L’objectif de ce projet long est de permettre aux étudiants de mettre en pratique l’utilisation des réseaux de Petri. Pour cela, il nous a fallu créer la couche basse de l’ensemble du programme afin qu’ils aient à disposition des fonctions haut niveau qui leur permettront de faire évoluer les différents composants de la cellule flexible en simulation.

Nœud Robot

La cellule flexible est composée, en plus de la ligne Montrac, de quatre robots industriels. Nous avons choisi de les modéliser en simulation uniquement par des modèles KUKA LBR IIWA 14 R820, des robots industriels 6 axes, afin de pouvoir implémenter un code facilement applicable à l’ensemble des robots.

Chaque robot est une ressource partagée entre quatre emplacements qui lui sont associés : deux zones de chargement/déchargement et deux postes de travail, ou tâches, qui permettent de traiter les produits.

Pour commander ces robots, des fonctions haut niveau seront à disposition des élèves afin qu’ils puissent les déplacer d’une position à une autre, de descendre ou monter le bras ou encore d’ouvrir ou fermer la pince. Il est donc nécessaire que ces robots puissent évoluer en parallèle et être commander indépendamment des autres. Pour cela, nous avons choisi de réaliser un nœud ROS par robot.

Réalisation

Chaque composant défini dans le logiciel V-Rep est défini par un entier appelé *handle* qui lui est propre. Nous allons pouvoir contrôler les robots en simulation à l’aide de ces entiers. Ils sont obtenus à l’aide d’un service V-Rep : simRosGetObjectHandle. Ce service nécessite le nom exact de chaque joint des robots pour en retourner l’*handle*. Ce nom comporte le nom du robot suivi du numéro du joint et sera donc le seul élément qui différera dans les différents nœuds ROS associés aux robots.

Une fois le *handle* associé à chaque joint d’un robot récupéré, nous allons pouvoir commander les mouvements de celui-ci. Pour cela, nous avons réalisé plusieurs fonctions internes au nœud ROS permettant le contrôle du robot : une pour atteindre une position prédéfinie, une permettant d’envoyer le robot dans une position définie par ses arguments, une mettant le bras en position basse, une autre pour relever le bras, ou encore une fermant la pince du robot, et une dernière pour ouvrir cette même pince.

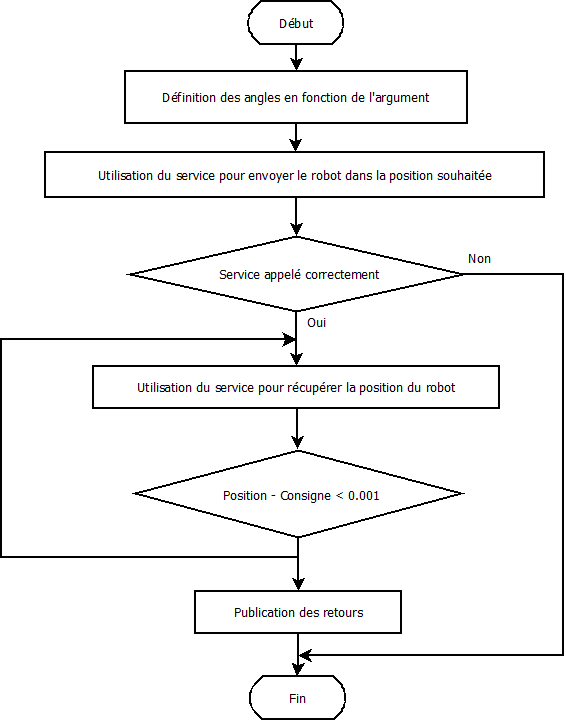
Les fonctions internes au nœud ROS

Exceptées celles permettant la commande de la pince, les fonctions que nous venons de citer sont toutes implémentées de la même manière.

Dans un premier temps, on définit un tableau de 7 valeurs. Ce tableau contient la valeur des différents angles du robot, permettant de définir la position de celui-ci. Ensuite, on utilise le service simRosSetJointState qui communique entre notre nœud ROS et le logiciel V-Rep afin de placer le joint du robot, identifié par son *handle*, dans la position que nous venons de définir.

