POLYTECH LILLE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

ART EMBARQUÉ

Rapport de projet

Informatique Microélectronique Informatique

Quatrième année

Encadrants :

Alexandre Boé / Thomas Vantroys

Élèves :

Thibaut SCHOLAERT / Jean-michel Tournier

Année :

2014/2015

Sommaire

Introduction

La création artistique peut être enrichie par l'apport de nouvelles technologies. Dans le cadre de notre 4ème année à Polytech Lille dans le département Informatique-Microélectronique-Automatique, nous nous sommes lancés dans un projet qui nous a particulièrement intéressés, qui consiste en l'accompagnement de la performance d'un danseur par la création d'une œuvre. Cette œuvre, exclusivement visuelle, sera projetée en direct sur le fond de la scène où évoluera le danseur.

Ce projet nous a donc permis de mettre en pratique des connaissances acquises pendant l'année ainsi que de développer notre capacité à gérer entièrement un projet en commençant par l'élaboration d'un cahier des charges jusqu'à une réalisation finale.

Nous commencerons par faire une présentation générale du projet. Ensuite, nous verrons le déroulement du projet en détails. Enfin nous expliciterons notre point de vue et nos ressentis après coup par rapport à ce projet.

I. Présentation générale du projet

1. Objectif du projet

L'objectif de ce projet est de créer un système permettant de générer des œuvres d'art qui viennent compléter la performance d'un danseur. Cette œuvre est exclusivement visuelle. Le but est de projeter cette œuvre en direct sur le fond de la scène où évoluera le danseur.

Notre but est donc de récupérer la position du danseur sur la piste, et des récupérer des informations sur ses mouvements afin de pouvoir générer la création visuelle en adéquation et en rythme avec les mouvements de l'artiste.

2. Description du projet

Tout d'abord, nous avons besoin d'un système permettant de récupérer des informations sur les mouvements du danseur. Nous avons, pour ce projet, choisi d'utiliser des *accéléromètres*, que l'on place sur chacune des mains du danseur grâce à leur intégration sur des gants. Un système embarqué sur le costume de l'artiste transfère les données de ces deux accéléromètres vers un ordinateur par le biais d'une transmission sans fil.

Une Kinect placée au plafond fait office de caméra. Elle récupère la position du danseur sur la piste en deux dimensions par rapport au sol. Nous traitons les données récupérées par le système embarqué et la Kinect sur l'ordinateur afin de créer l'œuvre visuelle. L'œuvre à réaliser fut dans un premier temps un disque qui se déplaçait dans une zone en fonction de la position détectée par le Kinect. La couleur, la taille et la forme, quant à eux, évoluent en fonction des données renvoyées par les accéléromètres.

3. Choix techniques : matériel et logiciel

Pour le système embarqué, nous avons choisi d'utiliser un *Arduino lilypad*. C'est un circuit imprimé sur lequel se trouve un microcontrôleur que nous pourrons programmer pour analyser les signaux envoyés par les accéléromètres. Pour ce qui est de l'envoi des données sans fil, nous auront besoin d'un couple de modules *XBee*. Ce sont des modules utilisant le protocole de transmission ZigBee, permettant la communication à courte distance par ondes radio. Il y aura donc un module XBee qui sera relié à l'arduino et le second qui sera branché en USB sur l'ordinateur et réceptionnera les données envoyées par son homologue. Pour alimenter tout ça, nous avons également besoin d'une alimentation portable, à savoir une batterie. Le système embarqué sera donc formé de l'arduino, des deux accéléromètres, d'un module XBee et d'une batterie.

Ensuite, nous aurons besoin d'une Kinect. Cette caméra nous permettra de localiser le danseur.

Nous aurons ensuite bien entendu besoin d'un ordinateur qui nous permettra de récupérer les données nécessaires et de faire tourner le programme qui créera l'œuvre en direct. À cet ordinateur seront donc reliés le module XBee récepteur ainsi que la Kinect.

Le programme qui sera téléversé sur l'arduino est codé en langage C. Pour ce qui est de la création du visuel, et du traitement des données, nous utilisons un logiciel appelé Processing. Il s'agit d'un environnement de développement libre basé sur le langage Java.

II. Déroulement du projet

1. Préambule

Afin de compléter notre cahier des charges et pour une meilleure organisation, nous avons séparé préalablement le déroulement du projet en plusieurs étapes. La première étant la récupération des données de la Kinect et des accéléromètres. La deuxième étape est l'envoi et la réception des données par XBee, nous passons ensuite par le traitement de ces données via Processing. Pour finir, la concrétisation et projection de l'œuvre sur scène en direct.

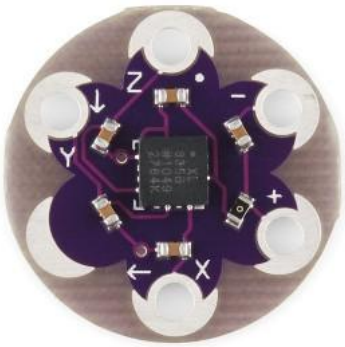
Pour une meilleure efficacité, nous nous sommes répartis les différentes tâches. L'un s'occupait de la partie système embarqué, à savoir la récupération des données des accéléromètres avec l'arduino lilypad et la communication XBee, tandis que l'autre s'attelait à la gestion de la Kinect et l'utilisation du logiciel Processing.

