heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Elisa-3 à la Maison d'Ailleurs

Projet réalisé dans le cadre du cours Science-fiction et Technologie 2015

A. Descoulds, A. Hug, S. Righitto, F. Saam

Remerciements

Nous tenons à remercier tout particulièrement Hector Satizabal Mejia pour nous avoir aidés dans ce projet. Le soutien moral, le matériel et le code qu'il nous a fournis ont été un élément clé dans la réussite de ce projet.

Un grand merci aussi à l'équipe qui a travaillé avant nous sur le projet. Ils nous ont fourni une documentation très complète qui nous a beaucoup aidés dans le démarrage de ce travail et que nous avons largement reprise ici.

Nous remercions également le professeur M. Andres Perez-Uribe qui était toujours à notre écoute et nous prodiguait de bons conseils, tout en nous laissant une grande liberté dans ce projet. Merci pour votre confiance !

A tous, un grand merci!

Table des matières

1 Introduction

Elisa-3 est un robot conçu par GCtronic[1] et équipé notamment de capteurs, d'émetteurs infrarouges et de leds RGB qui permettent de colorer le couvercle du robot.

Le but général de ce projet est de pouvoir mettre en place un scénario intéressant utilisant des robots. En effet, la Maison d'Ailleurs, musée de la science-fiction situé à Yverdon-les-Bains, mettra à notre disposition un espace afin de montrer notre travail, ceci dans le cadre de sa future exposition sur la thématique des robots « Portrait-Robot ».

Notre concept – détaillé par la suite – est le suivant : un des robots est défini comme étant un danseur fou. Son but est de transmettre sa passion pour la danse à tous les autres robots. Ceux-ci sont immobiles et ont une couleur les distinguant du robot danseur. Dès qu'un robot immobilisé est approché par le robot danseur, ce dernier commencera lui aussi à danser, et devient par conséquent à son tour un danseur fou. Après deux minutes, le spectacle est terminé et le visiteur du musée devra lui-même relancer le programme pour recommencer le spectacle.

Le projet décrit par la suite est en réalité une reprise d'un premier projet (réalisé par Marcel Sinniger, Dominique Jollien, Stéphane Maillard et Auriana Hug). Nous avons donc pu nous baser sur une première expérience. Toutefois, comme nous le verrons par la suite, nous avons fait face à de nouveaux challenges.

En outres, ce document explique les différents outils de programmation nécessaires pour travailler avec le robot Elisa-3. Les fonctionnalités imaginées dans notre scénario seront expliquées au niveau de leur mise en œuvre et de leur implémentation.

---

[1] GCtronic. 2014. « Elisa-3 - GCtronic Wiki ». 4 décembre 2014. http://www.gctronic.com/doc/index.php/Elisa-3.

2 Objectif de ce projet

Pour notre projet, nous avons imaginé un concept basé sur les caractéristiques du robot Elisa-3, que nous décrirons dans le chapitre suivant. De plus, nous nous sommes inspirés de l'idée de colonie. En effet, nous aurons à disposition une dizaine de robots, et leur petite taille permet de mettre en place aisément un environnement pour plusieurs robots de son type. Dès lors, il est intéressant de les utiliser ensemble dans cette optique.

Ainsi, nous pourrons réutiliser une partie du code déjà conçu lors du projet précédent, et l'objectif est d'y ajouter d'autres fonctionnalités, afin de rendre un projet ludique et fonctionnel pour la Maison d'Ailleurs.

3 Elisa-3

Ce petit robot est rond, mesure moins de 10cm de diamètre et possède de multiples senseurs et fonctionnalités. Nous allons les découvrir dans ce chapitre.

3.1 Déplacement

Pour se déplacer et tourner, le robot Elisa-3 possède deux roues indépendantes, chacune étant couplée à son propre moteur DC. De plus, un accéléromètre sur trois axes permet au robot de connaître son orientation dans l'espace. Le déplacement du robot et la charge sont indépendants de la gravité. Le robot travaille également à la verticale et à l'envers, grâce à ses aimants. Sa position peut être exploitée par le programme du robot afin de changer le comportement du robot, par exemple.

3.2 Détection d'objets

Elisa-3 est équipé de 8 émetteurs infrarouges couplés avec des senseurs répartis sur le côté de l'appareil. Ils permettent notamment la détection d'objets à proximité (*object avoidance*), ainsi que l'évitement de chutes lors de l'approche d'un bord de table (*cliff avoidance*).

3.3 Communication avec un ordinateur

Le robot possède aussi une antenne RF afin de communiquer avec un ordinateur. Il envoie les informations de ses capteurs et peut recevoir des instructions. En outres, le robot est équipé d'un processeur ATMEL ATmega2560 compatible Arduino. Un port micro-USB permet la connexion à un ordinateur à l'aide d'un câble micro USB, afin de changer le code du robot. Un sélecteur à 8 positions est présent sur le PCB.

