POLYTECH LILLE

Université Lille1 Sciences et Technologies

Rapport de projet

Console de contrôle pour robots déformables

TIRABY Céline

EL BEZ Ghada

Département: IMA-SA

responsable laboratoire: DEQUIDT Jérémie

tuteur école: REDON Xavier

2014/2015

Remerciements

Avant toute autre chose, nous tenons à remercier l'Institut National de Recherches en Informatique et Automatique (INRIA) qui nous a accueilli dans ses locaux et qui nous a donné les moyens techniques pour réaliser notre projet du semestre 8 dans de bonnes conditions.

Nous remercions tout particulièrement M.Dequidt Jérémie, notre tuteur de projet, qui nous a encadré tout au long de ce projet avec le souci de nous apporter une formation complète et une aide précieuse.

Nous souhaitons par ailleurs remercier M.Mario Sanz-Lopez de l'équipe DEFROST (DEFormable RObots Simulatio Team) qui nous a beaucoup aidé pour mener à bien ce projet.

Enfin, nous tenons à souligner que l'ambiance générale du projet a été très agréable et propice au travail.

Sommaire

Introduction

Dans cadre de notre projet du semestre 8, nous avons choisi de travailler le développement d'une console de commande de robots déformable au sein du laboratoire INRIA de Lille. Les créneaux horaires alloués à la réalisation de ce projet été de à peu-près 8h par semaine pendant 12 semaines. Notre mission principale était d'implémenter un code générique utilisable quelque soit le type du robot déformable qu'on souhaite commander.

1 Présentation générale du projet

1.1 Présentation de l'Inria de Lille et de l'équipe Defrost

Le centre de recherche **Inria Lille** - Nord Europe, situé à la Haute Borne, compte 17 équipes-projet. Celles-ci se concentrent sur les Sciences et Technologies de l'Information et de la Communication.

Elles conçoivent des logiciels innovants pour le commerce ou la logistique, ou encore, mettent au point des simulateurs médicaux ou des interfaces facilitant l'interaction entre l'humain et la machine. Elles orientent leurs recherches sur quatre grandes thématiques :

— Internet des données et Internet des objets
— Génie logiciel pour systèmes éternels
— Modèle patient dynamique
— Couplage perception/action pour l'interaction homme-machine

Nous avons réalisé notre projet au sein de l'équipe **Defrost** (DEFormable RObots Simulation Team). Parmi les dix-sept équipes du centre de Lille, l'équipe Defrost oriente ses recherches sur le thème Images, modèles et algorithmes pour la médecine et les neurosciences. Le rôle de cette équipe-projet est de mener des recherches sur des problèmes scientifiques liés à la simulation médicale numérique. Pour ce faire, ils utilisent une plate-forme logicielle : SOFA et collaborent avec des instituts chirurgicaux.
L'objectif commun des projets de l'équipe est de fournir des outils de diagnostics, de planification ou de guidage basés sur la simulation physique.

1.2 Présentation des robots déformables

Contrairement aux robots rigides, le nombre de degrés de liberté (DDL) des robots déformables est infinie. D'une part, un grand avantage consiste à multiplier les actionneurs et leurs types dans la structure pour élargir la taille de l'espace de travail. En revanche, ces actionneurs sont couplés ensemble par la déformation du robot ce qui rend le contrôle très

délicat. En outre, si la collision de leur environnement direct, les robots peuvent se déformer et aussi déformer l'environnement, ce qui complique encore plus le contrôle.