2. Gestion du système embarqué

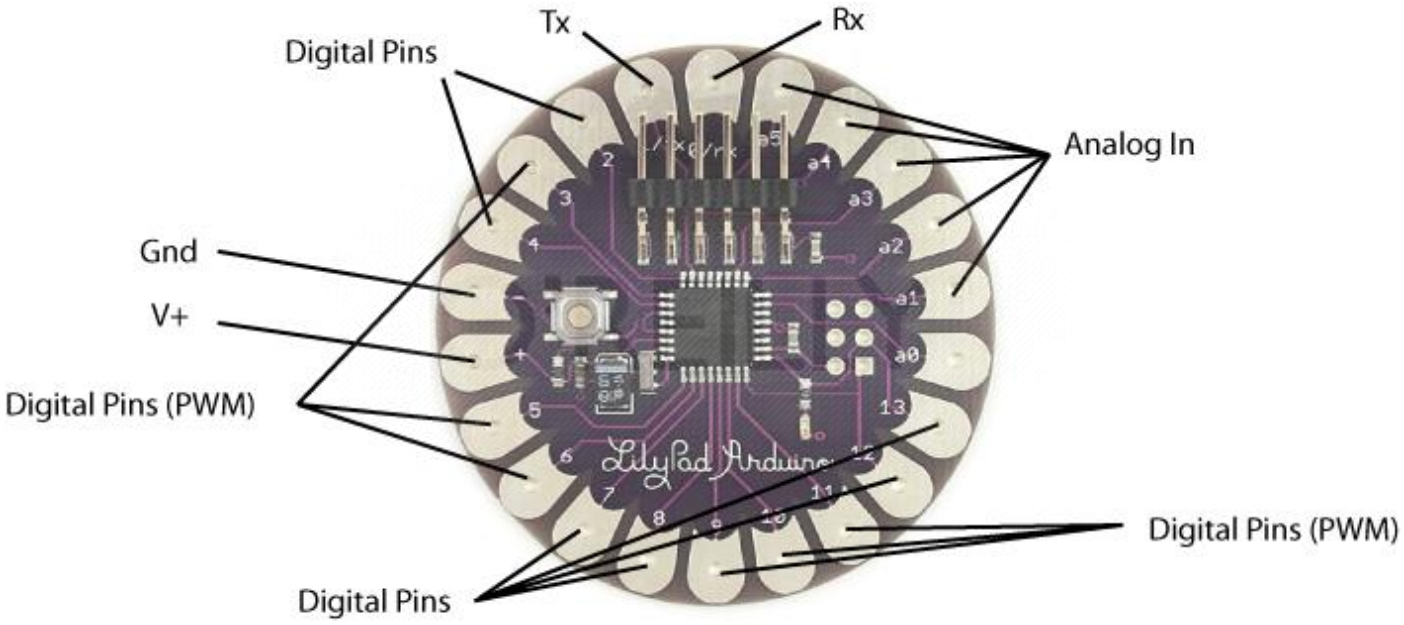
Nous avons donc, pour le système embarqué, choisi d'utiliser un Arduino lilypad. Cette carte équipée d'un microcontrôleur ATmega328V a été conçue pour une utilisation sur des vêtements ou des e-textiles. Il peut donc être cousu sur le costume du danseur avec du fil conducteur ainsi que les autres composants tels que les accéléromètres ou le module XBee fixé sur une platine de support.

2.1 Montage

Pour commencer, nous avons réalisé un premier petit montage avec seulement l'arduino et un accéléromètre afin de faire quelques tests et nous familiariser avec le fonctionnement du système. L'accéléromètre est alimenté en le connectant aux broches + et - de l'arduino et les données x, y et z sont branchées sur trois entrées analogiques de l'arduino (a0, a1 et a2). Le second accéléromètre est quant à lui connecté sur les 3 autres (a3, a4 et a5).



