

Activités proposées par :

- M. Emmanuel Broutier, professeur au lycée Michelet, CPGE*
M. Alain Caignot, professeur au lycée Stanislas, CPGE
M. Vincent Crespel, professeur au lycée St Louis, CPGE
M. Marc Derumaux, professeur au lycée Saint Louis, CPGE
M. Cédric Lusseau, professeur au lycée Hoche, CPGE
M. Gilles Moissard, professeur au lycée Janson de Sailly, CPGE
M. Pascal Serrier, professeur au lycée Benjamin Franklin, CPGE
M. Sébastien Villacampa, professeur au lycée Gustave Eiffel, CPGE
M. David Violeau, professeur au lycée Janson de Sailly, CPGE

TP3 : Acquérir et piloter des systèmes à l'aide de cartes Arduino et d'une Toolbox Xcos dédiée

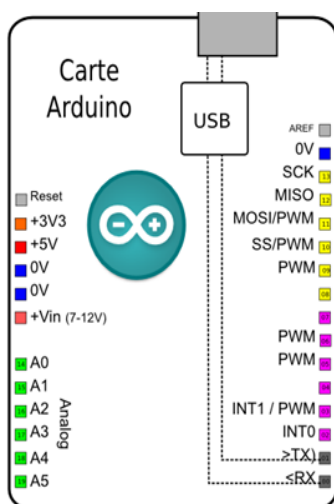
Présentation

Cette séance d'une durée de 45 minutes va montrer les possibilités de simulation « Hardware in the loop » du logiciel Scilab (version 5.4.0) et de son extension Xcos couplée à un Arduino. En effet, le module Arduino, développé pour Démosciences 2012, permet d'intégrer dans l'outil de simulation la commande d'un Arduino Uno ou Mega.

Pour l'instant, seules les fonctions les plus courantes ont été implémentées : lecture / écriture d'une entrée / sortie logique, lecture / écriture d'une entrée / sortie analogique, commande d'un moteur continu, d'un servomoteur et d'un moteur pas à pas. Cette Toolbox est encore en phase de développement, la version actuelle est la 1.0.

Présentation succincte de l'Arduino Uno

Avant de commencer, intéressons-nous à l'Arduino Uno (la Mega fonctionne de la même manière mais dispose de plus d'entrées-sorties).



Identifier les différents pins sur votre carte Arduino Uno :

14 (+6) Entrées – Sorties logiques (Pin Digital de 0 à 13) :

- Série asynchrone (0 : Rx et 1 : Tx) (les pins 0 et 1 ne seront donc pas utilisables)
- 2 Interruptions externes sur 2 et 3 (utilisées pour le codeur en quadrature)
- Sortie 13 couplée à une LED sur la carte

6 Entrées analogiques (A0 à A5) :

- Tension d'entrée < tension de référence (5v ou 1,1v ou AREF : référence externe)
- 6 CAN 10 bits (plage de 1024)
- Peuvent aussi fonctionner comme des E/S numériques

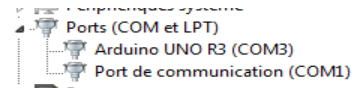
6 Sorties « Analogiques »


- 6 PWM sur 3,5,6,9,10 et 11 construites sur les pins d'entrées-sorties logiques

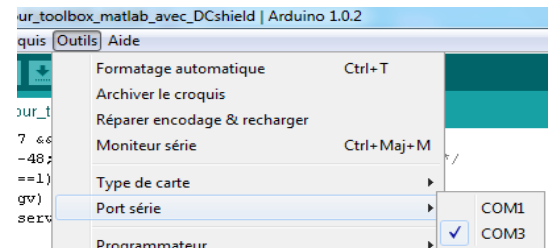
Un programme permettant le fonctionnement de la Toolbox doit être chargé dans l'Arduino. Celui-ci a été développé en langage de programmation (semblable au C++) en utilisant le soft gratuit téléchargeable en ligne à l'adresse <http://Arduino.cc/en/Main/Software>. Le guide d'installation est aussi disponible (<http://Arduino.cc/en/Guide/HomePage>)


Comme les mémoires sont de type Flash, le programme reste « indéfiniment » en mémoire, même sans alimentation. Pour charger le programme, nous utilisons la liaison USB qui fonctionnera comme un port série. Nous avons installé le driver qui se situe dans le répertoire du soft Arduino téléchargé précédemment.