1.3 Sujet du projet et définition du cahier des charges

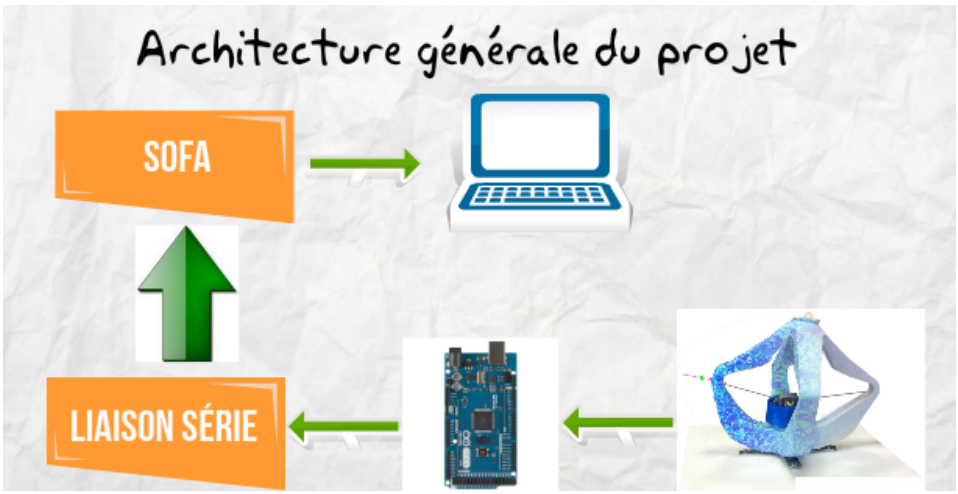
Le but de ce projet était de répondre au besoin de l'INRIA concernant l'implémentation d'un code unique pour l'utilisation de l'ensemble de ses robots déformables.

En effet, lors de la création d'un nouveau robot, il était nécessaire à chaque fois de modifier l'ensemble des codes d'interfaçage entre le prototype et la simulation, même pour ajouter un seul actionneur. Le besoin du laboratoire se résume donc à pouvoir réaliser un code unique, destiné à l'usage de n'importe quel robot.

Ainsi le code devait répondre au besoins suivants :

- Contrôler un robot dont le nombre d'actionneurs peut être variable
- Contrôler un robot dont le type d'actionneurs peut être variable
- Contrôler un robot dont les types d'actionneurs peuvent être mixtes (des servos moteurs et des pneumatiques par exemple)
- Permettre d'obtenir des "feedbacks" des différents actionneurs

L'architecture de ce projet est la suivante :



On voit donc que ce projet est composé de trois partie principales :

- La partie simulation: récupération des données du logiciel Sofa
- La partie gestion du port série
- La partie gestion de l'Arduino

Vous trouverez ci-dessous les différents détails et objectifs concernant ces parties :

Partie simulation

Le but est de récupérer les informations ,concernant les différentes forces et pressions appliquées, du logiciel de simulation Sofa, afin de pouvoir les transmettre par la suite via la liaison série.

Partie interface port série

Dans cette partie les objectifs à atteindre seront les suivants :

- Gérer les vitesses d'envoi des différentes informations
- Adapter les différents types de données
- Gérer la communication (les périodes d'échantillonnage)
- Permettre l'échange de données dans les deux sens

Partie Arduino

La partie Arduino devra permettre d'obtenir :

- Des informations sur le nombre d'actionneurs
- Des informations sur le type de ces actionneurs

2 Réalisation du projet

2.1 Prise en main des parties logicielles et matérielles

Afin de commencer le projet dans les meilleures conditions, il était nécessaire d'effectuer quelques recherches sur le fonctionnement des différents logiciels et du matériel présents à l'INRIA.

2.1.1 Arduino, VRPN et Sofa

La carte **Arduino** était l'un des équipements indispensables de notre projet. En effet, il s'agit d'une carte électronique qui possède un micro-controleur et nous permet ainsi de contrôler les actionneurs du robot ou encore d'envoyer des données relatives au robot déformable.
Nous avons donc commencé par prendre en main ce matériel. Ayant déjà une connaissance préalable des cartes Arduino, l'utilisation du logiciel Arduino s'est avérée relativement simple, notamment grâce aux nombreuses ressources documentaires disponibles sur internet et à son utilisation intuitive.

La bibliothèque **VRPN** (Virtual-Reality Peripheral Network) est une bibliothèque informatique libre, dont la principale utilisation dans ce projet était de permettre l'envoi et la réception de donnés.
Le manque d'informations la concernant nous a ralenti dans la prise en main des différentes fonctions recherchées.

