

FIGURE 1 – General structure of the algorithm.

1 Communication ordinateur-robot

Dans une démarche expérimentale, il est souvent requis d'implémenter les algorithmes développés, dans un environnement tel que Matlab, sur des systèmes robotiques déjà existants. La solution couramment utilisée est d'employer un nœud de calcul QNX sur lequel RTLAB est installé. Ceci requiert un processus d'adaptation autant au niveau de l'utilisation du logiciel qu'au niveau de l'adaptation du code Matlab, python c++ déjà développé. Bien souvent, le processus peut s'échelonner sur plusieurs semaines. Ceci motive alors la recherche d'alternatives permettant l'utilisation directe du code sur un système robotique. L'objectif étant de permettre l'utilisation de différents langage de programmation pour minimiser le temps d'adaptation. Une solution permettant l'utilisation du code Matlab, python, c++ et autre est alors mise sur pied.

1.1 Mise en contexte

Les solutions présentées dans ce rapport répondent toutes au besoin d'utiliser le code directement sur un montage robotique. Cependant, elle présentent toutes différents avantage et inconvénient. Comme processus de validation, les solutions ont été testées en répondant à la même problématique : l'utilisation d'un bras Jaco de Kinova directement par l'entremise de la connexion usb.

1.2 Définition des objectifs

Différents critères doivent être rencontrés pour qu'une solution soit acceptable et remplace le système déjà en place.

1. Permettre la communication bidirectionnelle.
2. Être compatible avec plusieurs systèmes robotiques.
3. Permettre une communication avec un délais de moins de 10 ms. (100 Hz)
4. Permettre une grande flexibilité dans le type de commandes qu'un utilisateur peut envoyer. (commandes articulaires, cartésiennes, force, etc.)

1.3 Solution de contrôle du bras Jaco développée sous Windows

Sachant que Windows est le système d'exploitation majoritairement utilisé au sein du laboratoire de robotique de l'université Laval, le besoin de développer une solution compatible Windows a été identifié. La stratégie employée est représenté dans la Figure ?? . Du point de vue global, la structure du système se résume en une chaîne de communication UDP entre différents programmes et le robot. Cependant, dans le cas de l'implémentation avec le robot Jaco, une API, *Applications*

Programming Interface, gère l'interface entre les actionneurs et les requêtes communiquées au robot via la connection USB. Dans le cas du Jaco, cette API permet l'utilisation de différentes fonctions rédigées en c++. Ceci cause problème lorsqu'un développeur tente d'utiliser un programme écrit en Matlab ou en python pour commander les articulations du robots. La solution à cet inconvénient est d'initier une instance Matlab via un script c++. Cette instance Matlab pourra ensuite exécuter des commandes que le script c++ lui envoie, tel que l'exécution de fonctions Matlab. Par la suite, il suffit d'utiliser directement l'API de kinova pour envoyer des commandes.

1.3.1 Exemple d'implémentation

Différents composants doivent être mis en place pour réaliser l'implémentation. Le présent exemple permet de configurer un projet visual studio pour contrôler un bras Jaco en communiquant avec Matlab.

Visual Studio Il est possible de télécharger VisualStudio sur le site web des développeurs.

Création d'un projet Un projet VisualStudio peut être créé en cliquant sur *file/new/project* dans la fenêtre de visual studio. Ensuite, sélectionner *Visual c++* dans le menu gauche de la nouvelle fenêtre et cliquez sur *Empty Project*. Dans le bas de la fenêtre, il est possible de nommer le projet et sélectionner un répertoire. Le répertoire choisi sera référencé comme étant le *workspace* ou *project directory* dans la plupart des tutoriels en anglais. Tout fichier placé dans ce répertoire fera partie du projet et pourra être référencé dans les scripts.