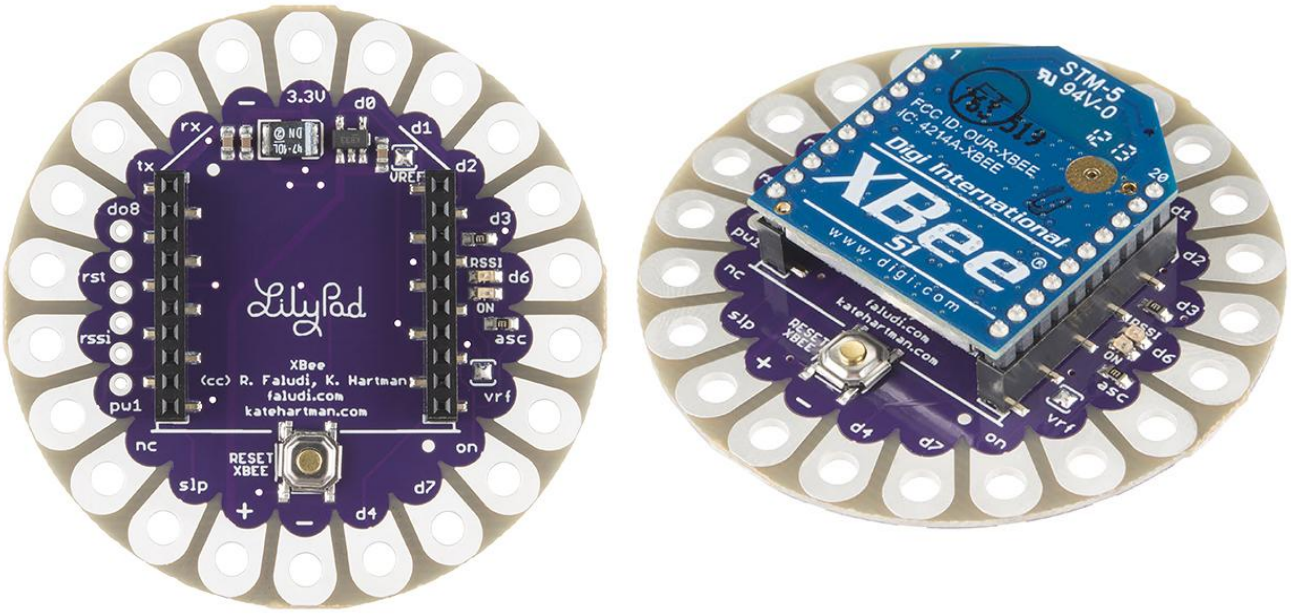
Accéléromètre



Arduino lilypad

Après avoir testé et vérifié le fonctionnement, nous avons pu essayer de transmettre ces données via les modules XBee. Après avoir fait quelques recherches sur leur fonctionnement, nous avons pu savoir comment réaliser notre montage.

Le module XBee possède deux broches séries, à savoir Rx pour la réception et Tx pour la transmission. Pour brancher le module XBee à l'arduino, il suffit de connecter la broche Rx du XBee avec la broche Tx de l'arduino et connecter la broche Tx du XBee avec la broche Rx de l'arduino. Il faut également l'alimenter en 3.3V en branchant les pins 1 et 10 aux broches 3.3V et ground de l'arduino. On peut voir ci-dessous le support qui embarquera le module XBee et permettra la couture sur le costume. On peut observer une broche + et une broche - (en bas) qui sont celles que l'on connecte à l'arduino. Nous n'utilisons que ces quatre broches.

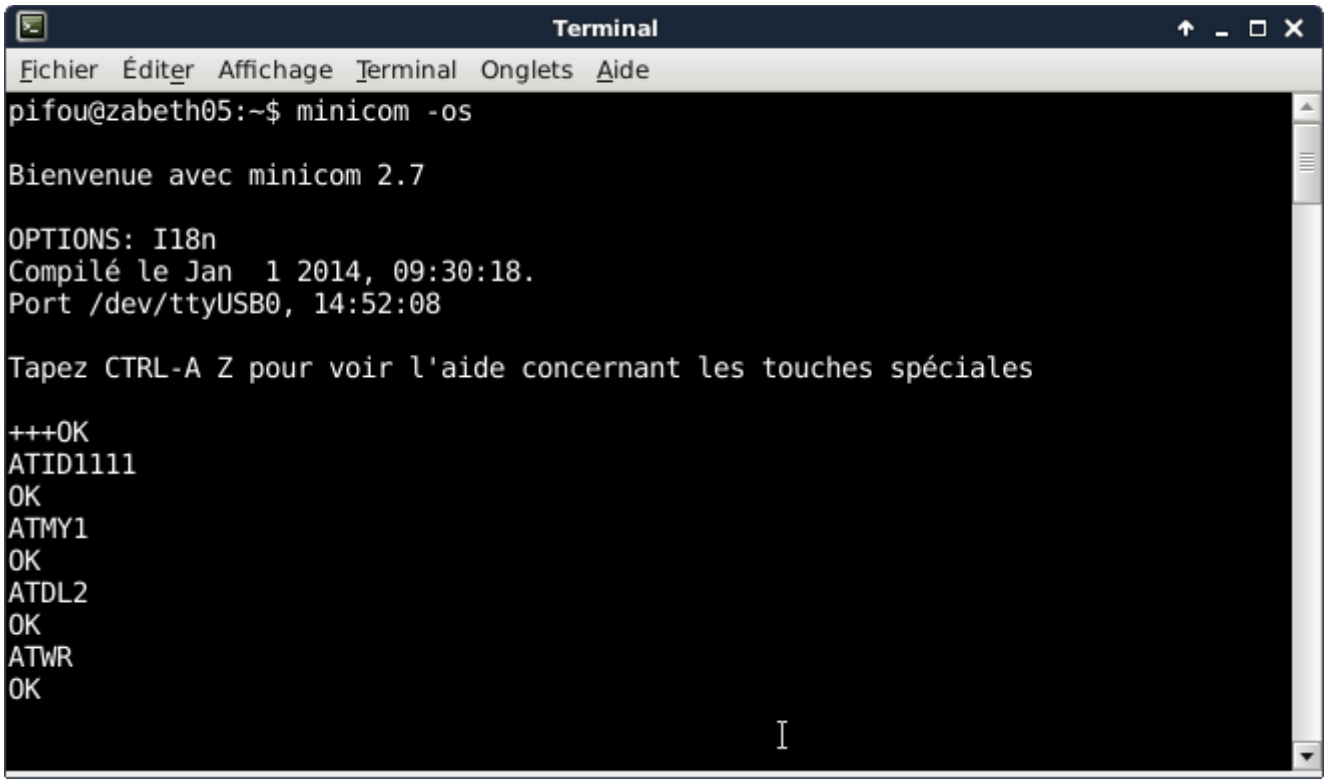


Platine de support pour module XBee

2.2 Configuration des XBee

Afin que les modules XBee puissent communiquer, il faut les configurer. En effet, il faut savoir que pour qu'il y ait une communication entre eux, certains paramètres doivent être correctement définis. Pour ce, nous les avons configurés un par un en les branchant en USB à un ordinateur. Nous avons utilisé la liaison série et nous nous sommes connectés à eux grâce à minicom.