3.4 Communication avec un autre robot

Ils peuvent également communiquer des informations avec un robot semblable.

3.5 Couleurs

Une led RGB centrale permet au robot d'afficher un large choix de couleurs sur le dessus du l'appareil. Elisa-3 dispose aussi de huit leds vertes sur le côté.

3.6 Batterie

Une batterie équipe le robot. Elle peut être chargée soit en le branchant avec un câble micro USB à l'ordinateur, soit en positionnant les deux contacts dorés sur une station de rechargement adaptée. Lors du rechargement, une petite led rouge frontale s'allume.

4 Documentation officielle

Le site officiel du fournisseur décrit les différentes manières de programmer et de gérer le robot Elisa-3. Voici le lien : http://www.gctronic.com/doc/index.php/Elisa-3.

5 Matériel utilisé

Pour ce projet, nous avons utilisé :

- 8 robots Elisa-3
- 1 antenne RF (avec son câble)
- 3 bases de recharges (avec leur câble)
- 1 environnement (l'arène)
- Phidget Touchrotation 1016
- Rapsberry Pi

6 Environnement de développement

GCtronic propose plusieurs IDE pour programmer les robots. Le seul que nous avons pu utiliser est Arduino. Malheureusement, la majorité des exemples de code sur le site du fournisseur ont été conçus pour AVR Studio. En outres, la traduction du code est pénible, car la librairie n'est pas complétement la même pour les deux IDE. Ainsi, la librairie "Avancée" doit être utilisée avec AVR Studio. Il s'est avéré impossible à la faire fonctionner en utilisant Arduino.

6.1 Installation du logiciel Arduino

Cette page web du fournisseur explique comment installer Arduino :

http://www.gctronic.com/doc/index.php/Elisa-3#Arduino_IDE_project

Après l'installation et la configuration de ce logiciel, il est nécessaire de faire un premier test. Pour ce faire, il est possible de suivre les instructions situées juste après la section qui décrit les modifications du fichier « boards.txt ».

L'utilisation des programmes développés pendant le cours nécessite l'utilisation de la librairie côté robot basique que nous avons modifié. Celle-ci a notamment dans le fichier constants.h des valeurs différentes pour LINE_IN_THR LINE_OUT_THR. Comme indique « Tools=>Serial Port », il faut choisir le port COM pour établir la connexion avec le robot. Nous avons rencontré des problèmes avec les ports USB 3.0. En effet, la connexion au robot ne fonctionne pas avec ces derniers. Heureusement, nous avons constaté que ceci fonctionne avec les ports USB 2.0. Sur les ordinateurs portables de LeNovo, on peut utiliser les ports USB, identifiés en jaune (ports USB 2.0).

6.2 Développement avec l'antenne

Pour qu'on puisse communiquer depuis un ordinateur avec les robots à l'aide de l'antenne, il faut tout d'abord installer quelques outils. Les procédures d'installation et de configuration sont décrites sur la page du fournisseur d'Elisa-3 :

http://www.gctronic.com/doc/index.php/Elisa-3#PC_side

La mise en œuvre d'un projet « PC-Side » est pénible. C'est pourquoi nous avons préparé une version de CodeBlocks portable qui est déjà préconfigurée (voir le chapitre 9 « Contamination »). Il faut installer MinGW1 dans le dossier C:\MinGW. Lors de l'installation, il est nécessaire de sélectionner les compilateurs C et C++ de MinGW, afin de les installer. Lorsque la procédure est terminée, il faut brancher l'antenne sur un port USB 2.0. En effet, le problème mentionné dans le chapitre précédent peut survenir.

6.3 Débugging avec l'antenne

La communication à l'aide de l'antenne permet aussi de récupérer des valeurs des capteurs des robots. En effet, les robots peuvent envoyer des valeurs au PC. Ceci permet de mieux comprendre ce qui se passe sur le robot. Le projet précompilé du fournisseur dans le chapitre « PC side » montre comment des valeurs peuvent être échangées. Une version exécutable de ce projet se trouve dans le dossier « Debugging\Debugging_avec_l_antenne » De plus, le sous-chapitre « Code du côté PC » du chapitre « Contamination » exploite cette fonctionnalité pour la récupération des états des batteries des robots.

7 Concept

Les robots sont tous dans un état d'attente, prêts à être activés. Un robot choisi au hasard parmi les robots disponibles et possédant un niveau suffisant de batterie devient le « danseur fou ». Lorsqu'un visiteur du musée touche l'interface TouchRotation, il commence à danser.

Quand le robot danseur s'approche des autres robots, il active ces derniers et ils deviendront à leur tour des danseurs. En quoi consiste la danse des robots ? Un robot danseur se déplace principalement, de manière aléatoire et en changeant de couleur toutes les 5 secondes. De plus, toutes les 40 secondes, il effectue des pas de danse prédéfinis.

