figure 1.

Application sur le robot réel

Transférez vos comportements sur le robot réel. Les caractéristiques du laser sont les suivantes :
— angle_min : -0.51

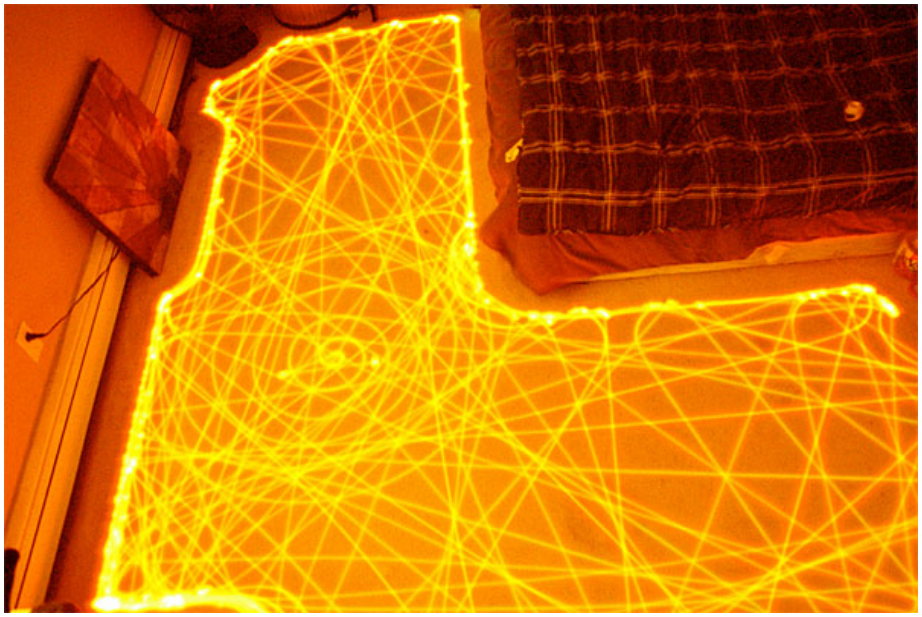

FIGURE 1 – Trajectoire d'un robot aspirateur.

— angle_max : 0.49
— angle_inc : 0.00158
— range_min : 0.449
— range_max : 10.