Tout d'abord, il faut déterminer un identifiant réseau afin qu'aucun autre XBee ni vienne polluer la communication. Ensuite, il faut déterminer, pour chacun, un identifiant personnel et un identifiant destination. Il faut également vérifier que leur baud rate soit identique. Pour la configuration, nous devons donc nous connecter au module XBee avec minicom, puis taper la commande '+++'. Minicom doit alors nous renvoyer un 'OK'. Une fois ce 'OK' reçu, nous sommes entrés dans la configuration et pouvons donc déterminer un identifiant réseau avec la commande 'ATID', un identifiant personnel avec la commande 'ATMY' et un identifiant destination avec la commande 'ATDL' comme ceci :



configuration XBee sur minicom

Nous avons donc choisi le réseau 1111 et chacun des identifiants des XBee sont 1 et 2. Initialement, nous avons un baud rate à 9600 bauds puis nous sommes passés par la suite à 57600 bauds pour un meilleur débit et donc une plus grande rapidité.

Passons maintenant au code qu'on a intégré à l'arduino.

2.3 Code Arduino

Cette partie n'a pas été la plus compliquée étant donné que nous avions déjà réalisé un mini projet sur des tourelles gérées par un arduino. Nous étions donc déjà familiarisés avec le mode de fonctionnement de ce genre de carte électronique. Nous avons donc codé en langage C pour une meilleure performance et une meilleure garantie de fonctionnement, ce qui nous amène au code et au makefile disponibles *en annexe.* Nous avons donc 5 fonctions principales :

- void init_serial(long int speed)
- void send_serial(unsigned char c)
- unsigned char get_serial(void)
- void ad_init(unsigned char channel)
- unsigned int ad_sample(void)

La fonction init_serial prend en paramètre le baud rate et donc initialise la vitesse en bauds. La fonction send_serial envoie un caractère placé en paramètre au port série. La fonction get_serial lit la valeur reçue par le port série et la renvoie. La fonction ad_init initialise le convertisseur analogique numérique, en prenant en paramètre le numéro de l'entrée analogique. La fonction ad_sample permet de récupérer la valeur reçue par le convertisseur sur l'entrée analogique précédemment initialisée.

Nous avons, grâce à ces fonctions, réalisé notre main qui fonctionne ainsi:

On commence par initialiser la vitesse en bauds donc ici 57600 ainsi que les variables. Ces variables servent à stocker les valeurs lues par le convertisseur analogique numérique, c'est à dire les valeurs retournées par les accéléromètres. La variable received nous sert à stocker la valeur lue par get_serial. Lorsque cette valeur est un 'd', on peut envoyer les données avec la fonction send_serial. En réalité, on gère donc l'envoi des données avec Processing, la communication entre les modules XBee étant dans les deux sens. Le XBee connecté à l'ordinateur enverra à son module appairé la valeur 'd' pour signaler qu'il est prêt à recevoir les données des accéléromètres. On envoie successivement les valeurs x, y et z du premier accéléromètre puis les valeurs du second.

Grâce à notre makefile, il suffit de taper la commande 'make upload' pour téléverser le code dans le microcontrôleur de l'arduino. Tout le système embarqué est maintenant opérationnel.

Maintenant que nous avons vu le câblage et la programmation de ces modules, il faut s'intéresser à leur intégration sur le costume du danseur.

2.4 Intégration sur le costume

En fin de projet, nous avons dû intégrer le système embarqué à un costume. Pour cela, nous avons commencé par coudre les accéléromètres avec un fil conducteur sur des gants noirs basiques. Ces cinq fils par gant, sont reliés à une nappe qui se prolongera jusqu'à atteindre l'arduino cousu sur le buste du costume.

Intégration sur le gant

Le système à accrocher sur les vêtements, avec Lilypad, XBee
Alimentation et connexion de gants

Afin de simplifier la réutilisation de ce projet, nous avons choisi de monter l'arduino, le module XBee et la platine d'alimentation sur une « plaque » de tissus, qu'il suffira de fixer à l'aide d'épingles sur les vêtements du « testeur » .

Vue du système complet, avec les deux gants et la batterie

3. Gestion de la Kinect via Processing

Notre premier objectif pour ce qui est de la Kinect était de la faire fonctionner. Cette caméra est équipée de plusieurs capteurs : un CCD détectant donc la couleur un autre détectant la profondeur (grâce à un projecteur infrarouge qui lui est couplé), un Micro à reconnaissance vocale que nous n'utiliserons pas et un capteur motorisé pour suivre les déplacements que nous n'utiliserons pas non plus. Nous ne nous sommes servis que de la lentille pour localiser la tête du danseur et ainsi pouvoir connaître sa position.

3.1 Prise en main

Nous avons tout d'abord installé le logiciel Processing qui nous permet de pouvoir gérer les données de la Kinect, créer une interface graphique ou encore afficher en direct ce que perçoit la Kinect. Nous avions ensuite besoin d'intégrer la bibliothèque Simple Open NI, qui possède toutes les fonctions nécessaires à l'utilisation et l'exploitation de la Kinect.

Une fois ces étapes complétées, nous avons pu tester quelques exemples de codes comme Depth Image. Cette exemple nous permet d'obtenir une image en fonction de la profondeur comme ci-dessous :

Depth Image Kinect

Nous sommes donc partis de ce code pour récupérer les distances entre la personne et la Kinect.