Après s’être assuré que ce service avait été correctement appelé, nous en utilisons un deuxième, le service simRosGetJointState, qui permet d’obtenir la position actuelle de chaque joint du robot. Ainsi, nous allons pouvoir tester l’erreur entre cette position réelle et la consigne définie afin de savoir si le mouvement du robot est terminé. On considère que la position souhaitée est atteinte lorsque la différence entre celle-ci et la consigne est inférieure à 0,1%.

Enfin, une fois ce test validé, on envoie des retours en publiant sur différents *topics* pour signifier à la commande ou aux autres nœuds nécessitant la position du robot que le mouvement est terminé et que la position souhaitée est atteinte.

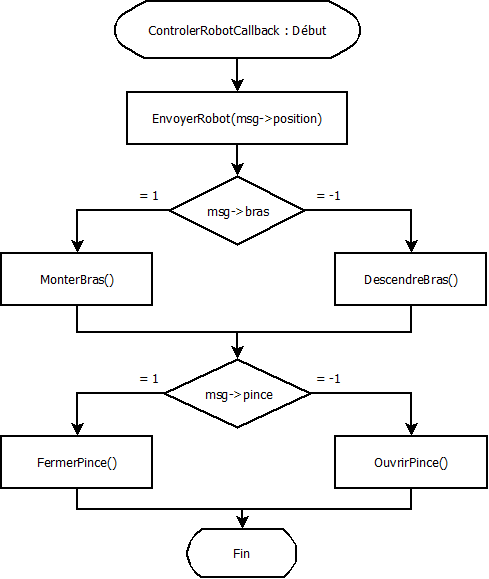
Par exemple, pour envoyer un robot dans une position prédéfinie, on utilise la fonction EnvoyerRobot(int numposition) :

La commande de la pince est réalisée de manière différente. Pour fermer ou ouvrir la pince, il faut simplement publier un message sur un *topic* permettant la communication avec l’objet V-Rep, 0 correspondant à l’ordre d’ouverture et 1 à celui de fermeture. De plus, comme pour les fonctions permettant de contrôler la position du robot, des retours sont réalisés pour connaître l’état de la pince à tout moment, mais aussi pour s’assurer que la pince est effectivement fermée ou ouverte. Ce dernier retour ne pouvant être réalisé à l’aide d’un service V-Rep, nous avons implémenté une temporisation basée sur le temps de la simulation, pouvant être récupéré à l’aide du service simRosGetInfo. On considère qu’après une seconde V-Rep, la pince a fini son mouvement.

Les fonctions Callback

Ces fonctions internes permettent au robot d’être commandé par le programme principal associé au nœud. Cependant, ce n’est pas ce que nous souhaitons dans ce projet puisque nous voulons que les robots soient contrôlés par l’utilisation de fonctions haut niveau. Aussi, des fonctions callback sont créées. Ces fonctions permettent de réaliser différentes actions après la réception d’un message sur un *topic*. Le programme principal contient alors uniquement la fonction qui permet d’appeler ces fonctions callback dès la réception d’un message sur le *topic* associé.

Aussi, chacun des nœuds Robot possède 6 fonctions callback qui vont permettre de contrôler le robot à l’appel de fonctions définies dans le nœud Commande (cf paragraphe Nœud Commande et fiche Fonctions haut niveau disponibles). Les étudiants peuvent donc envoyer le robot dans une position prédéfinie ou dans une position qu’ils peuvent définir manuellement, monter ou descendre le bras, ou encore ouvrir ou fermer la pince.

Cependant, lors de la réalisation du nœud Commande, nous avons pu constater que l’appel de ces fonctions était bloquant et ne correspondait pas à la logique du réseau de Petri. Aussi, une dernière fonction a été implémentée pour pouvoir contrôler entièrement le robot. Lors de la réception du message associé, c’est-à-dire lorsque la fonction haut niveau correspondante est appelée dans le nœud Commande, on va envoyer successivement l’ordre de position prédéfinie, celui de commande du bras et celui de commande de la pince :

Nœud Tâche

Comme nous l’avons dit précédemment, nous avons choisi de représenter les produits par des couleurs, les objets physiques étant trop compliqués à gérer dans la simulation, notamment pour au niveau des robots qui sont censés prendre et déplacer ces objets.