Le logiciel **SOFA**, conçu par l'INRIA, est une plate-forme de recherche et développement, permettant la réalisation de simulation physique et interactive. Ce logiciel open-source a été développé en C++ et regroupe aujourd'hui des algorithmes issus de nombreux domaines et se basant principalement sur la méthode des éléments finis (FEM). Dans notre projet, nous nous intéresserons aux différents codes permettant l'envoi des données de simulation des robots, appelés "scènes" SOFA.
La prise en main de ce logiciel fut un peu plus difficile. En effet, nous avons dû solliciter l'aide des membres de l'équipe Defrost afin de nous expliquer le fonctionnement des différentes scènes. De plus, nous ne l'avons utilisé fréquemment que vers la fin du projet.

2.1.2 Code déjà existant

L'un des premiers objectifs de ce projet a été d'étudier et de comprendre les codes déjà existants sur les robots, afin de les regrouper. Nous avions ainsi à disposition les codes d'un robot déformable à actionneurs de type "servos moteur" et ceux d'un robot dont les actionneurs étaient de type pneumatiques.

Dans la partie Arduino,

- Les codes étaient généralement créés pour un nombre d'actionneurs bien défini.

Notre premier objectif était donc de réaliser un programme viable quelque soit le nombre d'actionneurs.

- Le traitement des données était parfois commun à tous les types d'actionneurs mais certaines différences étaient notables.

Notre deuxième objectif était donc de différencier les parties du code nécessaires au bon fonctionnement des deux robots, sans que cela n'entrave le fonctionnement global.

Dans la partie liaison série (Plug-in SoftRobots),

- La liaison série n'était opérationnelle que dans le sens Ordinateur → Arduino, or le cahier des charges spécifiait l'envoi de données de l'Aruino vers l'ordinateur.

Ainsi, il était d'abord nécessaire de réaliser cette liaison, tout en respectant les bibliothèques déjà utilisées, à savoir VRPN et SOFA.

- Certains actionneurs de type servos moteurs recevaient un traitement spécial (c'est à dire que la formule utilisée était différente des autres) selon la façon dont ils étaient placés sur le robot. Ainsi, certains étaient traités de la même façon que les actionneurs pneumatiques, et d'autres non. De plus, aucun des paramètres utilisés dans les formules n'étaient commun entre les pneumatiques et les servos moteurs.

Un accès précis aux différents paramètres était donc nécessaire, ainsi qu'un code permettant de résoudre les problèmes liées aux différents traitements. A rappeler que ces codes étaient également créés pour un nombre d'actionneurs précis.

Dans les scènes SOFA,

- L'envoi des données relatives à tous les actionneurs s'effectuait simultanément, mais dans un ordre précis.

Le nouveau code devait donc respecter l'ordre d'envoi de donnés et fonctionner en parallèle avec les scènes SOFA.

2.2 Réalisation

2.1.1 Code Arduino

Nous avons séparé la réalisation du code Arduino en deux parties distinctes :

- Partie 1 : Envoi de données de la carte vers l'ordinateur
- Partie 2 : Généralisation du code recevant les données liées au logiciel SOFA.

La **première partie** consistait à concevoir un système intuitif et viable pour le transfert de données de l'Arduino vers l'ordinateur. En effet, les utilisateurs devaient pouvoir choisir facilement le type d'actionneurs, ainsi que leurs caractéristiques.
Un système de tableaux a donc été choisi, dont la taille (et donc le nombre d'actionneurs qu'il est possible d'implanter) est uniquement limitée par l'espace mémoire de la carte Arduino utilisée. La difficulté principale fut de choisir les caractéristiques à implanter (*cf Annexe1)*. Après une discussion avec l'équipe Defrost et quelques recherches dans l'ancien code, nous sommes arrivée aux conclusions suivantes :

	Servos moteurs	Pneumatiques
Caractéristiques nécessaires au fonctionnement du programme	Diamètres des poulies	-
	Angle minimum de l'actionneur	Pression minimum de l'actionneur
	Angle maximum de l'actionneur	Pression maximum de l'actionneur
	Données supplémentaires éventuelles	Données supplémentaires éventuelles

Les mêmes recherches ont été réalisées pour les actionneurs de type hydraulique et smart servos. Malheureusement, il fut difficile d'effectuer cette analyse, car l'INRIA ne dispose pas de robots avec de telles caractéristiques.