3.2 Code Processing

Nous sommes tout d'abord partis sur un tracking basique du point le plus proche vu par la Kinect. On commence par créer un rectangle de taille 640x480 noir. Nous repérons ensuite le point le plus proche détecté par la Kinect et nous dessinons à cet endroit un polygone qui possède initialement 50 côtés, ce qui nous donne l'illusion de voir un cercle. Ensuite, nous avons besoin d'utiliser les valeurs des accéléromètres pour colorer, changer de taille ou de forme le polygone ainsi créé. Nous avons donc besoin de créer un tableau de 6 entiers qui va stocker les valeurs des accéléromètres reçues par le XBee. Dans notre programme, nous écrivons sur le port série la lettre 'd' correspondant au signal qui indique que les données des accéléromètres peuvent être envoyées comme expliqué dans *la partie 2.3*. Nous lisons donc les 6 valeurs des accéléromètres à chaque fois que la lettre 'd' est écrite sur le port série. Une fois ces valeurs récupérées, nous les utilisons chacune pour une tache :

- La valeur 1 : l'axe x de l'accéléromètre 1 pour la couleur rouge du point
- La valeur 2 : l'axe y de l'accéléromètre 1 pour la couleur bleue du point
- La valeur 3 : l'axe z de l'accéléromètre 1 pour le nombre de côtés du polygone
- La valeur 4 : l'axe x de l'accéléromètre 1 pour la taille du point
- La valeur 5 : l'axe y de l'accéléromètre 1 n'est pas utilisé pour le moment
- La valeur 6 : l'axe z de l'accéléromètre 1 n'est pas utilisé pour le moment

Nous n'utilisons pas les axes y et z du deuxième accéléromètre pour l'instant. Ces valeurs sont comprises entre 0 et 255.

La couleur du point est changée continuellement au cours du temps. Par contre, pour la forme et la taille, il en est autrement. Lorsque les valeurs 3 ou 4 sont suffisamment petites ou grandes, ce qui correspond à une forte accélération, donc un mouvement brusque, on lance alors un processus léger, un thread, qui va gérer la modification de taille ou de forme suivant la donnée concernée. Le thread fonctionnant en parallèle avec le programme principal, il nous permet de modifier le point de façon temporelle et assez simplement.

On peut voir dans le code la création de notre polygone qui se faire lors de l'appel à la fonction :

```
polygon(avgX, avgY, size, edges);
```

Cette fonction prend en paramètre la position du point le plus proche, c'est à dire du danseur, avec avgX et avgY. Elle prend également size qui sera modifié lorsque la valeur sera inférieure à 60 ou supérieure à 200. Et prend en dernier paramètre edges qui sera modifié lorsque sa valeur sera inférieure à 45 ou supérieure à 210. Ces valeurs de seuil ont été choisies suite à des tests de notre part pour trouver un juste milieu pour pouvoir déclencher les threads mais pas non plus à l'excès. Néanmoins il peut être nécessaire d'ajuster ces valeurs lors de tests en situation réelle.

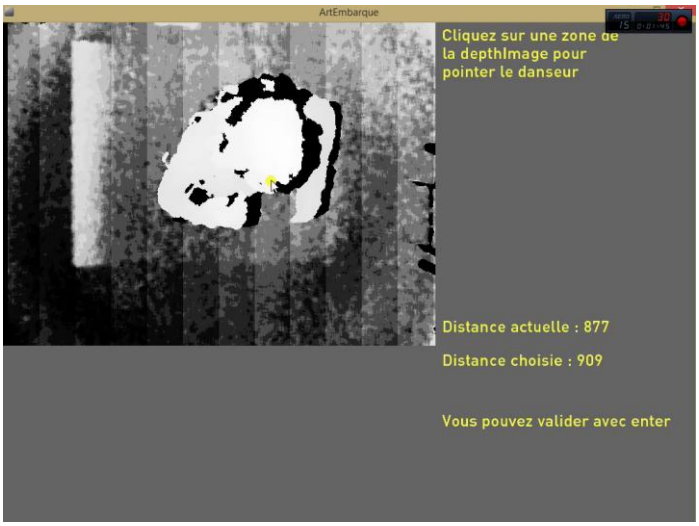
3.2 Evolutions du code

Notre code Processing, contrairement au code Arduino qui ne pouvait pas être amélioré, a connu une évolution tout au long de notre projet. Tout d'abord pour une question de fluidité au niveau de la couleur. Au début, lorsque les mouvements étaient trop brusques, les valeurs des accéléromètres changeaient de manière importante et cela créait un changement de couleur trop brusque et désagréable. Nous avons donc pensé à une technique pour atténuer ce changement de couleur. Le principe est simple, lorsqu'un changement trop important, nous avons choisi supérieur à 100 (toujours sur l'échelle 0 à 255), nous choisissons de ne changer la valeur que de 30. Ainsi, la couleur ne peut pas changer brusquement et on gagne en fluidité.