Pendant ce temps, les robots contrôlent constamment leur niveau de batterie. Si celui-ci est inférieur au minimum vital, ils passeront en mode « rechargement ». Ceci implique le suivi d'une ligne blanche qui est capable de les amener directement à la station de chargement. Une fois sur la station de rechargement, le robot doit continuellement contrôler s'il est en contact avec le chargeur. En cas de déconnexion, il contrôle son niveau de batterie. S'il est suffisamment haut, il commence la procédure de déconnexion définitive. Si le niveau est encore insuffisant, le robot essaie de se reconnecter. Un timer général sera chargé de contrôler la durée du spectacle. En effet, nous avons décidé de fixer une limite de 2 minutes pour le spectacle. Apres cela, les robots se remettent à l'état initial, donc en attente de la commande envoyée par le visiteur du musée.

8 Implémentation

Bien que nous ayons reçu le code fonctionnel de la première partie du projet, nous avons décidé de remplacer la librairie utilisée par la nouvelle, fournie par GCtronic. En effet, la première librairie ne permettait malheureusement pas d'utiliser toutes les fonctionnalités énoncée du robot (notamment le *cliff avoidance* et la communication entre les robots).

Toutefois, nous nous sommes rapidement rendu compte que le code existant ne fonctionnait pas avec la nouvelle librairie. Par conséquent, nous avons passé un temps considérable à la comprendre et à corriger le code pour le faire fonctionner.

Dans ce chapitre, nous présentons le concept final et son implémentation, ainsi que les difficultés rencontrées.

8.1 Écoute du signal de démarrage

Nous avons tenté de communiquer avec plus de 4 robots à l'aide de l'antenne. Cependant, nous avons rencontré un certain nombre de difficultés. Une variable nommée NUMBER_ROBOTS ne changeait apparemment pas le nombre de robots auxquels nous pouvions transmettre de l'information. En effet, bien que nous ayons modifié la valeur à 5 ou 6, nous ne pouvions communiquer qu'avec 4 d'entre eux. Après de nombreux tests effectués à d'autres endroits dans le code, le problème persistait, même en essayant de modifier la bibliothèque fournie. Nous avons fini par déduire que le nombre de robots ne peut être qu'une puissance de 2. En effet, en initialisant la variable NUMBER_ROBOTS à 8, nous avons réussi à faire fonctionner plus de 4 robots.

Dans son état initial, le robot attend que l'antenne lui envoie une commande. Ensuite, tous les robots passent au bleu puis un seul devient rouge pour démarrer la « contamination ». A ce stade, les robots bleus attendent de détecter qu'un autre robot passe à côté avant de démarrer. Voici le code vérifiant la proximité :

```
boolean checkNearbyObjects() {
    unsigned int proximityResultFront = proximityResultLinear[0] +
    proximityResultLinear[1] + proximityResultLinear[7];
    unsigned int proximityResultBack = proximityResultLinear[3] + p
    roximityResultLinear[4] + proximityResultLinear[5];

    if (proximityResultFront >= FRONT_PROX || proximityResultBack >= BACK_PROX) {
    return true;
    } else if (proximityResultLinear[2] > LATERAL_PROX || proximityResultLinear
[6] > LATERAL_PROX) {
        return true;
```

```
    } else {
        return false;
    }
}
```

Dans tous les états du robot, ce dernier vérifie son niveau de batterie afin de savoir quand il doit aller se recharger. Si le niveau minimal est atteint, il part à la recherche d'une ligne.

```
const unsigned int BATTERY_LEVEL_STOP_CHARGING = 950;
const unsigned int BATTERY_LEVEL_START_CHARGING = 650;
void updateBatteryLevel() {
  if (getTime100MicroSec() > (batteryStart + PAUSE_5_SEC)) {
    // update the battery level every 5 seconds
    readBatteryLevel();
    batteryStart = getTime100MicroSec();

    if (batteryLevel <= BATTERY_LEVEL_START_CHARGING) {
      //the robot must now go to charging state
      robotState = LOW_BATTERY;
    }
  }
}
```

8.2 Suivi des lignes et rechargement

Pour le suivi des lignes, nous avons d'abord repris le code conçu par le groupe précédent, puis nous avons remarqué quelques problèmes liés au fait que nous utilisions leur code avec une nouvelle version de la librairie prévue pour les robots. Notamment, ces derniers ne trouvaient plus la ligne à chaque fois et ne pouvaient donc pas aller se recharger. Nous avons donc décidé de changer les valeurs fixes du noir et du blanc par des valeurs calculées au démarrage du robot.

```
unsigned int line_in = 0, line_out = 0;
line_out = proximityResult[8];
line_in = line_out-50;
```

