# Projet Automatique et Réseaux

### Contrôle distant d'un bras Manipulateur

2 mars 2018

## 1 Introduction

On souhaite réaliser le contrôle d'un robot via le réseau Internet par voies filaire et sans-fil utilisant un moyen de communication entre les différents modules (cf. figure 1) de type client-serveur UDP/IP.

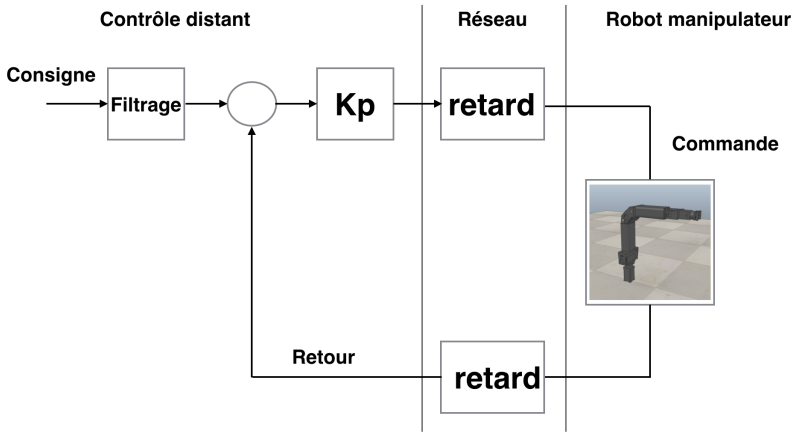


FIGURE 1 – Commande à distance d'un robot manipulateur

Le projet est décomposé en 2 sujets collaboratifs :

1. Mise en oeuvre d'un contrôle local et distant d'un robot manipulateur prenant en compte l'évolution du retard dans la boucle de commande ainsi que le développement d'une génération de trajectoire en position dans l'espace cartésien par l'utilisation du MGI à partir de la consigne de position de l'opérateur (touches flèches du clavier).
2. Le développement d'un simulateur de réseau avec la production de retards variables (constants, aléatoires, variables, ..) mais aussi d'informations de métrologie sur l'état du réseau (valeur moyenne du retard, écart type, prédiction).

Le travail entre les 2 groupes devra être collaboratif et comprendra tout au long du projet des réunions de synthèses et d'échanges sur l'avancement de chaque module. Le développement d'un protocole UDP/IP basé sur l'échange de données incluant un contrôle de flux devra être réalisé en commun avec les 2 groupes au début du projet. Ce protocole comprendra la prise en compte de la valeur du retard dans le contrôle du flux afin de gérer le temps d'attente entre 2 trames.

## 2 Contrôle distant

Il s'agira tout d'abord de réaliser en utilisant Visual C le contrôle en position distant basé client/-serveur UDP/IP soumis à un retard $T_r$ fixe et ensuite à un retard variable aléatoire pour le contrôle des 6 moteurs à courant-continus du bras manipulateur Robotis.

### 2.1 Plan de travail : premières étapes

1. Réaliser un premier client/Serveur UDP/IP sur votre machine capable d'échanger des données de type $float$ et $int$. Réaliser ensuite l'échange d'une structure de données comprenant à la fois des données de type $int$ et $float$. (cf ref. des documents section 5)
2. Réaliser un contrôle local en position du bras manipulateur sur V-REP.
3. Réaliser un deuxième client/Serveur de type UDP/IP capable de simuler un retard et une perte du réseau. Réaliser sa connexion avec le simulateur V-REP.
4. Insérer dans le premier client/serveur le contrôle distant du robot.
5. Connecter le premier client/serveur au deuxième client serveur afin de réaliser le contrôle distant.

### 2.2 Plan de travail : suite

La chaîne complète de commande distante étant maintenant réalisée et validée, vous pouvez aller plus loin.