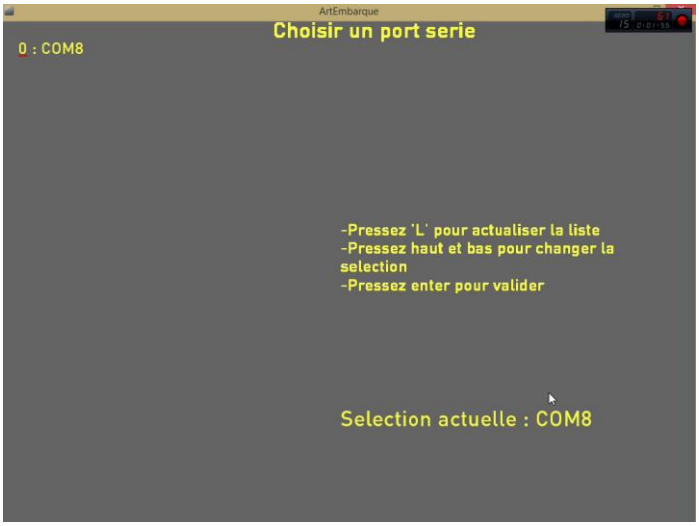
Ensuite, pour gérer la forme. Au début, nous avions choisi de créer un cercle normal. Pour pouvoir changer de forme, nous avons donc changé notre cercle en polygone avec un nombre important de côté.

Nous avons également amélioré la détection du danseur en faisant une moyenne entre tous les points se trouvant dans une plage de distance définie, ce qui évite les battements du point et nous a également fait gagné en fluidité.

Nous avons également vers la fin du projet, pensé à rendre notre système plus facile d'utilisation notamment par l'intégration d'une interface graphique de paramétrage du système. Cette interface graphique permet à l'utilisateur calibrer la distance du point à partir de laquelle la Kinect doit détecter. Il peut ensuite choisir le port USB sur lequel se trouve le module XBee, ça lui évite ainsi de devoir modifier les valeurs en dur dans le code.



Première vue, sélection de la distance du danseur



Seconde vue : sélection du port série, ici sous Windows, mais c'est aussi valable sous linux avec la liste des ttyX et ttyUSBX

III. Nos impressions sur ce projet

1. Difficultés rencontrées

Nous avons, lors de ce projet rencontré plusieurs difficultés. En premier temps au niveau de l'arduino. Initialement codé avec le software arduino afin de réaliser rapidement quelques tests, nous sommes rapidement passés à un code C beaucoup plus propre et efficace. Cependant, nous avons rencontré quelques erreurs. Notre code semblait fonctionnel et ne générait pas d'erreur lors du make upload. Nous ne savions donc pas si le problème venait des modules XBee, de notre code, du Makefile ou autre. Il y avait en fait un souci au niveau de la fonction d'initialisation de la vitesse, erreur que nous avons rapidement localisé et réparé. Cependant le système ne fonctionnait toujours pas, et finalement, nous avons upload le code sur l'arduino à partir d'un autre ordinateur et, par surprise, cela fonctionnait. Nous n'avons pas compris d'où venait ce problème. Ces petits soucis nous ont malheureusement ralentis dans l'avancement du projet.

Nous avons ensuite connu quelques soucis de fluidité. Nous étions à la base à 9600 bauds et avions testé plusieurs baud rate pour savoir lequel était le plus haut et stable que nous pouvions fixer. Nous n'avions pas de communication à 115200 mais un bon fonctionnement à 57600. Malgré une bonne communication entre les modules XBee, l'image manquait de fluidité. Nous avons donc modifié un peu notre programme Processing pour pouvoir comprendre. Nous nous sommes vite rendu compte que c'était à cause d'erreurs de traitement qui rendait de manière intempestive le point noir, et donnait une vision saccadée du point. Nous avons donc sagement décidé d'installer la Kinect sur Linux et faire tourner l'intégralité du projet sur Linux. Malgré quelques problèmes rencontrés, notamment le fait de devoir tout lancer en super utilisateur, nous avons finalement réussi à faire tourner la Kinect sous Linux, avec Processing. La gestion du port série sous Windows semble donc poser, de temps en temps, quelques problèmes. Cependant une fois la version « de développement » du programme remplacée par la version finale, et celle-ci n'utilisant plus la vue complète de l'image infrarouge « DepthMap », la fluidité semble être de retour sous Windows.

Nous avons également rencontré quelques problèmes avec le code Processing, mais sans soucis majeur empêchant notre progression. Ces difficultés étaient en général dues à la nouveauté pour nous de cet environnement de développement. Par exemple nous est venu la difficulté de rafraîchir la zone d'affichage afin de pouvoir modifier un texte. Il faut donc pour cela effacer l'ancien texte, ce qui correspond à réafficher dessus une couleur, Processing fonctionnant par couches. Le problème était alors qu'en effaçant tout l'écran, certains textes n'étaient pas redessinés car les fonctions les créant n'étaient appelés qu'une seule fois. Nous avons donc « rusé » en ne redessinant que des rectangles de la couleur de fond sur les textes à modifier. Cette solution invisible pour l'utilisateur nous a donc permis de pallier à ce problème assez rapidement.

Une des parties de ce projet que nous avions fortement sous-estimé était l'intégration du système sur le danseur. En effet le système, la couture et la disposition des éléments sur un tissu demandent une bonne réflexion. De plus, nous ne sommes pas vraiment familiers avec la couture, et celle-ci avec le fil conducteur demande encore plus de précaution afin d'avoir des contacts fonctionnels. Ainsi, le système fini tel que présenté en *II.2.4* nous as demandé beaucoup de temps.

Afin d'obtenir des résultats visuels à présenter, nous avons essayé pendant les vacances avec une connaissance danseuse de tester le dispositif. Nous avons essayé d'utiliser une salle des fêtes de mairie, mais nous avons essuyé plusieurs refus, soit par ce que la salle était déjà réservée pour d'autres activités ou que cette demande n'était pas vraiment traditionnelle puisque nous ne voulions évidemment pas louer la salle pour un week-end. Les tests ont donc eu lieu dans l'un de nos domiciles, mais cela n'était pas vraiment adapté car la hauteur de plafond ne laissait pas vraiment un champ assez grand à la Kinect, et l'idée de la projection en direct du flux d'image via vidéoprojecteur était aussi évidemment plus compliqué. Malheureusement, le fonctionnement n'as pas été parfait, la communication série s'arrêtant sans que l'on ne comprenne pourquoi au bout de quelques secondes. Néanmoins nous avons quand même pu réaliser quelques images qui serviront à la vidéo de présentation.

