TP 2 : Les outils de développements de ROS

Thomas Le Mézo - ENSTA Bretagne

2019

1 Design d'une architecture

1.1 Présentation du problème

Dans ce TP, nous allons reprendre l'exemple du bateau muni d'une nacelle.

Quelques explications : la mission du bateau est d'intercepter une cible dont la position sera connue (publiée sur un topic).

— Le bateau devra se rendre à la position de la cible et tourner autour.

— La nacelle devra pointer en permanence sur la cible.

Il vous faudra implémenter les nodes suivants :

— Un node simulant le bateau et la tourelle. Le bateau sera représenté par le vecteur d'état $\mathbf{x}_{bateau} = (x, y, \theta)$, on supposera sa vitesse constante et égale à 1 ; la nacelle par $\mathbf{x}_{nacelle} = (\theta)$ (on choisit de simplifier le problème).
On considère la dynamique suivante pour le bateau $f_1(X) = (v \cdot \cos(\theta), v \cdot \sin(\theta), u_1)$ qui sera donc piloté en variation de cap (on suppose $v = 1$ pour commencer). Et de manière identique $f_2(X) = (u_2)$.

— Un node de contrôle du bateau. On pourra utiliser la loi de commande suivante : soit e l'angle entre l'axe avant-arrière du bateau et la droite passant par le centre du bateau et la cible (ou encore l'angle de gisement).
  — Si $e > pi/4$ on tourne à fond à droite.
  — Sinon, on applique une loi proportionnelle à e.
On pourra supposer que $\dot{\theta} \in [-\pi/10, \pi/10]$ rad/s.

Question 1 *Créez un nouveau package appelé* `tp2`*. Vous ferez dépendre ce package de* `roscpp rospy tf std_msgs message_generation message_runtime geometry_msgs`*.*

1.2 Rappels & Aide

Dans la mesure du possible, il est préférable d'utiliser des messages standards de ROS. Vous trouverez le descriptif des messages (utiles) aux adresses suivantes :

— `http://wiki.ros.org/std_msgs`

— `http://wiki.ros.org/geometry_msgs`

Donnée physique	Message ROS
Position et Orientation	`geometry_msgs/PoseStamped`
Vitesse et Rotation	`geometry_msgs/Twist`
Tableau de float	`std_msgs/Float64MultiArray`

FIGURE 1 – Résumé des messages importants

Pour utiliser les fonctions mathématiques (cos, sin, etc.) en C++, vous pouvez charger la librairie :

```
#include <cmath>
```

Pour utiliser les fonctionnalités de C++11, il faut dé-commenter la ligne suivante dans le fichier `CMakeLists.txt` de vos packages :

```
add_compile_options(-std=c++11)
```

2 Simulateur du bateau

2.1 Premiers pas

Question 2 *Créez un nouveau node chargé de simuler le comportement du bateau. La dynamique sera simulée par un schéma d'intégration d'Euler des équations d'états. Pour cela, codez une fonction* `integration_euler()` *qui met à jour l'état du bateau X en fonction de la commande u et d'un dt. Cette fonction sera appelée dans la boucle principale du programme à 25Hz.*

Question 3 *Publiez la position et l'orientation du bateau dans un message de type* `PoseStamped`*. Nous supposerons l'altitude constante égale à zéro.*
Renseignez dans un premier temps l'information de position en étudiant la structure du message.
Vous remarquerez que ROS utilise des Quaternions plutôt que les angles d'Euler dans ses messages. Ils permettent d'éviter le phénomène de "Blocage de cadran" (Gimbal lock). Il existe une méthode pour effectuer la conversion entre les angles d'Euler et les Quaternions :

```
#include "tf/tf.h"
tf::Quaternion q;
q.setRPY(roll, pitch, yaw);
tf::quaternionTFToMsg(q, msg.pose.orientation);
```

Attention aux conventions pour les angles, nous choisissons la convention RPY (Roll, Pitch Yaw) d'où la fonction `setRPY()`*.*

2.2 Découverte de RQT

Pour vérifier le bon fonctionnement de votre node, nous allons utiliser l'utilitaire *RQT* de ROS (Ros-Qt). De nombreux plugins sont disponibles dans ce logiciel ce qui permet d'avoir des outils graphiques d'aide au développement et au contrôle en temps réel de l'état de votre robot.

