Examen de Middleware/ROS

1h, documents, codes de TP et internet autorisés

1 Travail demandé

Le but de l'examen est de programmer Baxter pour qu'il détecte une couleur donnée et pointe un de ses bras vers l'objet détecté. Le tout se fera en simulation. Le travail à rendre est sous la forme d'un fichier `main.cpp` (déjà existant et à modifier) et d'un fichier `control.launch` à créer.

2 Initialisation de l'environnement

2.1 Si ROS n'est pas initialisé sur le compte

Si votre environnement n'a pas encore été utilisé pour ROS, exécuter le script `ros_user_setup.sh` à télécharger depuis mon site web:

```
wget http://www.irccyn.ec-nantes.fr/~kermorga/files/ros_user_setup.sh
sh ros_user_setup.sh
```

Refermez ensuite le terminal et ouvrez-en un nouveau.

2.2 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_ros2016.git
```

3 Description du package

Le package téléchargé met en jeu 3 nodes qui communiquent via 4 topics. Le graphe est affiché ci-dessous.

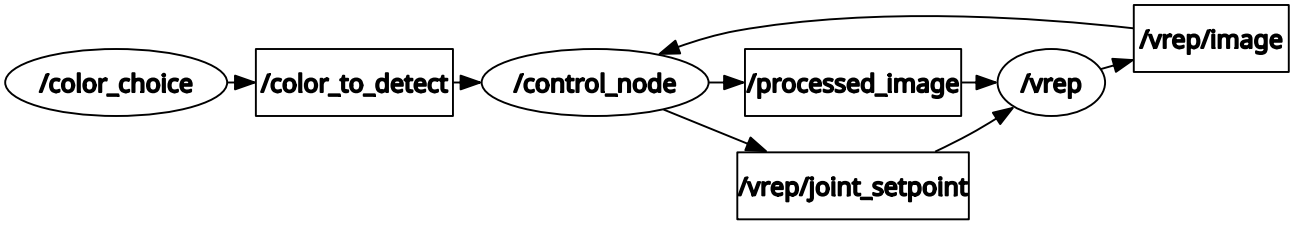

Figure 1: Graphe des nodes et topics

3.1 Nodes et topics existants

Lancez `vrep.launch`:

```
roslaunch ecn_ros2016 vrep.launch
```

Le simulateur V-REP est lancé avec le robot Baxter se tenant devant une table comportant 4 objets de couleur. Il n'y a rien à faire dans l'interface du simulateur, tout passera par les topics. Le node principal est `/vrep` et correspond au simulateur. Ce node publie et souscrit à de nombreux topics et propose également de nombreux services. En pratique les services ne sont pas utilisés. Le node secondaire est `/color_choice` et a pour fonction d'indiquer la couleur à détecter à chaque instant.
Les topics à considérer sont :

1. `/vrep/image` : le simulateur y publie les images de la caméra de Baxter.
2. `/vrep_ros_interface/joint_command` : le simulateur y souscrit pour les consignes de position des bras de Baxter.
3. `/color_to_detect` : le node secondaire y publie la couleur à détecter.

3.2 Paramètres

Au lancement du simulateur, les valeurs RGB de plusieurs couleurs sont chargées en tant que paramètres ROS. La valeur RGB d'une couleur est ainsi accessible via les paramètres :

$$\begin{cases} \texttt{/colors/<couleur>/r} \\ \texttt{/colors/<couleur>/g} \\ \texttt{/colors/<couleur>/b} \end{cases}$$

Par exemple, la syntaxe pour lire la valeur R de la couleur `blue` est ensuite :

```
std::string color = "blue";
int r;
nh.getParam("/colors/"+ color + "/r", r);
```

4 Objectifs

Le travail demandé est d'écrire :

- Un node qui permet de détecter une couleur donnée (via un paramètre) et de publier la consigne de position correspondante.
- Un launchfile pour exécuter quatre fois le même node, détectant à chaque fois une couleur différente. Les couleurs à détecter sont : `blue, red, green, yellow`.

Il est fortement conseillé de développer une première mouture fonctionnelle du node en écrivant en dur la couleur qu'il détectera, avant de le rendre générique via le launchfile.
En bonus, le node peut également publier l'image traitée qui s'affichera dans V-REP.

4.1 Node à créer

Un seul node est à créer en modifiant le fichier `main.cpp`. Ce fichier est compilé en un exécutable appelé `baxter_color`. Le node devra :

- Lire les valeurs RGB correspondant à la couleur passée en paramètre.
- Souscrire à `/vrep/image` pour avoir l'image courante. Ce topic contient des messages de type `sensor_msgs/Image` qui peuvent être convertis en image OpenCV au sein du callback:

```
im = cv_bridge::toCvCopy(msg, "bgr8")->image;
```

- Souscrire à `/color_to_detect` et la couleur courante à détecter. Ces messages sont de type `std_msgs/String` dont la structure est :

```
string data
```

- Publier sur `/vrep_ros_interface/joint_command` les consignes de position du bras. Ces messages doivent être de type `sensor_msgs/JointState` dont la structure est:

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string[] name
float64[] position
float64[] velocity
float64[] effort
```

  où `name` contient les noms des articulations à contrôler et `position` contient les consignes de position des articulations. Les autres champs ne sont pas utilisés.

- Si la couleur à détecter est la couleur que le node doit détecter, traiter l'image et calculer la consigne de position des bras de Baxter.
- Si la couleur à détecter est la couleur que le node doit détecter, publier les consignes de position pour faire bouger le robot.

4.2 Traitement d'image

La partie détection de couleur est déjà implémentée dans une class appelée `ColorDetector`. Cette classe est déjà instanciée mais les valeurs (r,g,b) doivent être récupérées via le paramètre ROS. La position en x et l'aire de l'object détecté sont obtenues via `cd.x()` et `cd.area()`. Pour les objets considérés, une estimation de la position 3D est alors:

$$Z = \frac{0.106}{\sqrt{\text{area}}} \quad \text{et} \quad X = x.Z$$

4.3 Consignes de position

Seules 4 articulations sont à contrôler, dont les noms sont indiqués ci-dessous. Ces noms sont à renseigner dans le champ `name` du message de type `JointState`.
Les consignes sont à renseigner dans le champ `position` du message. Les valeurs des 4 consignes dépendent du signe de X, pour faire bouger un bras ou l'autre en fonction de la position de l'objet détecté :

Articulation	Consigne si $X > 0$	Consigne si $X < 0$
`Baxter_leftArm_joint1`	$-\pi/4 + \text{atan2}(X - 0.25, Z)$	0
`Baxter_leftArm_joint6`	0	$-\pi/2$
`Baxter_rightArm_joint1`	0	$\pi/4 - \text{atan2}(-X - 0.25, Z)$
`Baxter_rightArm_joint6`	$-\pi/2$	0

4.4 Bonus

Le node créé peut également publier l'image traitée après l'appel à la fonction `process`. Le topic sur lequel publier cette image est `/processed_image`. Le message doit être de type `sensor_msgs/Image` qui peut être construit à la volée au moment de la publication avec :

```
cv::cvtColor(im_processed, im2, cv::COLOR_BGR2RGB);
im_pub.publish(cv_bridge::CvImage(std_msgs::Header(), "rgb8", im2).toImageMsg());
```