Avec ce code, les robots arrivaient à trouver la ligne plus facilement et ne se trompaient plus. Ensuite, comme la surface du nouvel environnement est désormais aimantée, nous avons remarqué qu'il était très difficile de les faire se connecter correctement au chargeur. En effet, il y avait toujours une des deux « pins » de rechargement qui ne touchait pas le chargeur. Nous avons donc ajouté dans le code une partie permettant au robot de faire des mouvements de repositionnement.

```
if (proximityResult[1] >= 700 || proximityResult[7] >= 700) {
   if ((getTime100MicroSec()-counterPerso) >= PAUSE_1_SEC) {
     counterPerso = getTime100MicroSec();
     if (testBool == 1) { testBool = 0; }
     else { testBool = 1; }
   }

   if (testBool == 1) {
     setRightSpeed(15);
     setLeftSpeed(-5);
   } else {
     setRightSpeed(-5);
     setLeftSpeed(15);
   }
}
```

Voici le premier code terminé permettant de suivre la ligne:

```
if (proximityResult[1] >= 700 || proximityResult[7] >= 700) {
   if ((getTime100MicroSec()-counterPerso) >= PAUSE_1_SEC) {
      counterPerso = getTime100MicroSec();
      if (testBool == 1) {testBool = 0;}
      else {testBool = 1;}
   }

   if (testBool == 1) {
      setRightSpeed(15);
      setLeftSpeed(-5);
   } else {
      setRightSpeed(-5);
      setLeftSpeed(15);
   }
}
else if (proximityResult[9]>line_out) {  // center left is leaving the line =>
turn right
   setLeftSpeed(15);
   setRightSpeed(-5);
```

```
} else if (proximityResult[10]>line_out) {   // center right is leaving the lnie
=> turn left
   setLeftSpeed(-5);
   setRightSpeed(15);
} else if (proximityResult[8]<line_out && proximityResult[9]>line_out &&
proximityResult[10]>line_out && proximityResult[11]>line_out) { // left ground is
the only within the black line => turn left
   setLeftSpeed(-10);
   setRightSpeed(15);
} else if (proximityResult[11]<line_out && proximityResult[8]>line_out &&
proximityResult[9]>line_out && proximityResult[10]>line_out) { // right ground is
the only within the black line => turn right
   setLeftSpeed(15);
   setRightSpeed(-10);
} else {
   setRightSpeed(15);
   setLeftSpeed(15);
}
```

Puis le concept ayant évolué, nous ne travaillons désormais plus avec des lignes bien tracées, mais avec des lignes aux contours flous. Par conséquent, nous avons dû refaire un nouveau code pour lequel nous avons reçu l'aide de la part d'Hector Satizabal Mejia l'assistant du cours, qui utilise un algorithme nommé "Braitenberg" permettant de suivre facilement des lignes floues en injectant des valeurs modifiées de certains capteurs directement dans les vitesses des roues.

```
void braitenbergLineFollower() {

 front_diff = ((int)proximityResult[9] - (int)proximityResult[10]) >> 5;

 setGreenLed(4, 0);
 //-------------------------------------------------------------------------
 if (inLine) {
   if ( (proximityResult[9] < LINE_OUT_THR_BK) && (proximityResult[10] <
LINE_OUT_THR_BK) ) {
     inLine = false;
     setGreenLed(0, 1);
     enableObstacleAvoidance();
   }
 }
```

```
 else {
   if ( (proximityResult[9] > LINE_IN_THR_BK) || (proximityResult[10] >
LINE_IN_THR_BK) ) {
     inLine = true;
     setGreenLed(0, 0);
     disableObstacleAvoidance();
   }
 }
 //-------------------------------------------------------------------------
 if ( !inLine && (proximityResult[8] > LINE_IN_THR_BK) && (proximityResult[11] >
LINE_IN_THR_BK) ) {  //special case... lateral sensors on the line: turn 90
degrees
   if (accY > 0) {      // check the slope and turn the robot to go down
     setLeftSpeed(15);
     setRightSpeed(-15);
     setGreenLed(1, 1);
     setGreenLed(7, 0);
   }
   else {
     setLeftSpeed(-15);
     setRightSpeed(15);
     setGreenLed(1, 0);
     setGreenLed(7, 1);
   }
   startTempAction(PAUSE_500_MSEC);
 }
 else {
   setGreenLed(1, 0);
   setGreenLed(7, 0);
   setLeftSpeed(CONSTANT_SPEED_FOLLOW - front_diff);
   setRightSpeed(CONSTANT_SPEED_FOLLOW + front_diff);
 }
}
```

Pour la partie rechargement, le comportement du robot est le suivant. Une fois connecté à la station de rechargement, le robot contrôle en permanence s'il reste en contact avec le chargeur. Si une déconnexion survient, le robot contrôle si le niveau de sa batterie est suffisamment haut pour retourner dans le spectacle ou non. Si oui, il recule et il se remet en jeu. Si non, il essaiera de se reconnecter à la station.