Afin de pouvoir correctement différencier les types d'actionneurs et leurs attributs lors de leur réception sur l'ordinateur, nous avons implanté un système basique de délimiteurs. Ainsi, chaque tableau contenant une caractéristique possède ce dispositif. De plus, il a également été nécessaire d'implanter ces délimiteurs à chaque fin de transmission des données d'un type, mais aussi à la fin de tous les envois. Il est à noter que seuls les types aux nombres d'actionneurs non nuls voient leur données envoyés vers la liaison série, pour des questions d'optimisation du temps de communication. De plus, afin d'obtenir une bonne coordination avec le plug-in SOFA, la transmission ne commence que si l'Arduino à reçu une lettre particulière envoyée par l'ordinateur.

La deuxième partie a demandé plus de temps à la conception et en tests. Afin de que le code fonctionne pour un nombre indéfini d'actionneurs, il était souvent nécessaire de créer des tableaux et des boucles. Malgré tout, le programme des servos moteurs utilisait la bibliothèque <servo.h>. Cette librairie permet à une carte Arduino de contrôler les servomoteurs de modélisme. Il était donc obligatoire de généraliser également le nombre de "servos" possibles dans notre code. La bibliothèque ne permettant pas de créer des tableaux de servos, nous avons eu recourt à une structure dont l'architecture est la suivante :

```
struct servom{
  Servo servo;
        };
servom servomo[NBSM];
```

Le programme déjà existant ne commençait qu'a la réception d'un "header", c'est à dire d'un chiffre choisi à l'avance et implanté dans le code Arduino. Afin de différencier les

données de type “servos moteur”, ou de type “pneumatique” par exemple, nous avons diversifié ces ”headers“ afin que le code puisse reconnaître le type de données à traiter. Un système de “passage” a été implanté afin de différencier les traitements à effectuer (*cf annexe 2*).

2.1.2 Partie communication série

La partie communication série utilise le plug-in SoftRobots de SOFA permettant le contrôle des robots déformables. Un fichier est créé par robot. Ainsi, le robot déformable possédant des actionneurs type servomoteur est contrôlé grâce à un fichier particulier, et le prototype du robot endoscopique à actionneurs pneumatiques par un autre. Le but était donc de réaliser un unique fichier, permettant le contrôle de tous les robots. Le fait que cette partie soit codée en C++ ne fut pas un obstacle, car nous disposions déjà de programmes dont nous pouvions nous inspirer.

Dans un premier temps, notre fichier générique devait pouvoir recevoir les données envoyées par l’Arduino. Il était important d’avoir une bonne coordination entre l’envoi et la réception. Nous avons donc implanté des “delays” (ou retard en français) de 1000 ms entre chaque envoi de donnée. Pour la réception, nous avons utilisé la fonction *“vrpn_read_available_characters*” dans le “setup” de notre programme , dont la structure est la suivante :

```
struct timeval timeout;
timeout.tv_sec = 1.00001;
timeout.tv_usec = 0;
vrpn_read_available_characters(portNumber,buffer,bytes,&timeout);
```

Le champ tiemout.tv_sec permettait de définir le temps de lecture de la fonction. Comme nous voulions lire une nouvelle information tout les 1000ms, nous avons défini ce timeout à 1,0001 secondes après différents tests.

Une fois les données récupérées, le but était de pouvoir correctement les classer dans des tableaux afin de pouvoir réaliser la suite du programme. Les délimiteurs envoyés par l’Arduino furent alors utilisés. Une fois les données reçues dans le setup, elles devaient être transférées dans le programme principal. Celui-ci fonctionnait exactement comme la fonction “loop” de l’Arduino, c’est à dire que les actions codées étaient répétées en continu.

Cette fonction permettait de rendre les données envoyées par SOFA exploitables par la carte Arduino. Elle comportait des formules mathématiques utilisant les caractéristiques

stockées dans le setup. Nous devions alors trouver un moyen de les transférer. Nous avons alors créé une “DATA” SOFA. Ces datas sont des variables propres à la bibliothèque SOFA, utilisables n’importe où dans le code (*cf annexe 3)*. Pour écrire dans l’une de ces variables, nous devions utiliser ce que l’on appelle un “accès en écriture”, ou “WriteAccessor”. Ainsi, tous les tableaux qui servaient à la réception des données Arduino ont été remplacés par des datas SOFA. Il suffisait ensuite de venir les lire dans la fonction principale grâce à des “ReadAccessor”.

