Département de génie informatique et génie logiciel

INF3995

Projet de conception d'un système informatique

Fichier expliquant les tests des fichiers intestables unitairement

Équipe No 102

Youssef Chourifi, Hannan Dissou, Ania Ben Abdesselam, Mady Semega, Ismael Adam Soule, Koceila Lakhdari

Avril 2021

Méthode de test

Les fichiers qui n'ont pas pu être tester sont tous des fichiers nœuds (node) ROS que nous avons créé. Ils vont tous être soumis à la même méthodologie de test. En effet, pour nous assurer que nous avons bien tester un requis, nous allons lui faire passer un test en quatre étapes :

1. Lancer le nœud. Nous assurer qu'elle répond bien à la commande lancée sur un topique en particulier et qu'elle effectue bien sa tâche.
2. Intégrer le nœud dans nos conteneurs Docker et nous assurer qu'elle se lance bien dans le conteneur et que l'étape 1 est toujours valide.
3. Lancer la simulation et tester avec l'architecture au complet en mode simulation pour nous assurer que la fonctionnalité développée fonctionne en validant la condition 2.
4. Tester l'architecture avec le nouveau nœud directement sur les robots Limo et nous assurer que la condition 2 fonctionnes toujours.

En testant nos fonctionnalités de cette manière, nous nous assurons petit à petit que la fonctionnalité que nous développons ait le comportement désiré.

Nœud explore_control

L'objectif de ce nœud est de lancer et arrêter Explore Lite dépendamment du message booléien qu'il reçoit sur le topic exploration_state.

Étape 1

L'étape 1 consiste à lancer la node avec la commande suivante :

rosrun explore_control control_explore.py

Ensuite, lorsque le nœud s'est lancé, nous allons publier avec la commande suivante

rostopic pub exploration_state std_msgs/Bool { data: true }

Lorsque cela est fait, nous nous assurons de voir le node « explore » qui s'est lancé.
Lorsque cela est fait, nous allons renvoyer la commande suivante pour arrêter Explore_Lite :

rostopic pub exploration_state std_msgs/Bool { data: false}

Nous devons observer le nœud « explore » disparaitre de la liste des nœuds actifs. Si cela s'est bien passé, le nœud a passé la première étape.

Étape 2

L'étape 2 consiste à ajouter le nœud control_explore.py dans notre conteneur de librairie de dépendances. Cela est fait en l'ajoutant dans le fichier Dockerfile et en la lançant dans le fichier entrypoint.sh dans /Limo/ros-packages depuis la racine du projet.

Lorsque cela est fait, nous pouvons lancer le coté robot avec le fichier start-limo.sh dans le dossier Limo. Si tout se lance correctement, nous pouvons passer au test de la node.

Nous devons en premier lieu nous assure qu'elle s'est bien lancée avec la commande :

rosnode list

Ensuite, lorsque le nœud s'est lancé, nous allons publier avec la commande suivante

rostopic pub exploration_state std_msgs/Bool { data: true }

Lorsque cela est fait, nous nous assurons de voir le node « explore » qui s'est lancé.
Lorsque cela est fait, nous allons renvoyer la commande suivante pour arrêter Explore_Lite :

rostopic pub exploration_state std_msgs/Bool { data: false}

Si tout fonctionne toujours, nous pouvons passer au troisième test.

Étape 3

Notre node fonctionne dans notre conteneur, nous pouvons donc passer au test avec l'architecture. Nous allons lancer le projet en simulation. Puis nous allons tout simplement lancer une mission et voir si le nœud explore s'est lancé et les robots bouger. Si oui, nous faisons arrêter la mission et le nœud explore est supposé partir et les robots s'arrêter. Si tout s'est bien passé, nous pouvons passer à la dernière étape.

Étape 4

L'étape 4 est essentiellement la même que l'étape 3, mais sur les robots physiques. Si les mêmes étapes qu'à l'étape 3 fonctionne avec les robots physiques, alors le nœud est fonctionnel et peut être intégré au système.

Nœud return_to_base

L'objectif de ce nœud est de faire revenir le robot à la base en recevant un true sur le topic return_to_base.

Étape 1

L'étape 1 consiste à lancer la node avec la commande suivante :

rosrun explore_control return_to_base.py

Ensuite, lorsque le nœud s'est lancé, nous allons publier avec la commande suivante

rostopic pub return_to_base std_msgs/Bool { data: true }

Lorsque cela est fait, nous nous assurons de voir des messages qui vont apparaitre sur les topics suivants :

- exploration_state avec le message False
- move_base_simple/goal avec un goal à atteindre

Si cela s'est bien passé, le nœud a passé la première étape.