*Du point de vue Commande :*

1. En utilisant une estimation de $G$ et de $\tau$ des actionneurs du bras manipulateur, déterminer en utilisant la relation de stabilité pour un retard constant maximal (donnée en annexe), les valeurs de $K_c$ stabilisant le contrôle pour des retards variant (hypothèse du retard constant, la variation doit être par étape toutes les 10 secondes par exemple) de $20ms$ à $1s$ (cf. schéma de commande de la figure 2).

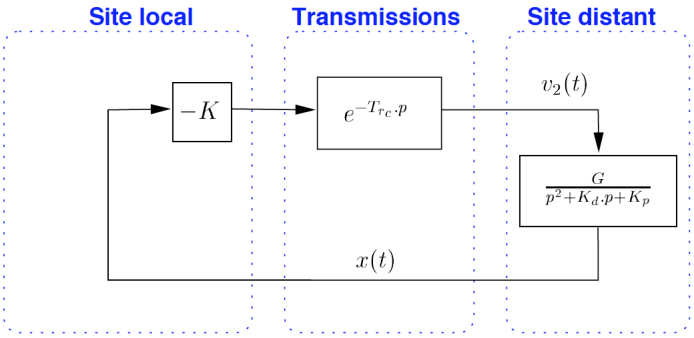


FIGURE 2 – Boucle de commande distante d'un moteur à courant-continu

2. Déterminer une loi adaptative du paramètre $K_c$ permettant à chaque instant de mesure du retard $T_r$ de stabiliser le contrôle du robot.

*Du point de vue Réseau :*

1. Il s'agit de mener une réflexion sur les mécanismes nécessaires pour la gestion du réseau pour la commande (ex : mesures du retard, contrôle de flux, etc..) ; et donc d'établir, en accord avec le groupe "commande", un protocole se rajoutant au dessus de UDP (niveau application) pour assurer les services désirés.
2. Le simulateur de pertes et retard doit être réaliste vis à vis d'une utilisation distante à travers Internet. Une première approche consiste en la simulation des lois de distribution aléatoire classiques : loi normale et loi de Poisson.

## 3 Simulateur V-REP

On utilisera pour la mise au point un simulateur physique V-REP incluant le modèle dynamique et géométrique du bras manipulateur Robotis.

La commande du robot est réalisée par un client/serveur TCP/IP local. Les fonctions de communication avec le simulateur via ce client/serveur ont été encapsulées dans un objet spécifique qui permet d'assurer la communication avec le simulateur.

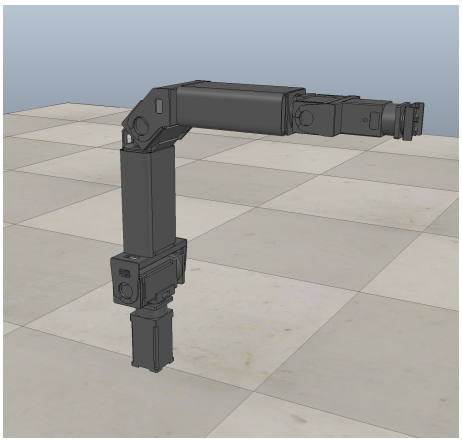

FIGURE 3 – Bras Manipulateur Robotis

### 3.1 Eléments fournis

Pour démarrer votre projet avec le simulateur V-REP, il vous est fourni deux éléments :

- Un fichier robotis_crayon.ttt qui contient le robot et qui devra être ouvert avec le simulateur V-REP.
- Un projet Visual Studio qui contient le programme de communication avec le simulateur.

Ces éléments sont stockés sur un Git. Vous trouverez dans la section 5, un tutoriel vous expliquant comment télécharger ces éléments.