Pour simuler le traitement des produits, par exemple l’assemblage de plusieurs composants et donc leur évolution, nous allons augmenter l’intensité de la couleur de base au fur et à mesure de la gamme de fabrication. Chaque couleur est associée à une valeur : par exemple, la valeur 10 est représentative d’un rouge clair, couleur de base du produit A ; 11 étant un rouge plus foncé représentatif du premier stade de traitement du produit.

La couleur du produit va donc évoluer au niveau des postes de travail, aussi appelés tâches. La cellule flexible contient 8 postes de travail, qui sont associés deux à deux à l’un des robots industriels présents dans le système.

Réalisation

Lorsqu’une navette contenant un produit arrive au niveau d’une zone de chargement /déchargement, le robot associé à cette zone doit se saisir du produit pour l’emmener à une tâche. Nous avons décidé qu’un produit présent à la zone Px ne pouvait être traité que par la tâche Tx. Le robot ne peut, par exemple, pas emmener un produit présent en P1 en T2 pour y être traité.

Pour pouvoir réaliser des traitements en parallèle les uns des autres, nous avons réalisé un nœud par tâche, c’est-à-dire 8 nœuds au total. Un nœud Tâche est associé à une table représentant les postes de travail dans notre simulation. Lors de la dépose d’une pièce sur une de ces tables, la table va prendre la couleur du produit qu’elle doit traiter. Ensuite, après une attente représentative du temps de traitement du produit, la valeur associée à la couleur va être incrémentée et la couleur de la table va changer pour signifier que le traitement est terminé et que le produit traité peut être récupéré pour effectuer la suite de la gamme.

Les fonctions du nœud ROS

Pour débuter le traitement d’un produit, celui-ci doit être posé sur la table. Pour cela, il est nécessaire de connaître la position du robot qui doit amener le produit au poste de travail, ainsi que l’état du bras et de la pince. En effet, il est nécessaire d’attendre que le robot soit dans la bonne position pour considérer que le produit est posé sur la table et donc qu’il peut être traité. Aussi, le nœud Tâche comprend trois fonctions callback qui permettent d’obtenir ces différents retours du robot lié à la tâche.

De plus, on y trouve également la gestion du traitement dans une fonction callback. Celle-ci est appelée dès la réception d’un message issu de la zone de chargement/déchargement associée à la tâche. A la réception d’un tel message, on connait donc le produit à traiter. Il nous faut ensuite attendre que le robot soit en position au-dessus de la table, bras descendu et pince ouverte, pour que le produit soit considéré comme posé et prêt à traiter. Pour simuler la pose du produit sur le poste de traitement, on colorise la table à l’aide de la couleur du produit à traiter d’un service V-Rep. Il nous faut ensuite attendre un certain temps représentatif de la durée de traitement du produit et défini dans le fichier de configuration. Pour cela, le temps de traitement est lu dans le fichier et sauvegardé dans une map pour pouvoir être récupéré en fonction du numéro du produit. Une fois cette durée écoulée, le produit est considéré comme étant traité et est incrémenté, donc la couleur de la table fonce. Le changement de couleur s’effectue à l’aide du service simRosSetIntegerSignal, le nom du signal correspondant au nom de la table à coloriser et sa valeur représentant la couleur souhaitée pour cette table.

Enfin, on trouve dans ce nœud une dernière fonction, qui permet simuler la prise du produit traité. Aussi, une fois que le traitement est terminé et en fonction de la position du robot, c’est-à-dire s’il se trouve au-dessus de la table avec le bras en position haute et la pince fermée, on va considérer que le produit traité a été retiré du poste de travail. Alors, la table reviendra à une couleur de base définie par la valeur 0, ne correspondant à aucun produit.