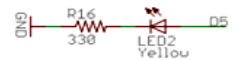
- Brancher l'Arduino
- Ouvrir le gestionnaire de périphérique dans le panneau de configuration et vérifier la présence du matériel dans Ports (COM et LPT).
- Retenir le port COM utilisé. Le numéro port COM doit être inférieur à 9. Si ce n'est pas le cas, il faut le changer dans les paramètres avancés.



-  Lancer le logiciel Arduino puis vérifier dans le menu déroulant « outils » que le type de carte et le port COM sont correctement sélectionnés.



- Ouvrir l'exemple blink dans fichier>exemples>01.basics puis téléverser (charger) le programme dans l'Arduino.  Téléverser avec un programmeur
- Vérifier que cet exemple permet de faire clignoter la diode associée au pin 13 à une fréquence de 0,5 Hz.
- Modifier le programme pour allumer la diode fournie (diode associée à une résistance) que vous brancherez sur un pin digital de votre choix (noir sur la masse GND et rouge sur la sortie logique de votre choix). Téléverser et vérifier le fonctionnement.



Installation de la Toolbox

La Toolbox peut s'installer à partir du gestionnaire de module ATOMS de Scilab. La gestion de la liaison série est intégrée à la toolbox. Elle a été optimisée au maximum mais pour l'instant elle ne fonctionne qu'avec Scilab 32bits.

Pour faire fonctionner la Toolbox, il faut aussi charger le programme dans l'Arduino.

- Ouvrir le fichier Toolbox_Arduino.ino qui se trouve à l'adresse indiquée dans l'aide puis le téléverser. Vérifier que le téléversement s'est bien terminé puis quitter.

L'Arduino est prêt pour être interfacé avec Scilab et Xcos grâce à la Toolbox que nous allons maintenant étudier.


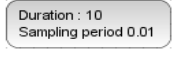


Démarrage avec la Toolbox Arduino : clignotement d'une LED et entrées-sorties logiques

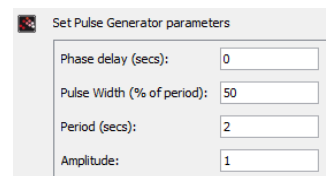
Démarrer le logiciel Scilab puis, lorsque la console Scilab est lancée, activer le module Arduino (situé dans le menu déroulant « module »). Démarrer l'extension Xcos.

Deux fenêtres s'ouvrent : le « Navigateur de palettes » (qui contient les différents icônes) et la feuille de travail. S'il n'y a pas le navigateur de palette, vous pouvez l'ouvrir depuis le menu déroulant « affichage » de la fenêtre Xcos.


Pour vérifier le bon fonctionnement, nous allons tenter de refaire clignoter une diode (exemple de base en programmation de microcontrôleur).

- A partir du module Arduino, glisser dans la fenêtre Xcos :

	Le bloc de configuration du port : double-cliquer dessus pour définir le port COM auquel est relié l'Arduino
	Le bloc d'échantillonnage : il permet de spécifier le temps de simulation ainsi que la fréquence d'échantillonnage. Laisser la durée et la période d'échantillonnage définie par défaut.
	Le bloc commandant une sortie logique (digital) de l'Arduino : en double-cliquant lui attribuer le pin digital 13
	Une entrée PULSE : à partir du module CPGE et à configurer pour avoir un clignotement de 1 seconde

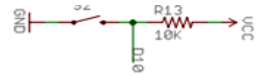


- Relier l'entrée Pulse et la sortie logique puis lancer la simulation. Les deux diodes Rx et Tx sur la carte doivent indiquer la communication série entre Xcos et l'Arduino tandis qu'une troisième diode reliée à la pin 13 doit clignoter.

- Vous pouvez maintenant remplacer la sortie logique 13 par une autre de votre choix et branche la diode fournie sur cette pin. Validez le fonctionnement.
- Remplacer l'entrée Pulse par un bouton poussoir en utilisant le bloc digital read 

- Brancher le bouton poussoir en reliant la masse (le fil noir) à GND (niveau logique bas), le fil rouge au +5v (niveau logique haut) et le fil bleu sur la pin digitale de votre choix, en concordance avec votre déclaration dans le bloc digital read.

Remarque : Nous avons câblé le bouton poussoir en utilisant une résistance de tirage (pull up) selon le schéma ci contre.