### 3.2 Programme de communication avec le simulateur

Nous allons ici présenter les principaux points d'intérêt du programme de communication avec le simulateur dont une partie est présentée ci-dessous.

```
1  int main(int argc, char* argv[])
2  {
3      // Time simulation parameters
4      float t=0.0; // Start time (ms)
5      float tfinal=50000; // End time (ms)
6      float dt=10; // Control loop period (ms)
7      TimeMeasurement simulation; // Measures the overall execution time of the ...
           simulation
8      double simulation_duration = 0.0;
9      TimeMeasurement loop; // Measures the execution time of each loop
10     double loop_duration = 0.0;
11     vector<float> loop_durations(tfinal/dt, 0.0);
12     int loop_index = 0;
13
14
15     // Sensors and actuation variables
16     vector<float> q_s(6,0.0); // Position sensor
17     vector<float> gamma_s(6,0.0); // Torque sensor
18     vector<float> q(6,0.0); // Position actuation
19     vector<float> gamma(6,0.0); // Torque actuation
20
21     /*****************************************************************************/
22     // Put your own control variables here
23     vector<float> t_step(6,0.0); // Step response time
24     vector<float> q_ref(6,0.0);
25     q_ref[0] = 1.0;
26     q_ref[1] = -0.5;
```

```
27      /*****************************************************************************/
28
29      // Initialization
30      VREPSimulation sim; // Connextion to simulator and set up
31      if(sim.isConnected()) // Check connexion is valid before continuing to process
32      {
33          cout << "Connected" << endl;
34          // Start Simulation
35          simulation.start(); // Start time measurement
36          sim.startSimulation();
37
38          // Main Loop
39          while (t < tfinal)
40          {
41              loop.start();
42
43              ////////////// Read Sensors
44              sim.getJointPosition(q_s);
45              sim.getJointTorque(gamma_s);
46
47              /*************************************************************/
48              // Your control loop will go in there
49              proportionnalController(gamma, q_ref, q_s, t, t_step);
50              /*************************************************************/
51
52              ////////////// Write Commands
53              sim.setJointTorque(gamma);
54
55              ////////////// Update Time
56              loop_duration = loop.elapsed_ms();
57              if(loop_duration < dt)
58              {
59                  Sleep(dt-loop_duration);
60                  t+=dt; // Since we're not really on real-time execution we neglect ...
                        Sleep imprecision
61              }
62              else
63              {
64                  cout << "Loop duration violated : " << loop_duration << endl;
65                  t+=loop_duration;
66              }
67              loop_durations[loop_index] = loop_duration;
68              ++loop_index;
69          }
70
71          // Stop Simulation
72          sim.stopSimulation();
73          simulation_duration = simulation.elapsed_ms();
74          std::cout << "Simulation duration : " << simulation_duration << " ms" << endl;
75  }
```

Le programme vous est fourni avec une boucle d'exécution déjà implémentée. Votre contrôleur devra être réalisé au sein de cette boucle qui assure une périodicité de 10ms.

La variable **tfinal** (ligne 5) contrôle la durée de la simulation. Vous devrez la modifier en fonction de vos besoins.

La boucle principale (entre les lignes 39 et 69 du programme ci-dessus) commence par la lecture des capteurs (position des différents moteurs et couples appliqués grâce aux méthodes **getJointPosition** et **getJointTorque** respectivement).

Après les lectures capteurs, votre contrôleur (ou interface de communication avec le contrôleur distant) doit être implémenté. Un contrôleur basique est fournit à titre d'exemple pour vous permettre de valider la bonne communication avec le simulateur.

Enfin la fonction **setJointTorque** vous permet de transmettre les commandes aux différents moteur (sous forme de couple à appliquer).

Si vous souhaitez plus d'informations sur le fonctionnement des différentes fonctions de communication avec le simulateur, vous pouvez vous référer au fichier *VREPSimulation.h* qui documente les différentes fonctions.

### 3.3 Notes sur la dimension temporelle de l'exécution

Afin que le comportement du simulateur et de son interaction avec le contrôleur soit aussi proche que possible de la réalité, il a été décidé que le projet serait réalisé en temps "réaliste" et que le simulateur serait découplé temporellement du programme communiquant avec lui.