Il était également important que notre code puisse permettre l’utilisation de différents types d’actionneurs (pneumatiques, servos moteurs,...). Nous avons donc mis en place le même système de “passage” que pour l’Arduino (*cf annexe 4)*. Le nombre de passages est égal au nombre de type d’actionneurs présents sur le robot sélectionné (dans notre cas nous avons commencé par 2). Chaque passage traite un type d’actionneur en particulier. Ainsi le premier passage est réservé au type servo moteur. Si le nombre de servos moteurs est nul, alors la donnée n’est pas envoyée vers l’Arduino. Il en est de même pour le type pneumatique.

Dans un souci d’optimisation, certaines parties du programme sont communes à tous les types d’actionneurs. Les parties du code dédiées à un type d’actionneur sont effectuées uniquement à l'apparition d’un numéro de passage précis.

Les variables ont été remplacées par des tableaux dont l’espace dépend du nombre d’actionneurs du type sélectionné. Afin d’éviter que le code ne soit trop long, nous avons créé des variables “génériques”, dont les valeurs sont modifiées à chaque nouveau passage dans la boucle. La même variable est donc utilisée que ce soit des actionneurs pneumatiques ou servos moteurs.

Le code permet ainsi d’envoyer un “paquet” de données, dont la taille correspond au nombre d’actionneurs. Le header dont nous avons expliquer l’usage dans la partie Arduino est également ajouté au paquet.. C’est ici la fonction *“ vrpn_write_available_characters”* qui permet l’envoi.

3 Tests et résultats

3.1 Tests effectués

Nous avons consacré beaucoup de temps à réaliser les tests et le déboggage de notre code . En effet, celui -ci comporte de nombreuses parties, qui devaient fonctionner parfaitement entre elles.

Nous avons effectué nos tests de la façon suivante :

- Tout d'abord, afin de pas détériorer les robots déformables, de nombreux tests ont été effectués "en dur". Des valeurs prédéfinies qui n'avaient pas forcement de lien avec la réalité, étaient utilisées. Nous avons testé séparément les parties envoi/réception de l'Arduino, ainsi que la partie concernant le plug-in SoftRobots.

- Des tests ont ensuite été réalisés après avoir lié le plug-in avec le logiciel SOFA. Nous avons pu avoir accès aux scènes des deux robots conçus par l'INRIA. Il suffisait donc de comparer les valeurs entre les anciens et le nouveau code.

- L'une des parties les plus importante était le bon fonctionnement entre le nouveau code du plug-in et le nouveau code de l'Arduino. Nous avons donc réalisé des tests liant l'ensemble avant de les implanter sur les robots réels.

- Enfin les tests ont été réalisés sur les vrai robots.

3.2 Résultats obtenus

3.2.1 Des résultats concluants

Après avoir effectué les étapes décrites dans ce rapport, nous avons pu constater que les tests étaient concluants sur les deux robots. Il suffisait d'entrer les données souhaitées dans l'Arduino pour permettre le contrôle d'un robot pneumatique, ou avec des servos moteurs.

La prise en main du code nécessite tout de même quelques informations au préalable. En effet, il est important que les données de la scène SOFA soient positionnées dans le même ordre que sur le programme Arduino. Par exemple, les informations du "*servo nord*" doivent se situer à la même position que ce soit dans la scène ou dans le code Arduino. Des commentaires ont été ajoutés dans le code Arduino afin de faciliter son utilisation.

3.2.2 Pour aller plus loin

Nous avons manqué de temps pour effectuer la réalisation des Feedbacks. Des problèmes (qui seront exposé dans la suite de ce rapport) nous ont ralenties dans la conception de ce code.
De plus, le programme fonctionne uniquement pour 2 types d'actionneurs, les tests étant impossibles pour ceux de type hydraulique ou smart servos.