- Relier l'entrée et la sortie logique, correspondant au bouton poussoir et à la diode, de façon à l'allumer à chaque appui.

Vous savez désormais interfacer des entrées-sorties logiques de l'Arduino avec le logiciel de simulation

Acquérir une grandeur analogique




Dans cette partie, nous allons utiliser un potentiomètre afin de générer une tension V_a variant entre 0 et 5v (principe du pont diviseur de tension). On pourrait acquérir la tension de n'importe quel capteur, mais il faut faire attention à ne pas mettre plus en entrée que la tension de référence (5v par défaut).

L'acquisition analogique se fait grâce à un CAN (Convertisseur Analogique Numérique) de 10 bits : c'est à dire que la plage 0-5v est convertie en $2^{10} = 1024$ nombres numériques, soit une plage allant de 0 à 1023.

La résolution est donc de 4,9 mV.

L'application flash <http://fisik.free.fr/ressources/CANflash.swf> permet de comprendre le principe de fonctionnement d'un CAN 3bits.

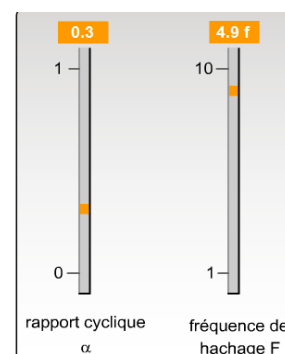
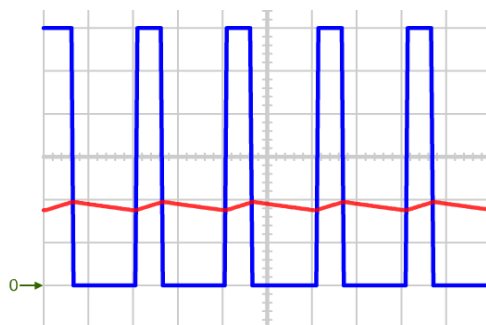
- Brancher les fils rouge et noir respectivement au +5v et à la masse accessible sur les pins de l'Arduino.
- Brancher le fil bleu (V_a) sur une des entrées analogiques.
- Ouvrir l'exemple potentiomètre et double-cliquer sur l'entrée analogique  pour spécifier votre numéro de pin.
- Lancer la simulation et faire tourner le potentiomètre.
- Vous devez normalement voir une courbe variant de 0 à 1023 selon l'angle du potentiomètre.

Maintenant que vous savez acquérir une grandeur analogique, essayons de commander les sorties analogiques de l'Arduino.

Commande de sorties analogiques : PWM


Les sorties analogiques de l'Arduino Uno sont disponibles sur les pins de sorties logiques 3,5,6,9,10 et 11. Parler de sorties analogiques est donc un peu un abus de langage. En effet, pour générer cette sortie en minimisant les pertes d'énergie, l'Arduino utilise des PWM (Pulse With Modulation, MLI en français).

Le principe est simple : générer un signal créneau de 0 (État logique bas) à 5v (État logique haut) dont le rapport cyclique est variable. Comme la fréquence de ce signal est élevée (environ 490Hz dans ce cas), si le système connecté à la sortie PWM est « lent », il ne voit à ses bornes que la tension moyenne du signal PWM (il fonctionne comme un filtre).



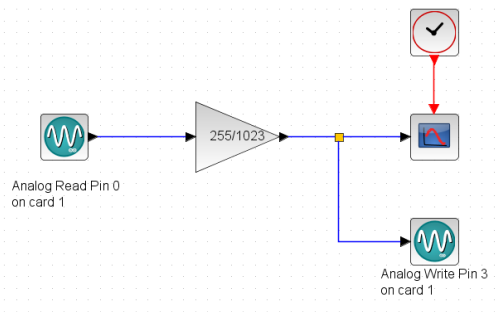
Courbe issue d'une application flash présentant le fonctionnement d'un hacheur série <http://fisik.free.fr/ressources/hacheurserie.swf>

Le but est de réaliser un variateur de lumière à partir de la consigne issue du potentiomètre. Nous allons utiliser pour cela la diode. C'est un système très rapide (le temps de réponse est de l'ordre de la nanoseconde) elle va donc clignoter à la fréquence de 490Hz avec un rapport cyclique variable. Ce sont nos yeux qui vont jouer le rôle de filtre en ne retenant que la valeur moyenne de l'intensité lumineuse perçue.