Pour pouvoir respecter cette hypothèse de temps "réaliste", la période d'exécution du simulateur et du programme de communication seront de 10ms. Pour assurer le bon fonctionnement de vos simulations, il vous faudra donc faire attention aux points suivants :

- Au niveau du simulateur, le respect de la période de 10ms est très limite. Il vous faut donc vous assurer, au moment de lancer votre simulation, que cette période est respectée. Ainsi, comme le montre la figure ci-dessous, la rubrique *Simulation time* indique le temps de simulation, le temps réel et un facteur de temps-réel. On considérera que la simulation respecte l'hypothèse de temps réaliste si ce facteur se situe entre 0.90 et 1.0. Assurez vous donc du respect de cette contrainte avant de lancer la simulation.

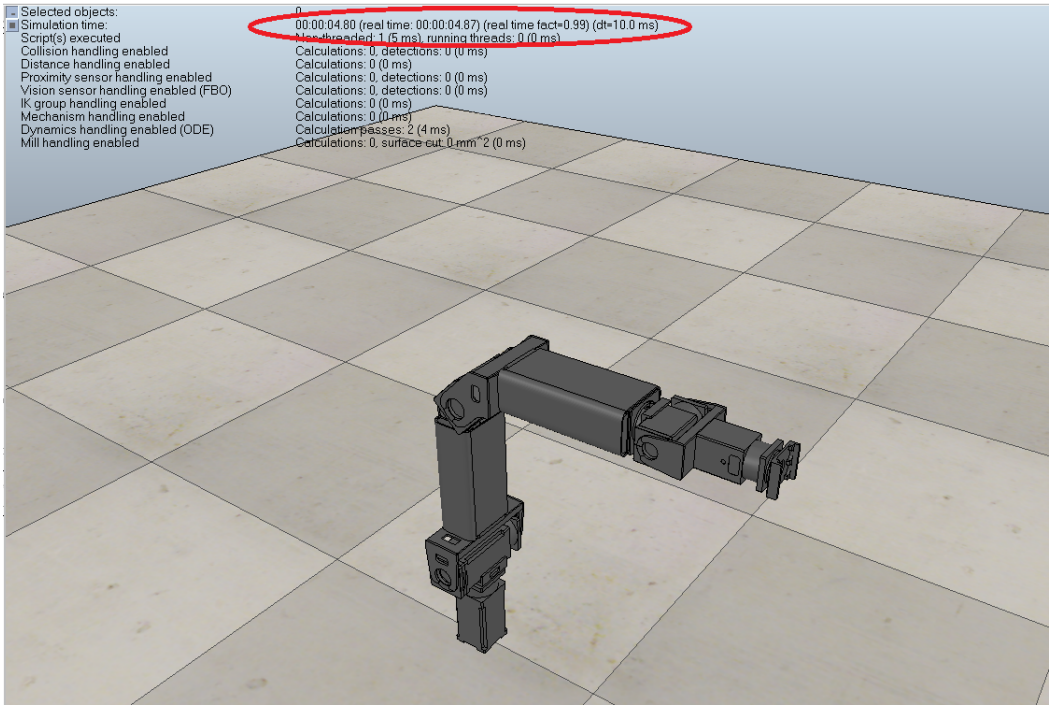


FIGURE 4 – Le facteur de temps-réel (cercle rouge) doit se situer entre 0.90 et 1.0.

- Dans le programme de communication, il vous faudra restreindre l'usage de fonctions d'affichage ou d'écriture dans des fichiers, très gourmandes en temps d'exécution. En cas de dépassement de la durée d'exécution de 10ms, un message vous sera automatiquement affiché.

Il est à noter que le non respect des périodes d'exécutions pourra entraîner un comportement instable du robot. En cas de comportement inattendu du robot, vérifiez donc le bon respect des contraintes temporelles au niveau du simulateur et de votre programme.