Question 4 *Entrez la commande* `rqt` *dans un terminal.*
Nous allons utiliser le plugin "Topic Monitor" que vous trouverez dans le menu "Plugins>Topics". En cliquant sur la case à cocher de votre topic qui publie la position du bateau, assurez vous que le message est correct avec la bonne fréquence.

D'autres plugins sont également intéressants à utiliser. Dans un premiers temps vous pouvez essayer les deux suivants :

- "Visualization>Plot" pour afficher des graphs en temps réel,
- "Introspection>Node Graph" pour afficher l'ensemble de vos nodes et les échanges d'informations entre-eux.

Vous devriez constater que les courbes s'affichent mal : c'est à cause du champ `stamp` du message qui gère le temps et qui est bloqué sur 0s. Vous pouvez utiliser la fonction :

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

pour remplir ce champ.

Question 5 *Abonnez votre node à un message de type* `Twist` *pour la commande u. Ce message fournira une consigne de rotation suivant l'axe z.*
Testez votre programme en publiant une commande u via le plugin "Message Publisher" de RQT et en visualisant la position en x, y de votre robot.

2.3 Roslaunch

Le lancement des nodes à la main est fastidieux. Nous allons maintenant découvrir `roslaunch` qui devrait nous permettre d'automatiser le lancement de vos nodes.

La structure minimale d'un fichier `.launch` est la suivante :

```
<launch>
        <node name="boat_simulator" pkg="tp2" type="boat_simulator" />
</launch>
```

— `name` correspond au nom de l'instance que vous voulez donner au *node* (qui peut être différent du nom du fichier binaire).
— `type` au nom de votre exécutable

À savoir : Roslaunch s'occupe de lancer le *master* dans le cas où il n'en trouve pas d'actif sur votre machine.

Pour lancer un fichier `.launch` avec roslaunch, la commande est la suivante :

```
roslaunch package_name file.launch
```

Pour continuer d'afficher les messages de debug de type `ROS_INFO()` dans la console, il faut également ajouter l'option de node `output="screen"`.

Question 6 *Créez un fichier de type `.launch` à la racine de votre package pour lancer le node de simulation du bateau et testez le lancement automatique.*

Remarque : roslaunch possède de nombreuses autres options dont vous aurez surement besoin dans la suite de ce TP. Vous pouvez consulter une documentation plus complète à l'adresse suivante : `http://wiki.ros.org/roslaunch/XML`.

2.4 Parameters

Lorsque vous lancez un node, il arrive assez souvent que vous souhaitiez lui donner des paramètres de configuration. Par exemple, il pourrait être utile de spécifier une position initiale au bateau. C'est le rôle du *Parameter Server*.

Pour accéder à un paramètre du server, il suffit d'insérer les lignes suivantes :

```
// pour n le NodeHandle
n.param<double>("nom_du_parametre", variable, valeur_par_defaut); // pour un float
```

Des variantes de cette fonction existent notamment en fonction du *namespace*, que nous verrons plus tard. Vous pouvez consulter la documentation sur les liens suivants : Parameter Server et Roscpp Implementation.

Les paramètres peuvent être passés en argument dans le fichier `.launch` grâce à la ligne suivante :

```
<param name="position_x" type="double" value="10.0" />
```

Question 7 *Ajoutez dans le simulateur du bateau la possibilité de définir la position initiale du bateau par des paramètres. Complétez le fichier `.launch` pour tester l'utilisation des paramètres.*

Remarque : Il est également possible d'envoyer des paramètres au lancement d'un *node* par la commande `rosrun`.

```
rosrun tp2 boat_simulateur position_x_init:=10.0
```

2.5 Découverte de RVIZ

Pour tester si votre bateau se comporte bien, nous allons maintenant découvrir l'outil RVIZ qui permet d'afficher la trajectoire en 3D d'objets dans une "scène".