4 Problèmes rencontrés et solutions envisagées

4.1 Problèmes rencontrés

L'un des principaux problèmes que nous avons rencontré durant la réalisation de ce projet était les mises à jour assez récurrentes du logiciel Sofa. En effet, ceci nous beaucoup ralenti puisque à chaque fois la réinstallation d'une nouvelle version de Sofa prenait énormément de temps pendant lequel nous étions incapables de réaliser nos tests.

Un autre inconvénient était la répartition horaire du projet qui changeait assez souvent et donc qui impliquait une réorganisation de notre planning prévisionnel en plus des interruptions qu'il a fallu gérées.

Un dernier petit bémol était que l'équipe Defrost ne disposait encore pas de robots déformables fonctionnant avec des actionneurs hydrauliques ou des smart servos ou des robots mixtes avec deux types d'actionneurs. Nous n'avions donc pas pu tester notre code sur un prototype réel.

4.2 Solutions envisagées

Afin de palier au problème de la lenteur des mises à jour Sofa, nous avons consacré ce temps pour travailler sur notre programme Arduino en lui apportant les modifications nécessaires afin d'améliorer son fonctionnement.

Nous avons également fait preuve de beaucoup de flexibilité et nous avons réussi à nous adapter aux différents changements des créneaux horaires alloués au projet. Une organisation solide et un travail rigoureux étaient essentiels.

N’ayant pas eu l’occasion de tester l’ensemble de notre code sur de vrais prototypes, Nous avons effectués de nombreux tests en dur afin de nous assurer de son bon fonctionnement et donc de permettre à l’équipe Defrost de l’utiliser à l’avenir sur des robots déformables avec différents types d’actionneurs.

Conclusion

Malgré les quelques problèmes que nous avons rencontrés, nous avons pu répondre à la problématique principale de notre projet qu'est le développement d'un code permettant de commander des robots déformables avec un nombre variable et des types différents d'actionneurs. Actuellement, nous sommes capable de commander n'importe quel robot déformable en rentrant simplement les caractéristiques relatives aux différents actionneurs qui le composent sur sa carte Arduino et l'aide d'une simulation informatique sur Sofa. Ceci permet de réaliser un gain de temps considérable et une grande facilité d'utilisation.

Nous n'avons néanmoins pas pu réaliser les "feedback" qui nous ont été demandé au début du projet par manque de temps. Ceci n'a toutefois aucune incidence sur la commande des robots déformables puisque l'utilité des feedback se limitaient juste à avoir un retour sur les données envoyées par Sofa afin de commander le robot.

Ce projet nous a été très bénéfique puisqu'il nous a permis non seulement de mettre en pratique nos connaissances de l'Arduino et de la gestion d'une liaison série mais aussi d'acquérir de nouvelles connaissances en programmation objet notamment grâce à l'utilisation de C++.

Annexes

Annexe 1

```
// Actionneurs type servos moteurs
float diametre_SM[7] = {35.5,35.5,35.5,35.5,998, 999}; // ATTENTION : la plage réservée aux diamétres est de 7. 999 et 998 sont les délimiteurs de fin d'envoi, toujours le garder en fin de tableau.
float angle_min_SM[7] = {165,165,15,15, 997, 999} ; // ATTENTION : la plage réservée aux angles minimum est de 7. 999 et 998 sont les délimiteurs de fin d'envoi, toujours le garder en fin de tableau.
float angle_max_SM[7] = {15,15,165,165, 996, 999} ; // ATTENTION : la plage réservée aux angles maximum est de 7. 998 et 999 sont les délimiteurs de fin d'envoi, toujours le garder en fin de tableau.
float donnee_sup_potentielle[3]={995, 999}; // Espace reservé pour UNE information suplémentaire potentielle

float angle_min_SM_arduino[5] = {180,180,15,15} ; // ATTENTION : la plage réservée aux angles minimum est de 5. 999 et 998 sont les délimiteurs de fin d'envoi, toujours le garder en fin de tableau.
float angle_max_SM_arduino[5] = {15,15,180,180} ; // ATTENTION : la plage réservée aux angles maximum est de 5. 998 et 999 sont les délimiteurs de fin d'envoi, toujours le garder en fin de tableau.
```