- Ajouter sur le schéma Xcos précédent un bloc sortie analogique  disponible dans la Toolbox Arduino.
- Double-cliquer pour spécifier que cette sortie sera associée au pin PWM de votre choix et brancher la diode en fonction.

L'entrée du bloc sortie analogique correspond à la valeur du rapport cyclique. Cette valeur peut varier de 0 (0%) à 255 (100%). Il est possible de mettre une valeur supérieure à 255, mais cela n'a pas de sens car le rapport cyclique restera à son maximum (100%). Nous avons vu précédemment que la consigne issue du potentiomètre varie de 0 à 1023. Il faut donc adapter cette consigne pour profiter au maximum de la plage de réglage allant de 0 à 255.

- Ajouter un gain de 255/1023 disponible dans la Toolbox CPGE.
- Relier l'entrée potentiomètre à la sortie PWM en passant par le gain puis simuler.



Vous pouvez maintenant commander l'intensité lumineuse de la diode en agissant sur le potentiomètre.


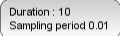


Commande en boucle ouverte d'un moteur à courant continu

Pour réaliser la commande d'un moteur, nous allons utiliser un hacheur externe. C'est nécessaire car l'Arduino ne peut pas délivrer suffisamment de puissance pour alimenter un moteur.

Le hacheur fonctionne selon le même principe que le signal PWM précédent : il hache la tension issue de l'alimentation externe grâce à des transistors ; la tension moyenne dépend alors du rapport cyclique. C'est justement le signal PWM de l'Arduino qui est utilisé pour commander ces transistors.

Il est possible d'utiliser des cartes toutes prêtes qui permettent de relier simplement l'Arduino, le hacheur externe et le moteur. Il n'est pas beaucoup plus compliqué de créer son propre hacheur à partir de circuits imprimés comme le L293D.

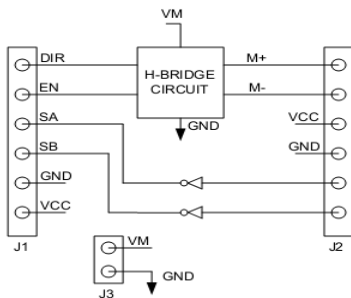
Nous allons utiliser dans un premier temps la carte motorshield révision 3.

- Brancher cette carte sur l'Arduino. Le moteur et le potentiomètre sont déjà branchés.
- Ouvrir un nouveau fichier Xcos et ajouter le bloc de configuration du port  et le bloc échantillonnage  (Duration : 10, Sampling period 0.01)
- Ajouter un bloc entrée analogique  et la régler pour qu'elle donne l'information issue du potentiomètre
- Ajouter un bloc moteur continu  et régler sur la carte motorshield (option 2) puis sur le moteur 1
- Ajouter un gain d'adaptation du CAN vers PWM de 255/1023
- Lancer la simulation.

Vous commandez votre moteur en boucle ouverte à l'aide du potentiomètre. Vous pouvez vous rendre compte qu'il existe une forte tension de seuil avant la rotation. Elle est liée à d'importants frottements secs dans le réducteur.

Récupération des informations issues d'un codeur

La carte Motorshield alloue automatiquement les pins de pilotage des moteurs et utilise malheureusement un des pins d'interruption qui seront nécessaires pour faire fonctionner les codeurs. Nous allons donc maintenant utiliser la carte de commande PmodHB5 qui est plus permissive et dont vous voyez le schéma ci-dessous.



Le signal PWM pour la commutation du pont en H sera envoyé sur l'entrée EN. Il faut aussi spécifier le sens de rotation avec l'entrée DIR.

SA et SB correspondent aux signaux des deux capteurs en quadrature de phase.

GND : VCC est l'alimentation des capteurs.

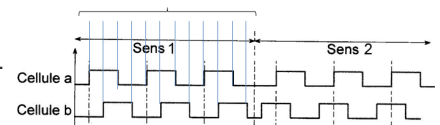
Vm et GND correspondent à l'alimentation externe (chaîne d'énergie). Nous avons placé une fiche femelle pour plus de commodité.

Grâce à la quadrature des deux signaux, on arrive à déterminer le sens de rotation.