Question 8 *Lancez RVIZ avec la commande `rviz` dans un terminal. Pour ajouter des objets à la scène, utilisez le bouton "Add". Ajoutez un objet de type "Pose" pour visualiser le bateau et abonnez vous au topic publiant sa position. Des erreurs devraient apparaitre.*

Toutes les informations affichées dans RVIZ doivent être référencées temporellement et spatialement. Vous devriez voir le nom du "Fixed Frame" par rapport auquel RVIZ positionne les objets de la scène dans la catégorie "Global Options".

Question 9 *Dans votre message publiant la position du robot, renseignez les champs `frame_id` avec le nom de la "Fixed Frame" et `stamp` avec un temps (en utilisant `ros::Time::now()` par exemple).*
Vérifiez que le bateau s'affiche correctement dans RVIZ.

Il est également possible sous RVIZ d'afficher un modèle 3D : plus d'information ici.

Pour afficher un modèle 3D, il faut publier un message un peu particulier. Il s'agit d'un "Marker" du package `visualization_msgs`. La description des champs à remplir est donnée sur la page web. Pour afficher le modèle 3D, il faudra écrire les lignes suivantes et aller vérifier la documentation en ligne !

```
ros::Publisher vis_pub = node_handle.advertise<visualization_msgs::Marker>( "visualization_marker", 0 );

visualization_msgs::Marker marker;
marker.header.frame_id = "map";
marker.header.stamp = ros::Time();
marker.ns = "boat";
marker.id = 0;
marker.type = visualization_msgs::Marker::MESH_RESOURCE;
marker.action = visualization_msgs::Marker::ADD;
marker.pose = // ToDo
// tf::quaternionTFToMsg(q, marker.pose.orientation);
marker.scale.x = 1;
marker.scale.y = 1;
marker.scale.z = 1;
marker.color.a = 1.0; // alpha = transparence
marker.color.r = 1.0;
marker.color.g = 1.0;
marker.color.b = 1.0;
marker.mesh_resource = "package://tp2/meshs/boat.dae";
vis_pub.publish( marker );
```

Question 10 *Téléchargez le modèle 3D (`boat.dae`) du bateau sur Moodle et placez le dans un nouveau répertoire `meshs` de votre package `tp2`. Créer un nouveau topic qui publie le modèle 3D du bateau. Depuis RVIZ, abonnez vous à ce topic pour afficher le bateau ("Add">"By Topic">nom du topic).*

Remarque : Les fichiers `.dae` utilisés dans ce TP ont été générés depuis un modèle 3D Blender. Vous pouvez également charger des fichiers de type `.stl` (CAO) ou `.mesh` (Ogre) dans RVIZ.

2.6 Installation de Packages additionnels

Avant d'implémenter un contrôleur pour le bateau, nous allons essayer de piloter manuellement le bateau (clavier ou joystick). Pour cela, installez les packages de téléopération suivant :

```
sudo apt install ros-melodic-teleop-tools ros-melodic-key-teleop ros-melodic-mouse-teleop
 ↪  ros-melodic-joy-teleop
```

Remarque : D'une manière générale, l'ensemble des packages de ROS peuvent être téléchargés sous le nom `ros-melodic-nom_du_package`.

Le package `key_teleop` va publier un message de type `Twist`. Pour le publier sur le bon topic, ce package vous donne la possibilité de remaper le nom du topic de publication.

Question 11 *Testez le pilotage de votre bateau à l'aide de la souris ou du clavier*

```
roslaunch mouse_teleop mouse_teleop.launch mouse_vel:=votre_nom_de_topic
# ou
rosrun key_teleop key_teleop.py key_vel:=votre_nom_de_topic
```

3 Contrôleur du bateau

Dans cette partie, nous allons écrire un contrôleur pour notre bateau et voir comment nous pouvons facilement générer plusieurs bateaux sans ré-écrire de code.