2. Evolution

Par manque de temps nous n'avons pas pu faire évoluer le projet, afin de supporter deux danseurs. Néanmoins nous avions bien réfléchi à la façon de le faire. Il faudrait donc dupliquer le matériel Arduino embarqué, et c'est Processing qui devra associer chacun des deux ports séries aux zones détectes. Il faut donc, au lieu de détecter tous les points dans une plage de distance puis faire une moyenne, créer deux ensembles de points, qui seront distingués par un espace trop grand avec des précédents points détectés. Ainsi, lorsqu'un point est détecté mais qu'il est trop éloigné de la zone des autres, il viendra s'ajouter dans le deuxième groupe. Enfin, afin de ne pas inverser l'association entre les groupes et les ports série, on pourra associer une couleur grâce à une LED de couleur placée sur la tête du danseur. En effet, dans la boucle qui détecte la distance des points, on peut aussi détecter la couleur.

3. Bilan

Nous finissons ce projet sur un bilan globalement positif, étant donné que nous avons respecté notre cahier des charges. Cependant, par manque de chance et de temps, il n'y malheureusement pas eu la collaboration avec des professionnels et artistes espérée en début de projet. Nous n'avons donc pas pu réaliser de test réel au cours du projet. Cela aurait pu nous aider et nous aiguiller sur nos choix d'évolution de notre système. Il est vrai que quelques tests au fur et à mesure de l'avancement du projet auraient pu grandement nous aider dans l'évolution et le peaufinage de notre projet. Nous avons choisi certaines animations de façon assez arbitraire, plus en fonction de notre découverte de l'environnement de développement graphique que d'une motivation artistique ou d'une demande particulière.

Le projet fonctionne donc sur la plupart des systèmes d'exploitation tels que Linux, Windows ou encore Mac OS X grâce au fonctionnement multiplateforme de Processing. (Il faut néanmoins effectuer les opérations nécessaires au fonctionnement de la Kinect sur chaque OS, disponible à l'adresse suivante :

https://code.google.com/p/simple-openni/wiki/Installation . De plus, Processing permet assez facilement l'export vers un exécutable spécifique par OS, ce qui accroit la simplicité d'utilisation pour l'éventuel utilisateur final. Il n'y a alors plus à installer Processing, il faut seulement avoir java installé sur l'OS.

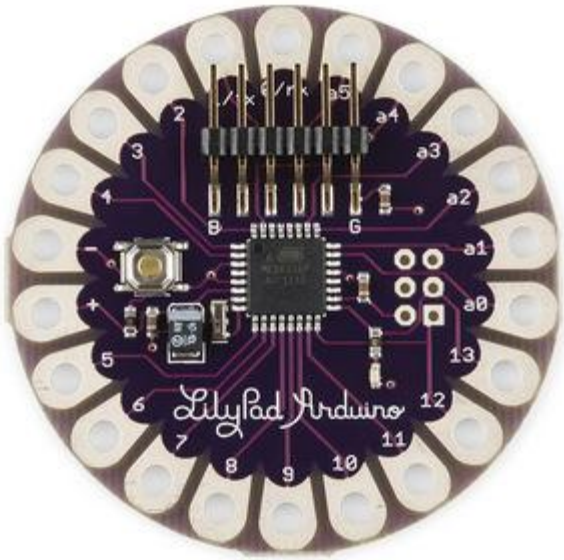
Enfin, le projet n'est finalement pas si couteux dans son ensemble, le matériel étant assez commun. Il peut donc assez facilement être réutilisé et amélioré. En effet, même la Kinect d'ancienne génération peut se trouver d'occasion pour moins de 40€, il faut juste trouver ou fabriquer un adaptateur pour l'utiliser sur un PC.

Conclusion

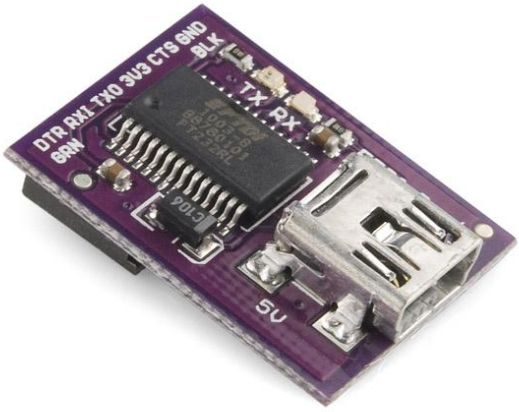
Pour conclure, nous avons créé un système qui accompagne la performance d'un danseur par la création d'une œuvre visuelle dans le cadre d'un projet réalisé durant notre quatrième année d'école d'ingénieur dans le département Informatique, Microélectronique et Automatique.

Durant ces douze semaines, nous avons pu mettre en pratique nos connaissances et en en développer de nouvelles, mettre à l'épreuve notre savoir-faire et gérer entièrement un projet.