Nous allons avoir différentes options pour compter les impulsions :

- x1 : incrémente/décrompte (selon le sens de rotation) le compteur à tous les fronts montants de la voie A.
- x2 : incrémente/décrompte (selon le sens de rotation) le compteur à tous les fronts montants et descendants de la voie A.
- x4 : incrémente/décrompte (selon le sens de rotation) le compteur à tous les fronts montants et descendants de la voie A et de la voie B

Sur nos moteur, nous avons trois périodes pour chaque cellule par tour d'arbre moteur. Nous pouvons donc en x4 obtenir 12 incréments par tour de l'arbre moteur.



Comme nous avons un réducteur de rapport 1/53, cela doit nous donner 636 incréments par tour en sortie du réducteur.

Pour faciliter le câblage, nous utilisons une shield de prototypage associée à une plaquette d'essai sans soudure.


- [Brancher cette carte shield pré-cablée sur l'Arduino, puis ouvrir le fichier encodeur.zcos](#)

Vous pouvez constater que la direction (DIR) est reliée à la pin 12, que la vitesse (EN = signal PWM) est reliée à la pin PWM compatible 9.

- En double-cliquant sur le bloc moteur DC , changer le type de carte de commande pour PmodHB5 (option 3), puis choisir le pin de PWM (EN) et le pin de direction (DIR). *Rappel : le pin de PWM doit forcément être 3,5,6,9,10,11.*

Pour faire fonctionner un codeur, nous avons besoin d'entrées en interruption. Ce sont des entrées capables d'arrêter le programme principal lors d'un changement d'état afin d'exécuter un sous-programme (dans notre cas incrémenter un compteur). Les pins 2 et 3 sont les seuls pins de l'Arduino Uno supportant les interruptions (il y en a plus sur le Mega). Pour faire fonctionner un codeur en x4, il faut que les deux voies soient câblées sur les pins d'interruption. Par contre en x1 ou x2, seule une voie doit être branchée en interruption, l'autre peut être reliée à une entrée logique classique.

Vous remarquerez que les voies SA et SB ont été ici câblées sur les pins 2 et 3 (compatibles interruptions) et que l'alimentation des codeurs a bien été reliée aussi.

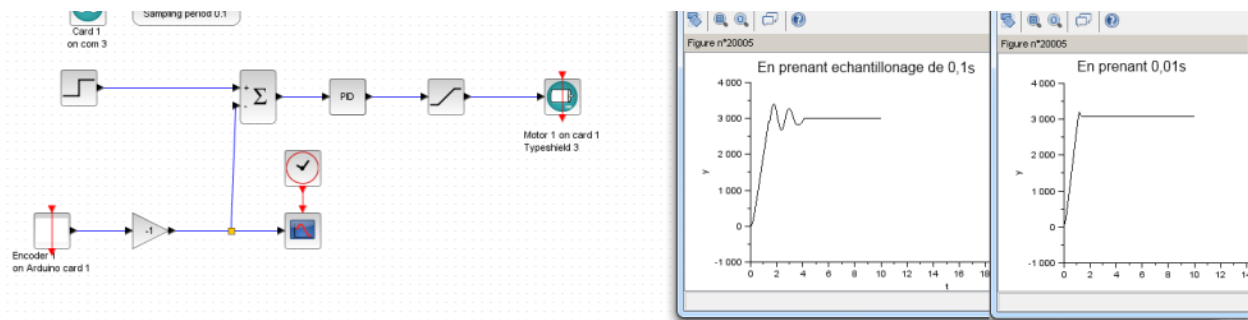
- En double-cliquant sur le bloc encoder , vous pouvez choisir le numéro du codeur (1) et le mode de fonctionnement (x1, x2, x4). Ensuite vous devez spécifier les bons pins pour la voie A (forcément 2 ou 3) et la voie B, qui permet de trouver la direction.
- Le potentiomètre est branché sur la pin analogique 2 : vérifiez qu'il est bien déclaré dans le programme.
- Pour l'alimentation du pont en H, branchez la fiche d'alimentation et vérifiez que le chargeur 12v est branché dans la prise.

Un gain de 127/1023 a été ajouté entre la consigne issue du potentiomètre et le moteur. En effet, les moteurs admettent une tension nominale maximum de 6v, or l'alimentation du chargeur est en 12v. Il faut donc envoyer uniquement 50 % de la tension max 12v, ce qui correspond à un rapport cyclique max de 127 (255*0,5) pour une valeur de consigne max de 1023.

- Lancer la simulation et tester la commande avec le potentiomètre.
- Choisir une valeur fixe de commande avec le potentiomètre et vérifier les rapports de multiplication x1, x2, x4 en regardant la courbe obtenue.