Annexe 2

```
void loop()
{

    int i,j,test;
    int generique_max[5];
    int generique_min[5];
    int nb_generique=0;

    if ((passage ==1) && (nbdeSM == 0))
    passage++;

    if ((passage ==2) && (nbdepneumatique ==0))
```

```
passage++;

if (passage ==3)
{ if (nbdeSM == 0)
  passage=2;
  else
    passage=1;
}


 Serial.print(passage);

while(((test!=98) && (passage==1)) || ((test != 99) && (passage==2)))
{

      test=Serial.read();
  Serial.print("t ");
  Serial.print(test);

}


      if (test == 98)
  {

      nb_generique=nbdeSM;
  for (i=0; i<nb_generique; i++)
  {
      passage=1;
      generique_max[i]=angle_max_SM_arduino[i];
      generique_min[i]=angle_min_SM_arduino[i];
  }
      }
      if (test == 99)
      {
      nb_generique=nbdepneumatique;
   for (i=0; i<nb_generique; i++)
   {
```

```
        passage=2;
        generique_max[i]=pression_max_arduino[i];
    generique_min[i]=pression_min_arduino[i];
     }
        }



 int pos[nb_generique];
 int posm[4]={30,40,50,60};
int posp[nb_generique];


for (j=0; j<nb_generique; j++)
{
delay(1);

while(!Serial.available());

if (passage==1 && nbdeSM != 0 )
{
pos[j]=Serial.read()+15;

}

if (passage==2 && nbdepneumatique !=0)
pos[j]=0.33*Serial.read();

}



for (j=0; j<nb_generique; j++)
{
if (generique_min[j]<generique_max[j])
pos[j]=constrain(pos[j], generique_min[j],generique_max[j]);
else
```

```
  pos[j]=constrain(pos[j], generique_max[j],generique_min[j]);
  Serial.print("positions ");
  Serial.print(pos[j]);
  }

  for (j=0; j<nb_generique; j++)
  {
  if (passage==1 && nbdeSM != 0 )
  (servomo[j].servo).write(pos[j]);
  if (passage==2 && nbdepneumatique !=0 )
  {
  analogWrite(PINpressure[j],pos[j]);
  }
  }


  passage++;
  test=0;

}
```

Annexe 3

```
unsigned char packet[nombre_actionneurs+1];
        WriteAccessor<Data<vector<double> > > infos_a_p= Infos_a_p;
        infos_a_p.resize(nombre_actionneurs);

        WriteAccessor<Data<vector<double> > > angle= Angle;
        angle.resize(nombre_actionneurs);
```

Annexe 4

```
if (passage==1)
        {
        tabinfomin[i]=angle_min[i];
        tabinfomax[i]=angle_max[i];
        infos_a_p[i]=(sentData.getValue()[i]+15)*360.0/(diametres[i]*3.14);
        }

        if (infosservosm[2] != 0 && passage==2)
        {
        tabinfomin[i]=pression_min[i];
        tabinfomax[i]=pression_max[i];
        infos_a_p[i]=sentData.getValue()[i];
        }
```

Bibliographie

“Christian Duriez ≫ Research.” *Shacra*. N.p., n.d. Web. 2 May 2015.
<https://team.inria.fr/shacra/christian-duriez-research/>

“Compiling And Running Standard Apps.” *VRPN*. N.p., n.d. Web. 2 May 2015.
<http://www.cs.unc.edu/research/vrpn/vrpn_standard_stuff.html>

“Deformable Robots Simulation Team.” *Présentation*. N.p., n.d. Web. 2 May 2015.
<http://www.inria.fr/equipes/defrost>

“Qt - Download Open Source Step 3.” *Qt*. N.p., n.d. Web. 2 May 2015.
<https://www.qt.io/download-open-source/>

“SOFA.” *SOFA*. N.p., n.d. Web. 2 May 2015. <http://www.sofa-framework.org/>

“Software / SOFA.” *Shacra*. N.p., n.d. Web. 2 May 2015.
<https://team.inria.fr/shacra/software/>

“WHAT IS ARDUINO?” *Arduino*. N.p., n.d. Web. 2 May 2015. <http://www.arduino.cc/>