Définition du fichier principal du projet Pour être en mesure d'écrire le script, un fichier source doit être créé. Pour créer ce fichier, il faut cliquer droit sur l'item *Source Files* dans l'arbre du projet situé à la droite de la fenêtre Visual Studio. Ensuite, il faut sélectionner *add/new item* et cliquer sur *c++ File*. Il est ensuite possible de nommer le fichier dans le bas de la fenêtre.

Utilisation d'un joystick Pour être en mesure d'envoyer des commandes avec un joystick, la librairie SDL2 doit être téléchargée. Bien sûr, si le projet ne requiert pas de contrôleur joystick, cette étape n'est pas nécessaire. Après avoir décompressé le dossier téléchargé, lire le fichier VisualC.html pour les instructions sur la démarche à suivre pour utiliser la librairie (en particulier la partie parlant de l'utilisation avec Visual Studio). Les détails de l'utilisation pratique de cette librairie seront couverts dans une section subséquentes.

Lien avec Matlab La description de cette partie se fera en trois temps. Dans un premier temps, une explication des différentes composantes devant être mises en places pour monter l'environnement de développement sera présentée. Ensuite, l'installation des pilotes pour l'envoi de commandes au bras Jaco est expliquée. Finalement, une description de différentes parties du script permettant le contrôle du bras Jaco sera fait.

Environnement de travail Les différentes librairies, exécutables et les entêtes (*headers*) nécessaires à l'utilisation de Matlab via un script c++ sont disponibles dans le répertoire retourné lorsque l'on envoie les commandes suivantes dans la console Matlab :

1. `fullfile(matlabroot,'bin','win64')`.
2. `fullfile(matlabroot,'extern','include')`,
3. `fullfile(matlabroot,'extern','lib')`,

Pour les utiliser dans un projet Visual Studio, il est nécessaire de suivre les étapes suivantes :

- Il est tout d’abord très important de s’assurer que l’option x64 est sélectionnée dans le menu déroulant dans le haut et au centre de la fenêtre principale de Visual Studio.
- Aller dans *Project\’nom-de-votre-projet’ Properties...* dans le haut de la fenêtre principale de Visual Studio.
- Naviguer dans l’onglet Debugging de la fenêtre venant de s’ouvrir.
- Écrire *PATH=* suivi du résultat obtenu lors de l’envoi de la commande 1 dans le console Matlab, en omettant les guillemets,(devrait ressembler à *C :\Program Files\MATLAB\R2017b\bin\win64*) dans le section *Environnement*. Écrire Directement à la suite de la dernière entrée : *;%PATH%*
- Naviguer dans l’onglet VC++ Directories de la même fenêtre.
- **Pour les prochaines étapes** : Si un item était déjà présent dans cette ligne, il faut simplement séparer les entrées avec un ” ;”.
- Insérer le résultat obtenu lors de l’envoi de la commande 2 dans le console Matlab, en omettant les guillemets, dans le section *Include Directories*.
- Insérer le résultat obtenu lors de l’envoi de la commande 3 dans le console Matlab, en omettant les guillemets, dans le section *Library Directories*.
- Naviguer dans l’onglet *Linker* et le sous-onglet *Input* de la même fenêtre.
- Entrer au début d’*Additional Dependencies* l’entrée suivante : *libmx.lib ;libeng.lib ;libmex.lib ;libmat.lib ;*.

Pour tester l’environnement de travail, un script c++ est incluse avec les version récentes de Matlab. Il est accessible en entrant la commande :

— *edit([matlabroot\extern\examples\eng_mat\engdemo.cpp'])*;

Il faut seulement copier-coller l’entièreté du script dans le fichier source (.cpp) du projet Visual Studio. Le clic sur le bouton *Local Windows Debugger* lancera la compilation et devrait faire apparaître un invité de commande. La première compilation est normalement beaucoup plus longue que les compilations subséquentes.