Voici le code qui contrôle la connexion avec la station de rechargement:

```
unsigned int chargeContactDetected = 0;
boolean charging() {
  if (batteryLevel > BATTERY_LEVEL_STOP_CHARGING) {
    return false;
  }
  else if (CHARGE_ON) {
    chargeContactDetected++;
    if (chargeContactDetected > 20) {
      return true;
    }
    return false;
  }
  else {
    chargeContactDetected = 0;
    return false;
  }
}
```

Voici le code de la routine effectuée par le robot dans le cas où il est connecté avec le chargeur:

```
case IN_CHARGER:
setLEDcolor(255, 0, 255);
setLeftSpeed(0);
setRightSpeed(0);

if (!charging()) {

    if (batteryLevel < BATTERY_LEVEL_MORE_CHARGING) { //we need more charging
          setLEDcolor(255, 255, 0);
          previouslyCharging = true;
          robotState = LOW_BATTERY;
     } else {
          previouslyCharging = false;
          robotState = GOING_OFF_CHARGER;
     }
}
break;
```

8.3 Communication entre les robots

A plusieurs reprises, nous avons testé la communication locale entre robots. Après avoir résolu différents problèmes initiaux liés à la librairie, nous avons réussi à échanger des messages entre les robots. Voici les deux fonctions développées afin de pouvoir envoyer ou recevoir des messages.

```
void sendToRobots(unsigned char toSend) {
  if (irCommDataSent() == 1) {
    irCommSendData(toSend);
  }
  irCommTasks();
}
unsigned char received = NO_DATA;
unsigned char senseCommunication() {
   irCommTasks();
   if (irCommDataAvailable() == 1) {
       received = irCommReadData();
   }
   return received;
}
```

Finalement, nous avons dû enlever la communication locale du projet final, car c'était une source de problèmes pour le suivi des lignes. En effet, la charge de travail générée par les opérations d'envoi et de réception des donnés diminue la réactivité des capteurs du robot. Ceux-ci se trouvent alors en difficulté quand ils doivent suivre les lignes.

8.4 Base mode

Le « base mode » est le comportement de base du robot. Celui-ci est activé quand un robot en attente de démarrer son spectacle entre en contact avec un robot qui danse. Son comportement est assez simple: il doit avancer de manière constante en évitant les obstacles qu'il trouve sur son pasage. De plus, il change sa couleur de manière aléatoire toutes les 5 secondes. Enfin, chaque 40 secondes il effectue une des danses prédéfinies. Voici le code source qui permet d'implémenter ce « base mode »:

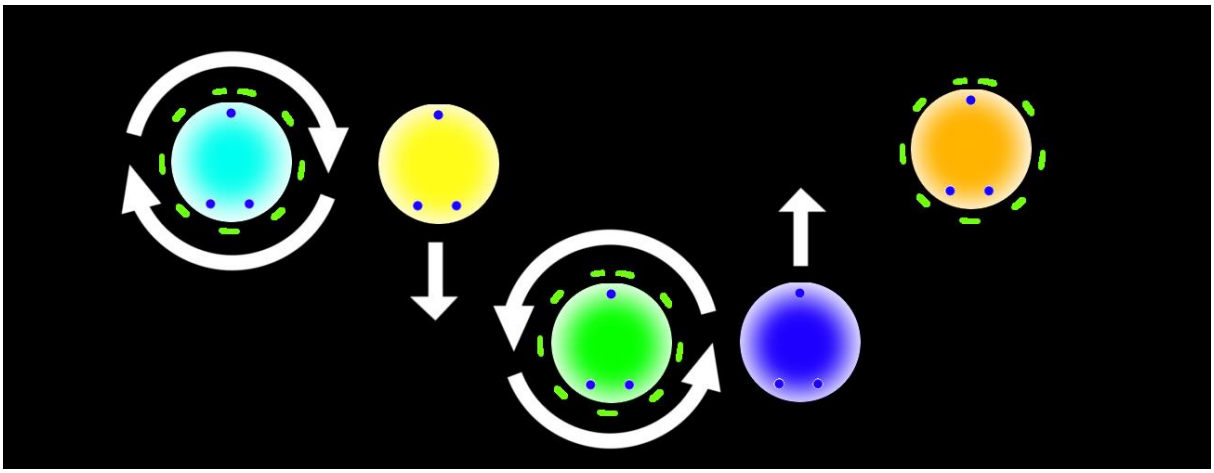
```
unsigned long int startChangeColor = 0;
unsigned int randDance = 0;
void baseMode() {
  switch (base_state) {
    case 0:
      setRandomColor();
      enableObstacleAvoidance();
```

```
      setLeftSpeed(NORMAL_SPEED);
      setRightSpeed(NORMAL_SPEED);
      base_state = 1;
      break;
    case 1:
      currentTime = getTime100MicroSec();

      if ((currentTime - startChangeColor) >= (PAUSE_5_SEC)) {
        startChangeColor = currentTime;
        setRandomColor();
      }

      if ((currentTime - startDance) >= (PAUSE_40_SEC)) {
         //startDance = currentTime; this will be updated at the end of the dance
         in order to let the base mode continue PAUSE_40_SEC
        randDance = rand() % 2;
        switch (randDance) {
          case 0:
            robotState = DANCE_1;
            break;
          case 1:
            robotState = DANCE_2;
            break;
          default:
            robotState = DANCE_1;
        }
      }
  }
}
```