Asservissement en position d'un moteur

Vous avez maintenant tous les outils pour tenter de faire l'asservissement en position du moteur. Il faudrait bien sûr identifier les caractéristiques du moto-réducteur (on sait déjà que les frottements secs sont très importants) afin de définir un bon correcteur.



Pour l'instant l'utilisation simultanée de l'encodeur, de la lecture analogique et de la commande du moteur peut poser des problèmes lors de la transmission des données, surtout pour des fréquences d'échantillonnage faibles. Nous essayons de régler ce problème.

Bonus : Utilisation d'un servomoteur

Un servomoteur est un ensemble constitué d'un moteur électrique, d'un réducteur, d'un potentiomètre et d'une partie électronique analogique ou numérique pour la commande. C'est donc un asservissement de position (ou de vitesse pour les servomoteurs à rotation continue). L'utilisateur dispose de trois fils :

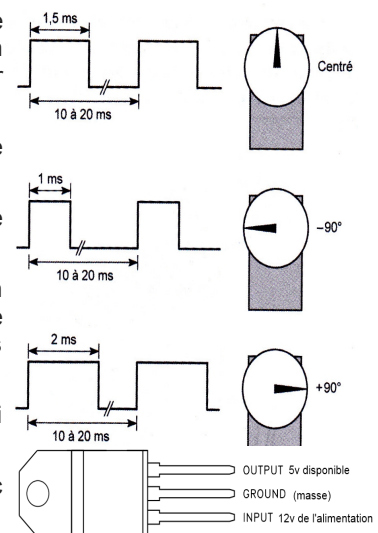
Le rouge et le noir correspondent à l'alimentation du servomoteur qui doit généralement être comprise entre 4,8 et 6v.


Le fil restant (souvent jaune ou blanc) sert à transmettre la consigne au servomoteur sous forme d'impulsions espacées de 10 à 20ms.

Le codage de ces impulsions est fait de telle façon qu'une impulsion de 1,5ms correspond à la position centrée (de repos), une impulsion de 1ms correspond à un angle de 90° dans le sens trigonométrique et enfin une impulsion de 2ms correspond à un angle de 90° dans le sens horaire. Toutes les autres largeurs d'impulsion donneront des valeurs intermédiaires.

Pour un servomoteur à rotation continue, la largeur des impulsions donne la vitesse de rotation ainsi que le sens.

Il ne faut pas utiliser l'alimentation de la carte Arduino pour alimenter le servomoteur. Il faut donc utiliser un régulateur de tension de 5v relié à l'alimentation 12v.



- Utiliser la carte protoshield pour brancher le régulateur : masse de la fiche d'alimentation sur la pin du milieu et +12v sur la pin de gauche (regarder la figure ci-dessus). Brancher le servomoteur avec le fil noir à la masse, le fil rouge au +5v du régulateur et le fil restant à la pin digital 10.
- Créer un fichier de simulation Xcos avec les blocs classiques (configuration et échantillonnage) et un potentiomètre.
- Ajouter un bloc servomoteur  et le configurer comme étant le numéro 1 (associé à la pin 10, le numéro 2 est associé à la pin 9). L'entrée de ce bloc est un nombre compris entre 0 et 180 (90 correspond à la position neutre. Il faut donc multiplier la sortie du potentiomètre par un gain de 180/1023).
- Relier les éléments et vérifier le bon fonctionnement.

Conclusion

On se rend compte qu'il est maintenant possible de faire de l'acquisition et de la commande de systèmes directement avec Xcos en utilisant la Toolbox Arduino.

Bien entendu, les performances n'égaleront pas celles des « cartes temps réel » proposées par les différents constructeurs. On s'en rend compte lorsque l'on demande beaucoup de choses en même temps avec une fréquence d'échantillonnage faible. Nous avons déjà optimisé la liaison série. Pour améliorer cette Toolbox, nous envisageons plusieurs possibilités, comme notamment utiliser des protocoles de communication plus évolués comme TCP ou UDP associés à des liaisons Ethernet ou Wifi beaucoup plus rapides que la liaison série mais des contraintes inhérentes à ceux-ci rendent la tâche difficile. Une autre piste qui va être étudiée est la génération du code cible Arduino par Scilab. Cela permettrait de supprimer les aller-retours pendant la simulation et d'avoir un échantillonnage indépendant de la liaison série.