Étape 2

L'étape 2 consiste à ajouter le nœud return_to_base.py dans notre conteneur de librairie de dépendances. Cela est fait en l'ajoutant dans le fichier Dockerfile et en la lançant dans le fichier entrypoint.sh dans /Limo/ros-packages depuis la racine du projet.

Lorsque cela est fait, nous pouvons lancer le coté robot avec le fichier start-limo.sh dans le dossier Limo. Si tout se lance correctement, nous pouvons passer au test de la node.

Nous devons en premier lieu nous assure qu'elle s'est bien lancée avec la commande :

rosnode list

Ensuite, lorsque le nœud s'est lancé, nous allons publier avec la commande suivante

rostopic pub exploration_state std_msgs/Bool { data: true }

Lorsque cela est fait, nous nous assurons de voir des messages qui vont apparaitre sur les topics suivants :

- exploration_state avec le message False

- move_base_simple/goal avec un goal à atteindre

Si tout fonctionne toujours, nous pouvons passer au troisième test.

Étape 3

Notre node fonctionne dans notre conteneur, nous pouvons donc passer au test avec l'architecture. Nous allons lancer le projet en simulation. Puis nous allons tout simplement lancer une mission. Une fois cela fait, nous appuyons sur le bouton de retour à la base dans l'interface. Si le robot se déplace vers sa position initiale avec les messages qui apparaissent sur les topics comme au point 2, nous pouvons passer à la dernière étape.

Étape 4

L'étape 4 est essentiellement la même que l'étape 3, mais sur les robots physiques. Si les mêmes étapes qu'à l'étape 3 fonctionne avec les robots physiques, alors le nœud est fonctionnel et peut être intégré au système.

Nœud restart_package_container

Ce nœud a pour but de d'arrêter son excision lorsqu'elle reçoit un message sur le topic restartPackagesContainer. Avec le drapeau « --restart always » sur le conteneur de dépendance de librairie, celui-ci va redémarrer lorsque la node aurai arrêter son exécusion.

Étape 1

L'étape 1 consiste à lancer la node avec la commande suivante :

rosrun update_pkg restart_package_container.py

Ensuite, lorsque le nœud s'est lancé, nous allons publier avec la commande suivante

rostopic pub return_to_base std_msgs/Bool {data: true }

Lorsque cela est fait, nous nous assurons de voir que l'exécision de la node s'est arrêter et que celle-ci a disparu dans la liste des nodes. Si cela s'est bien passé, le nœud a passé la première étape.

Étape 2

L'étape 2 consiste à ajouter le nœud restart_package_container.py dans notre conteneur de librairie de dépendances. Cela est fait en l'ajoutant dans le fichier Dockerfile et en la lançant dans le fichier entrypoint.sh dans /Limo/ros-packages depuis la racine du projet.

Lorsque cela est fait, nous pouvons lancer le coté robot avec le fichier start-limo.sh dans le dossier Limo. Si tout se lance correctement, nous pouvons passer au test de la node.

Nous devons en premier lieu nous assure qu'elle s'est bien lancée avec la commande :

rosnode list

Ensuite, lorsque le nœud s'est lancé, nous allons publier avec la commande suivante

rostopic pub exploration_state std_msgs/Bool {data: true }

Lorsque cela est fait, nous nous assurons que le conteneur de librairies (ros-packages-server) ait bien été fermé puis redémarré en regardant les logs du conteneur. Si cela s'est bien passé, nous pouvons passer à l'étape suivante

Étape 3

Notre node fonctionne dans notre conteneur, nous pouvons donc passer au test avec l'architecture. Nous allons lancer le projet en simulation. Puis nous allons tout simplement appuyer sur le bouton de mise à jour dans l'interface. Si le conteneur de librairies (ros-packages-server) ait bien été fermé puis redémarré, cela veut dire que tout s'est bien passé et que nous pouvons passer à l'étape suivante

Étape 4

L'étape 4 est essentiellement la même que l'étape 3, mais sur les robots physiques. Si les mêmes étapes qu'à l'étape 3 fonctionne avec les robots physiques, alors le nœud est fonctionnel et peut être intégré au système.

Nœud save_map

Ce nœud a été commencé, puis nous avons pris la décision de ne pas faire le requis de cette fonctionnalité (R.F.18). Ce nœud est donc non testé.