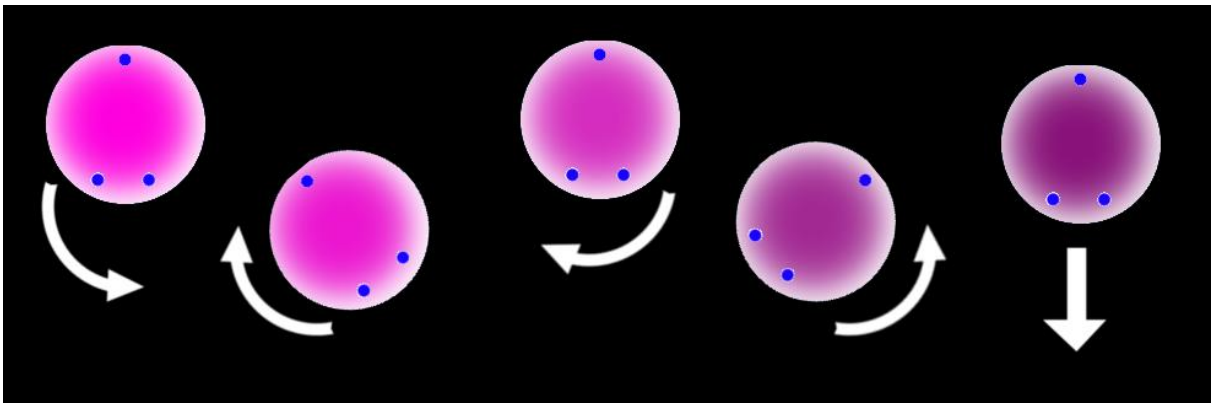
8.5 Danses

Nous avons choisi d'implémenter deux danses pour ce projet. L'objectif des danses est d'utiliser au maximum les différents mouvements possibles d'Elisa-3, tout en étant visuellement artistique. Voici deux schémas afin de mieux comprendre les pas de danse effectués par les robots.

Danse 1



Danse 2



Pour la danse 2, le schéma illustre seulement la première moitié de la danse. La deuxième partie est symétrique à la première: le robot effectue les mêmes mouvements mais dans le sens inverse.

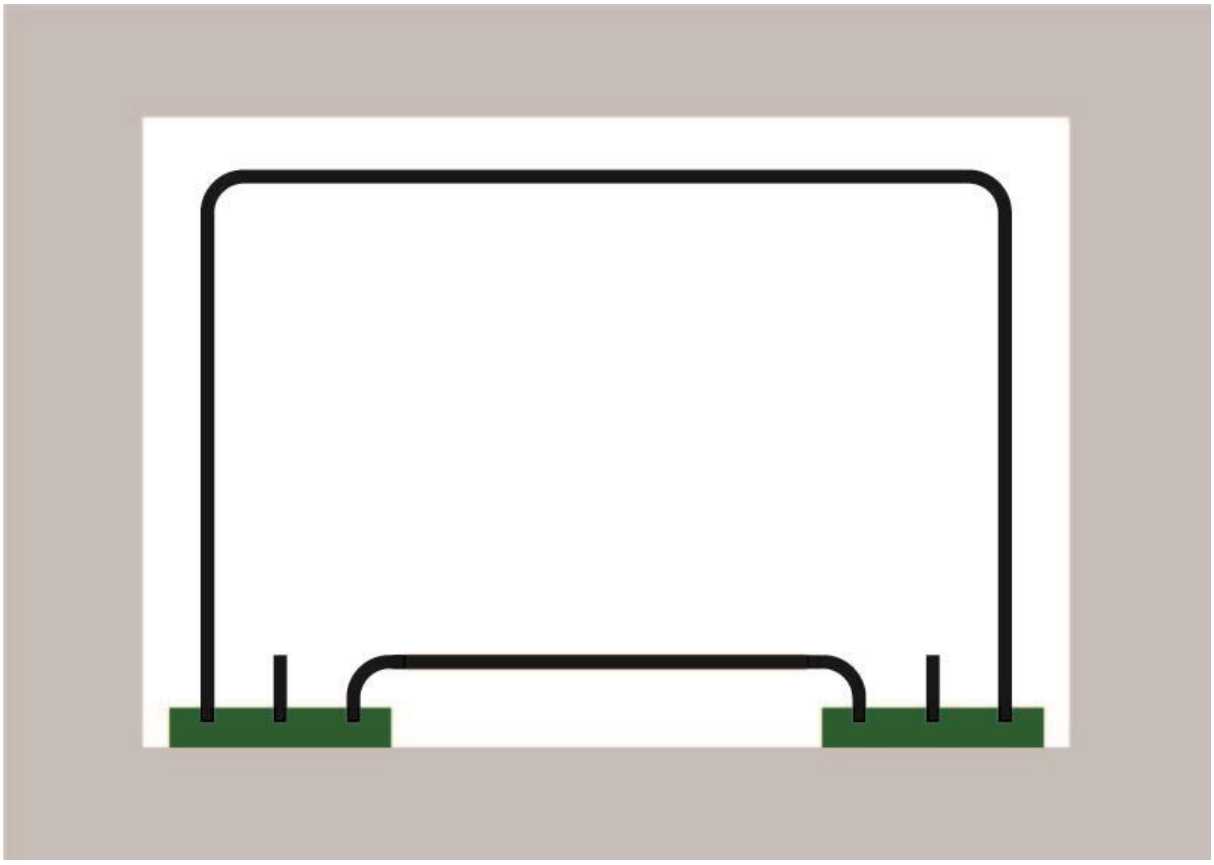
Nous ne montrons pas le code entier des deux danses dans ce rapport. Par contre, il est possible de le trouver sur notre repo Github. Toutefois, voici le code des deux fonctions utiles – rotate() et turn() – pour les danses :

```
//rotationDirection : 0 --> clockwise ; 1 --> cunterclokwise
void rotate(int rotationDirection, unsigned int rotationSpeed) {
  switch (rotationDirection) {

    case 0:
      setLeftSpeed(rotationSpeed);
      setRightSpeed(- rotationSpeed);
      break;
    case 1:
      setLeftSpeed(- rotationSpeed);
      setRightSpeed(rotationSpeed);
      break;
  }
}

//turnDirection : 0 --> clockwise ; 1 --> cunterclokwise
void turn(int turnDirection, unsigned int turnSpeed) {
  switch (turnDirection) {

    case 0:
      setLeftSpeed(turnSpeed);
      setRightSpeed(0);
      break;
    case 1:
      setLeftSpeed(0);
      setRightSpeed(turnSpeed);
      break;
  }
}
```