Installation de Kinova SDK Pour être en mesure d’envoyer des commandes au robot jaco, il est nécessaire d’installer les pilotes fournis sur le site web de Kinova. Suivant l’installation de Kinova SDK, des directives doivent être suivies pour que le robot soit bien reconnu par le système d’exploitation. Ces directives sont disponible dans le répertoire d’installation de *KinovaSDK\Guides\Kinova SDK-User Guide* (souvent situé dans *Program Files x86*). Pour tester l’envoi de commandes au robot et valider l’installation, une interface graphique est disponible dans *KinovaSDK\GUI*. Des exemples de scripts c++ sont aussi disponibles dans le répertoire *KinovaSDK\Examples*.

À la suite du test, il est maintenant possible de combiner les informations des deux tutoriels pour faire fonctionner des scripts matlab, transférer des données de l’environnement de travail matlab vers le *stack* du programme c++ et vice versa.

Dans cette section, un survol de l’implémentation de l’environnement de développement permettant le contrôle du bras Jaco sous Windows avec l’option d’utiliser des scripts Matlab a été présenté. Bien que seulement le bras Jaco a été mentionné, il est toutefois possible d’utiliser les API *c++* d’autres systèmes robotiques tels que les bras de Universal Robot. Cependant, une autre option pour contrôler le bras UR5 sous windows est disponible. Cette dernière est présentée dans la section suivante.

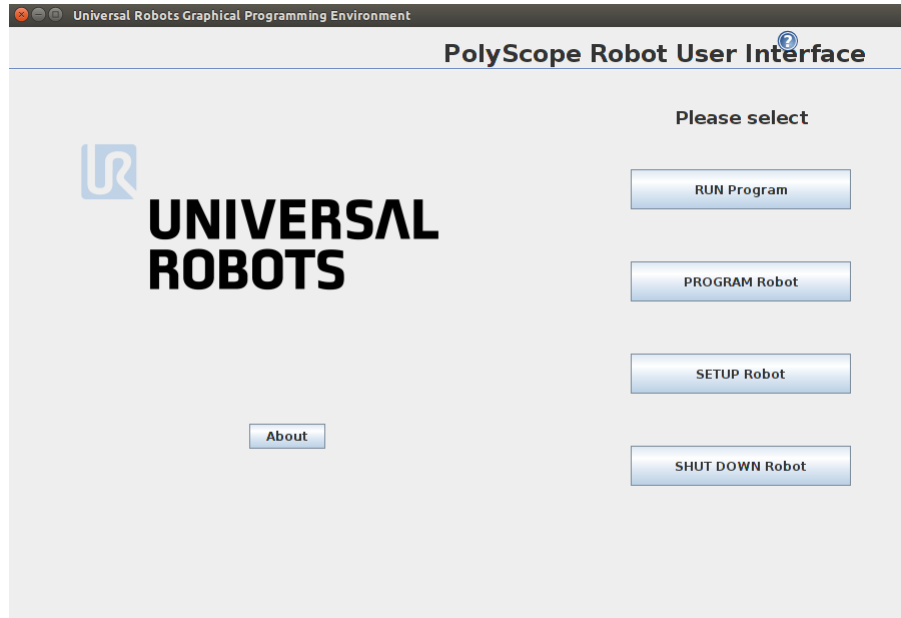


FIGURE 2 – Interface initiale d’URsim

1.4 Solution de contrôle du bras UR5 développée sous Windows

Le contrôle du bras UR5 sous windows présente quelques défis puisque la majorité des outils de développement pour cet équipement sont développés sur la plateforme Linux. Sachant que cette plateforme est souvent méconnue, la solution développée pour interagir avec le bras tentera de minimiser le travail requis avec Linux et fera le lien entre Matlab, URsim et un logiciel de visualisation ce nommant VRep. Ce dernier permet l’insertion et la visualisation de différents objets dans l’environnement du robot puisque URsim ne peut que représenter le robot lui-même. Tout d’abord, cette section expliquera l’installation de URsim et les différentes fonctionnalités utiles à la mise en place de l’infrastructure. Ensuite, le moyen de communication entre Matlab et URsim sera décrit. Finalement, l’interaction avec le logiciel de visualisation VRep sera expliqué.

