Test de recrutement robotique : iFollow

Objectifs :

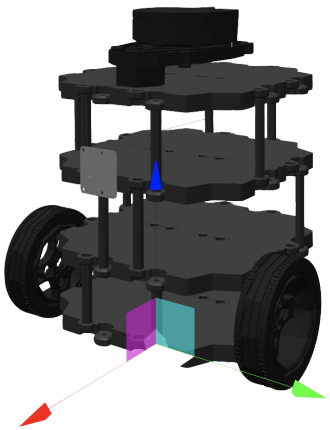
Le but de cet exercice est d'évaluer les capacités du candidat à intégrer et utiliser différentes technologies logicielles au sein du framework ROS.

Evaluation :
Nous allons évaluer la qualité de rédaction des réponses, de la manière de coder les différents exercices, mais aussi le temps mis par le candidat pour résoudre chaque exercice. Le fait de ne pas avoir réussi ***une ou plusieurs questions n'est pas éliminatoire****, l'ensemble des réponses seront évaluées à leur juste valeur. Faites de votre mieux et bon courage !*

Dans ce test, nous cherchons à contrôler un robot mobile de différentes manières.

Nous mettons à disposition du candidat :

- Des AprilTags dans le dossier *tags_test.zip* permettant de répondre aux questions 4 à 6.



Turtlebot 3 modèle Burger

1. Mise en place de l'environnement de test :

Pour cet exercice, vous installez ROS 18.04 ou bien 20.04 et vous utiliserez une simulation du Turtlebot 3 (model burger).

Dans cette question, aucun développement n'est demandé. Il suffit de se baser sur des ressources déjà existantes et disponibles sur le web. On souhaite faire évoluer ce robot dans un environnement simulé.

Rédigez (et exécutez) l'ensemble des étapes permettant de mettre en place cette simulation du turtlebot 3. A la fin de ces étapes, vous devez disposer :
- d'une visualisation du robot et d'une map dans Rviz (map laissée au choix du candidat).
- d'une stack de localisation et de navigation permettant au robot de se localiser et de se déplacer dans cette map.

Rédiger également les commandes à exécuter pour :
- Contrôler le robot en téléopération (clavier)
- Contrôler le robot en lui donnant un nav goal 2D

2. Multiplexeur de commande.

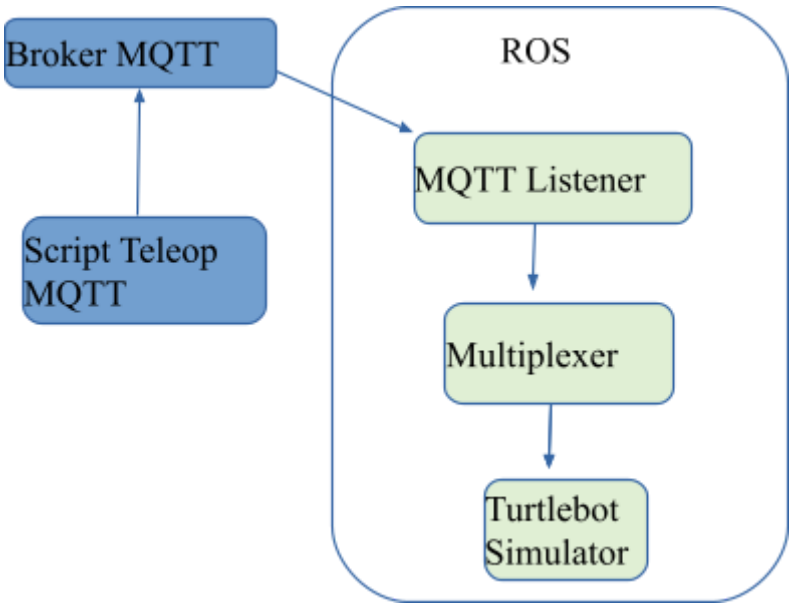
Le robot peut être piloté en consigne de vitesse par différentes sources de contrôle. Ajoutez un nœud permettant de multiplexer une commande provenant de 2 sources de contrôle différentes.
Appelons ces 2 entrées de commande de vitesse /cmd_local et /cmd_web. Précisez les commandes à effectuer pour switcher d'une source de contrôle à une autre.

3. Téléopération à distance.

Nous souhaitons maintenant piloter le robot simulé "à distance" (même si pour l'exercice, l'ensemble de ces codes est exécuté sur le même PC). Une première solution serait de faire un export ROS_MASTER_URI sur un pc distant pour directement publier des consignes de vitesses dans le topic correspondant. Nous ne voulons pas procéder ainsi. Ici, nous souhaitons plutôt contrôler le robot avec un pc "distant" ne disposant pas de ROS. Pour cela nous utiliserons le protocole MQTT. Ce protocole permet de publier et de souscrire à des "topics" (non ROS) en transitant par un broker MQTT.

- Côté simulateur ROS :
Développez un nœud python ou C++ ROS permettant de souscrire à un topic MQTT et de publier une consigne de vitesse sur le topic ROS /cmd_web . Vous devez définir vous même le format du message MQTT représentant la consigne de vitesse.

- Côté client non ROS :
Développez un script python en C++ (indépendant de ROS) qui observe l'appui des touches directionnelles du clavier pour envoyer une consigne de vitesse via une publication MQTT. On souhaite ainsi recréer un mode téléopération du robot, mais en passant ici par le protocole MQTT.



4. Envoi de Goal déterminé par un tag visuel

Indépendant de Q3.

Nous décidons maintenant d'envoyer des nav goals au robot d'une manière différente. Au lieu de renseigner au robot un goal via un topic ou un service, nous souhaitons ici lui envoyer une photo contenant un code. A chaque code correspond une position 2D. Ce code une fois reconnu doit permettre l'envoi du nav goal correspondant. Ce code est un marker visuel de type AR tag.

Vous trouverez en pièce jointe 3 April Tags (AR-tags) différents. Pour chaque AR-tag vous disposez de son original au format pdf et d'une photo de l'AR tag imprimé. Définissez 3 nav goals dans votre carte et pour chacun des goals, choisissez un AR-tag fourni.

Développez ensuite un noeud ROS (c++ ou python) permettant :
- de recupérer une photo contenant l'image code (au format jpg en renseignant le chemin du fichier de la photo de l'AR-tag par exemple)
- d'analyser la photo et décoder l'AR-tag
- et enfin d'envoyer le nav goal au robot correspondant à cet AR-tag

5. Utilisation des caractéristiques visuelles (bonus)

Indépendant de Q3

- Faites en sorte que l'orientation du tag de l'image corresponde à l'orientation du nav goal envoyé.

6. Utilisation d’une capture d’image (bonus)

Modifiez votre code de façon à ce que le noeud ROS de décodage d’AR-tag puisse:
- prendre une photo avec la webcam du PC (sur l’appel d’un service Trigger par exemple)
- analyser l’image capturée
- décoder l’AR-tag présenté sur la photo s’il y en a un
- et enfin envoyer le robot au nav goal correspondant à l’AR tag en question

Documents attendus :

L’ensemble de votre travail devra être disponible sur un repository GitHub ou GitLab (**en public**). Présentez l’ensemble des réponses et la présentation de votre travail dans le fichier Readme. Précisez (honnêtement) le temps que vous avez passé sur chacune des questions.