# 4 Stabilisation du contrôle d'un moteur à courant-continu avec retard fixe

On considère le modèle linéaire du second ordre d'un moteur à courant-continu suivant :

$$H_{BO}(p) = \frac{KGe^{-T_{rc}p}}{p^2 + K_d p + K_p} \tag{1}$$

On étudie les racines solutions de l'équation suivante :

$$1 + H_{BO}(p) = p^2 + K_d p + K_p + KGe^{-T_{rc}p} = 0 \tag{2}$$

On s'intéresse au cas limite lorsque $a = 0$ $p = j\omega$ :

$$\begin{cases} -\omega^2 + K_p + KG\cos(\omega T_{rc}) = 0 \ (\Re) \\ K_d\omega - KG\sin(\omega T_{rc}) = 0 \ (\Im) \end{cases}$$

on a :

$$\begin{cases} \omega^2 = K_p + KG\cos(\omega T_{rc}) \\ \left(\frac{K_d\omega}{KG}\right)^2 = \sin^2(\omega T_{rc}) = 1 - \cos^2(\omega T_{rc}) \end{cases}$$

On obtient l'expression d'une équation du second degré :

$$\cos^2(\omega T_{rc}) + \frac{K_d^2}{KG}\cos(\omega T_{rc}) + \frac{K_p K_d^2}{K^2G^2} - 1 = 0 \tag{3}$$

De la forme :

$$aX^2 + bX + c = 0 \tag{4}$$

Avec $X = \cos(\omega T_{rc})$, $a = 1$, $b = \frac{K_d^2}{KG}$ et $c = \frac{K_p K_d^2}{K^2G^2} - 1$

Si $\Delta < 0$ alors la solution est complexe alors que le résultat de la fonction $\cos(\omega T_{rc})$ est un réel. On va donc étudier le cas où $\Delta \geq 0$. On a :

$$\Delta = \frac{K_d^4}{K^2G^2} - 4\left(\frac{K_p K_d^2}{K^2G^2} - 1\right)$$

Avec :

$$\cos(\omega T_{rc})_{1,2} = \frac{1}{2}\left(-\frac{K_d^2}{KG} \pm \sqrt{\Delta}\right)$$

Avec

$$(\omega T_{rc})_{1,2} = \theta_{1,2} = \arccos\left(\frac{1}{2}\left(-\frac{K_d^2}{KG} \pm \sqrt{\Delta}\right)\right) \tag{5}$$

Lorsque l'on étudie des racines de l'équation :

$$\left(\frac{K_d\omega}{KG}\right)^2 = \sin^2(\omega T_{rc})$$

On multiplie et divise par $T_{rc}$, on obtient :

$$\frac{K_d\omega T_{rc}}{KGT_{rc}} = \sin(\omega T_{rc})$$

On remplace ensuite $\omega T_{rc}$ par l'expression précédente, on obtient :

$$(T_{rc})_{1,2} = \frac{K_d}{KG\sqrt{1 - \cos^2(\theta_{1,2})}}\arccos\left(\frac{1}{2}\left(-\frac{K_d^2}{KG} \pm \sqrt{\Delta}\right)\right) \tag{6}$$

Soit :

$$(T_{rc})_{1,2} = \frac{K_d}{KG\sqrt{1-\left(\frac{1}{2}\left(-\frac{K_d^2}{KG} \pm \sqrt{\Delta}\right)^2\right)}} \arccos\left(\frac{1}{2}\left(-\frac{K_d^2}{KG} \pm \sqrt{\Delta}\right)\right) \tag{7}$$

Cette solution correspond à un cas limite de stabilité, c'est-à-dire $T_{rcmax}$. A priori nous pouvons faire l'hypthèse que la stabilité sera assurée pour le cas où

$$T_{rc} < T_{rcmax}$$

Avec $T_{rcmax} \in \mathbb{R}^{*+}$. Une solution avec un retard limite complexe, négatif ou nul n'est pas acceptable. On prendra la solution la plus petite appartenant au domaine de définition.

# 5 Références

- Tutoriel : Créer une clé Git et récupérer le projet de communication avec le simulateur : Y :\PublicMEA\dubreuil\GIT\tuto-git-0.1.pdf
- Tutoriel : Les bases sur les socket en C/C++ : `http://broux.developpez.com/articles/c/sockets/`