3.1 Loi de pilotage

Question 12 *Créez et testez un nouveau node pour le pilotage du bateau. Implémentez un contrôleur en suivant la loi de commande de l'introduction. On supposera que la position de la cible est publiée sur un topic (message de type* `PoseStamped`*).*

- Remarque 1 : pour l'arc-tangente, il est conseillé d'utiliser la fonction $\theta = atan2(y, x)$ en C++.
- Remarque 2 : votre node devra normalement être abonné au simulateur et à la position de la cible et également publier une commande pour le simulateur.
- Remarque 3 : la position de la cible pourra également être affichée dans RVIZ.
- Remarque 4 : pour passer d'un Quaternion à des angles d'Euler, vous pouvez utiliser la fonction

```
// SOLUTION 1
#include "LinearMath/btMatrix3x3.h"
double roll, pitch, yaw;
tf::Matrix3x3(q).getRPY(roll, pitch, yaw); // Voir Question 3, avec q le quaternion

/// SOLUTION 2
double yaw = tf::getYaw(q); // Avec q un message de type Quaternion (geometry_msgs)
```

- Remarque 5 : Pour calculer une erreur entre deux angles et gérer le problème de discontinuité, on peut utiliser la fonction dents de scie : $e = 2 \cdot \text{atan}(\tan(\Delta\theta/2))$.
- Remarque 6 : On pourra choisir de limiter la vitesse de rotation du bateau tel qu'il réalise un rayon de giration de 5m à la vitesse de 1m/s.

3.2 Node Graph & namespaces

Pour bien vérifier le fonctionnement de l'ensemble de vos nodes, nous allons utiliser un outil très pratique dans RQT appelé *Node Graph*.

Question 13 *Dans RQT, chargez le plugin "Node Graph" et vérifiez la structure des nodes en cours d'exécution. Quelle est la correspondance des formes et couleurs par rapport aux topics, nodes, subscrivers et publishers ? Votre simulateur est-il correctement abonné à votre contrôleur ?*

Nous allons maintenant étudier le principe des *namespaces*.

- Première étape : vérifiez que vos nodes publient de manière "relative" : i.e. il ne doit pas y avoir de "`/`" au début du nom des topics.
- Seconde étape : dans votre fichier `.launch`, ajouter la balise suivante pour déclarer un namespace (`ns`) :

```
<group ns="ns_name">
        ...
</group>
```

Question 14 *Faire apparaitre trois bateaux simultanément avec leur simulateur et contrôleur associé. La position de la cible sera publiée sur un topic global à tous les bateaux. Vérifiez le fonctionnement de votre code avec le Node Graph.*

Remarque 1 : pour afficher trois Marker différent, il faudra remplir la variable namespace (`marker.ns`) avec des nom différent pour chaque bateau. Vous pouvez utiliser la fonction

```
std::string ns = ros::this_node::getNamespace();
```

pour obtenir le nom du namespace défini dans votre fichier `.launch` qui pourra également vous servir à remplir la variable namespace du Marker.

Remarque 2 : pour éviter d'ajouter trois marker dans RVIZ, vous pouvez publier le message de mesh sur le même topic (nom global avec un "/").

Travail à rendre

- Une capture d'écran du Node Graph de la Question 14.
- Une capture vidéo ou une capture d'écran de RVIZ à la question 14.
- Archive de votre package `tp2` avec vos noms.

Pour réaliser une capture vidéo, il est très fortement recommandé d'utiliser les commandes de la librairie *ffmpeg* sous linux.

```
sudo apt install ffmpeg
```

```
ffmpeg -video_size 1280x720 -framerate 25 -f x11grab -i :0.0+0,0 -c:v libx264 -qp 0 -preset ultrafast
↪   video.mkv
```

Détails sur quelques options :

- `-video_size` : taille de la fenêtre
- `-i :0.0+50,60` : point en haut à gauche de la fenêtre par rapport à l'écran en x=50,y=60.