9 Mise en place finale

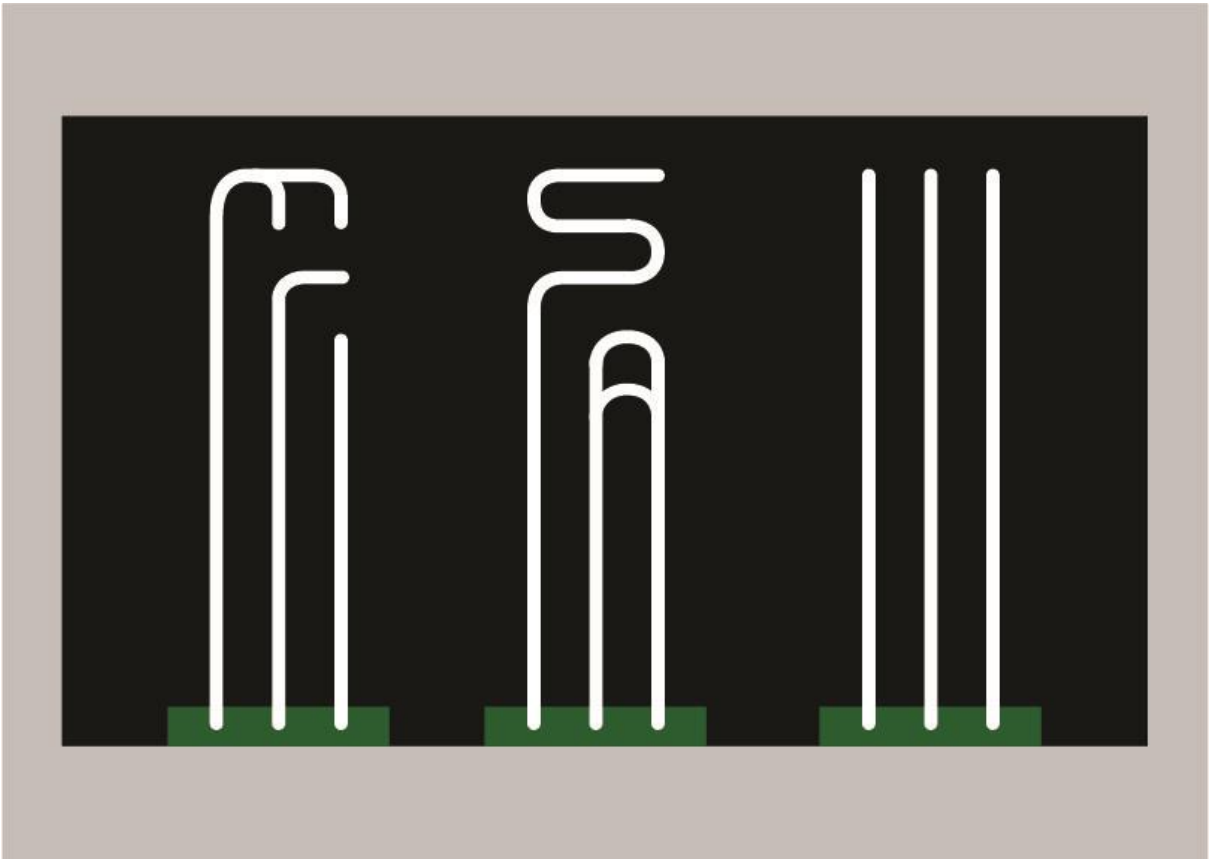
L'arène est un bon exemple de démonstration de l'évolution constante du projet. En effet, le concept ayant déjà évolué avec le fait qu'il est passé d'un groupe d'étudiants à un autre, il a aussi subi d'autres changements notoires après la visite du directeur de musée, M. Patrick Gyger.