1.4.1 URsim et son utilisation

URsim permet l’interface avec un robot virtuel ou un robot réel connecté sur le réseau. Ce logiciel permet la programmation de routines de base ainsi que la définition de fonctions plus complexe permettant, entre autre, la communication UDP avec d’autres logiciels. URsim, en soit, n’est disponible que sur Linux. Cependant, sur le site web de l’entreprise Universal Robots, une machine virtuelle sur laquelle URsim est déjà installée est disponible. Ceci permet de conserver l’environnement de développement Windows en exécutant URsim dans une machine virtuelle Linux. Le logiciel avec lequel cette machine virtuelle a été utilisée pour l’environnement de développement a été VmWare. Après avoir suivi les instructions d’installation fournies avec les fichiers disponibles sur le site web de Universal Robots, il est possible de lancer URsim. L’interface initiale d’URsim est présente dans la Figure suivante :

- Le bouton *RUN Program* permet de faire fonctionner des routines contenant différentes fonctions. Ces routines sont communément appelées *URscript*. Ces scripts peuvent varier en complexité allant d’une simple séquence de positions à atteindre jusqu’à un programme

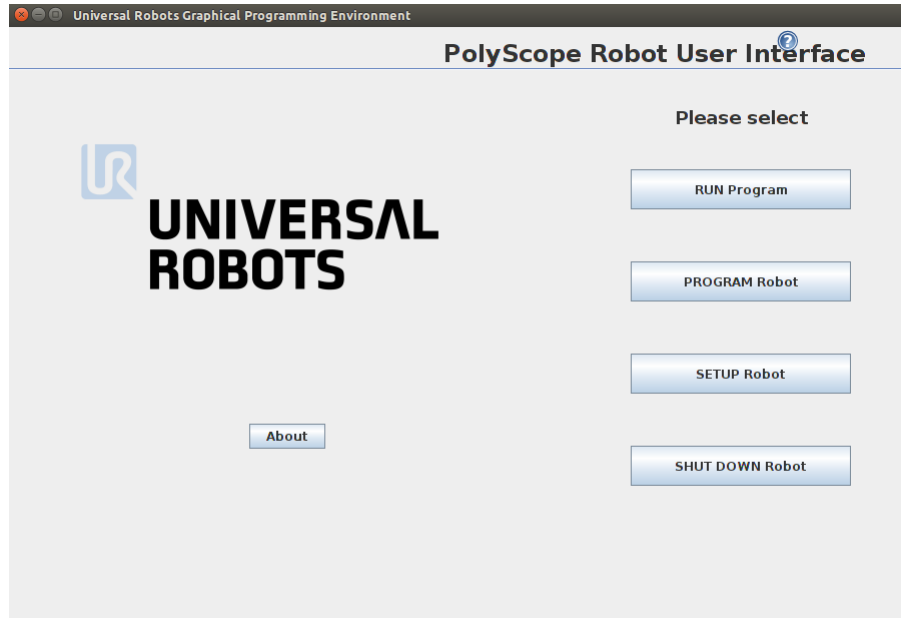


FIGURE 3 – Interface initiale d’URsim

complexe gérant de multiples sockets de communications UDP et répondant à des appels de fonctions listées dans l’API d’URsim.

- Le bouton *PROGRAM Robot* permet la rédaction ou la modification d’URscripts
- Le bouton *SETUP robot* permet la modification de paramètres tels que l’adresse IP d’un robot réel qu’un utilisateur voudrait contrôler via l’instance d’URsim installée sur la machine virtuelle.

Quand un utilisateur charge un URscript, il appui alors sur *PROGRAM ROBOT*, puis sur *Load Program* (ou sur *Empty program* si il n’y a pas de programmes déjà disponibles). Une fois cette opération complétée la fenêtre présente à la Figure ???. Ceci constitue l’interface de programmation d’URscripts. Un manuel de programmation URscript est inclu dans l’appendice ??