9.1 L'arène initialement prévue



- Pour 7 robots (2 en réserve)
- 2 bases de rechargement
- Dimensions: 65 x 55 cm
- Surface: blanche, aimantée et lisse
- Disposition: verticale

9.2 L'arène finale



- Pour 8 robots (2 en réserve)
- 3 bases de rechargement
- Dimensions: env. 70 x 55 cm (à vérifier)
- Surface: noire, aimantée et lisse
- Disposition: penchée (angle d'environ 30°, les bases en bas)
- Forme: rectangle avec des bords blancs

Nous remarquons donc qu'une base supplémentaire a été placée, que le fond blanc est devenu noir, les lignes - par conséquent - blanches, et avec un dégradé. L'image ci-contre l'illustre. Mais la réalité sur le terrain est encore bien différente puisqu'il est possible que nous rencontrions d'autres problèmes sur place lors de l'installation. Affaire à suivre!

10 Conclusion

Durant ce projet, nous avons rencontré de nombreuses difficultés. La nouvelle bibliothèque utilisée ne possédait malheureusement pas de documentation officielle développée- De plus, elle produisait un certain nombre de comportements étranges, par exemple une limitation apparente ne permettant pas d'envoyer des commandes à plus de quatre robots de manière simultanée. La compréhension et l'utilisation de cette bibliothèque nous a donc posé de nombreux problèmes et la complexité du code contenu a rendu très compliquée la résolution de bugs et la modification de certaines fonctionnalités. Un autre problème notoire a été la difficulté d'automatiser les tests, c'est-à-dire que nous avons été obligés de transférer le code sur chaque robot individuellement à chaque modification importante, et comme nous en possédions une petite dizaine, cela s'est avéré très chronophage.

Malgré tout cela, nous avons réussi à réaliser le projet en remplissant les objectifs du cahier des charges. Il est en effet possible aux robots d'effectuer des danses que nous avons prédéfinies. Afin que les spectateurs profitent au maximum des effets visuels du robot, nous avons varié les couleurs de manière intelligente et ludique pour procurer un rendu des plus attrayants. Une des fonctionnalités phares que nous avons implémentée étant le rechargement autonome: lorsqu'un robot atteint un seuil prédéfini de batterie faible, il cherche une station de recharge en suivant une des lignes situées sur le sol. Si cette station est déjà occupée, il ne force pas le passage, mais au contraire il se retire de manière intelligente et se met à la recherche d'une autre base libre. Cette fonctionnalité est extrêmement utile car dans le contexte d'un musée, il serait très pénible et inconvenant pour la personne responsable de devoir recharger les robots individuellement chaque fois qu'ils en ont besoin. Il faudrait presque un employé à temps complet chargé de cette mission. Un autre atout de notre projet tient dans le fait que l'on peut utiliser un capteur digital (le Touchrotation) permettant à un visiteur - en rapprochant sa main - de donner l'ordre aux robots de démarrer, plutôt que de devoir faire cela de manière moins ludique à l'aide d'un ordinateur par exemple. Le visiteur n'est donc pas seulement passif, mais est un acteur à part entière de l'exposition, ce qui est idéal dans le cadre d'un musée aussi ludique que la Maison d'Ailleurs.

En conclusion, ce projet nous a permis de nous familiariser avec les robots Elisa-3, de découvrir ses forces mais aussi ses faiblesses. Finalement, nous sommes arrivés à développer des cas d'utilisation fonctionnels et qui correspondent aux attentes de la Maison d'Ailleurs. En outre, nous sommes fiers d'avoir pu fournir un tel résultat d'après les difficultés rencontrées et l'évolution constante du concept.

L'année du robot est une année spéciale et nous sommes confiants dans le fait que les Elisa-3 apporteront leur pierre à l'édifice de l'exposition Portrait-Robot.