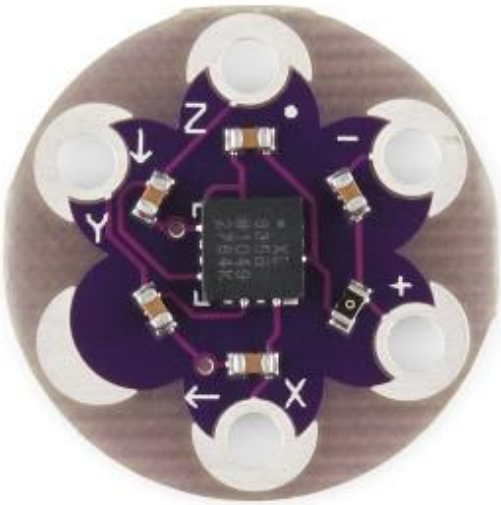
Le cahier des charges fixé en début de projet a été respecté et, bien qu'il n'y ait pas eu de collaboration avec des professionnels, nous sommes satisfaits du travail réalisé. De nombreuses améliorations sur ce projet peuvent cependant être amenées pouvant rendre le système plus interactif ou plus diversifié visuellement par exemple.



Arduino lilypad

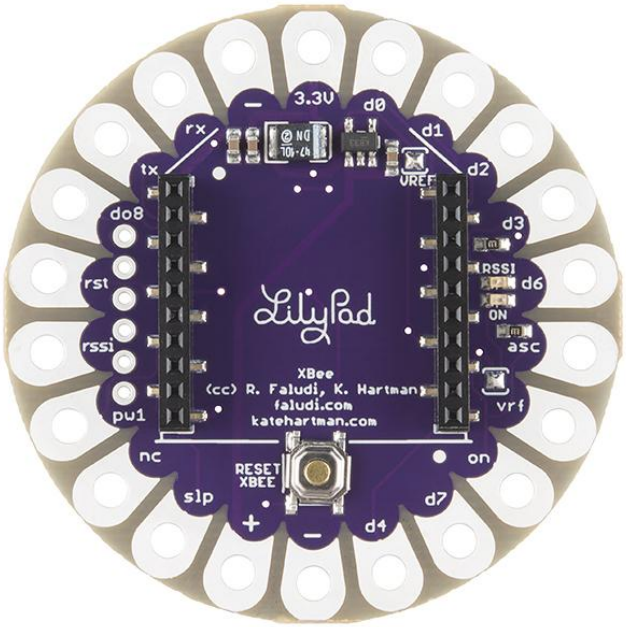


Lilypad FTDI basic breakout

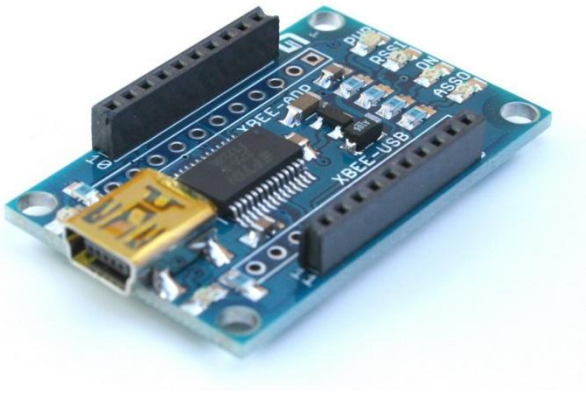


Accéléromètre

Module XBee



Support XBee



Adaptateur XBee-USB

Kinect

CODE C POUR L'ARDUINO LILYPAD

```
#include <avr/io.h>// for the input/output register
#include <util/delay.h>
// For the serial port

#define CPU_FREQ  16000000L   // Assume a CPU frequency of 16Mhz

void init_serial(longint speed)
{
/* Set baud rate */
  UBRR0 = CPU_FREQ/(((unsignedlongint)speed)<<4)-1;

/* Enable transmitter & receiver */
  UCSR0B =(1<<TXEN0 |1<<RXEN0);

/* Set 8 bits character and 1 stop bit */
  UCSR0C =(1<<UCSZ01 |1<<UCSZ00);

/* Set off UART baud doubler */
// UCSR0A &= ~(1 << U2X0);
}


void send_serial(unsignedchar c)
{
  loop_until_bit_is_set(UCSR0A, UDRE0);
  UDR0 = c;
}


unsignedchar get_serial(void)
{
  loop_until_bit_is_set(UCSR0A, RXC0);
return UDR0;
}


// For the AD converter

void ad_init(unsignedchar channel)
{
  ADCSRA|=(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
  ADMUX|=(1<<REFS0)|(1<<ADLAR);
  ADMUX=(ADMUX&0xf0)|channel;
  ADCSRA|=(1<<ADEN);
}


unsignedint ad_sample(void)
{
  ADCSRA|=(1<<ADSC);
while(bit_is_set(ADCSRA, ADSC));
return ADCH;
}


// For the I/O

void output_init(void)
{
  DDRB |=0b00010000;
}
```

```
int main(void)
{
  init_serial(57600);
//declaration des variables unsigned char pour etre sur
un octet
unsignedchar data1x=0;
unsignedchar data1y=0;
unsignedchar data1z=0;
unsignedchar data2x=0;
unsignedchar data2y=0;
unsignedchar data2z=0;
unsignedchar received;

while(1)
{
//on recupere la data recue par A0, A1, A2, A3, A4 et A5
      ad_init(0);
      data1x=ad_sample();
      ad_init(1);
      data1y=ad_sample();
      ad_init(2);
      data1z=ad_sample();
      ad_init(3);
      data2x=ad_sample();
      ad_init(4);
      data2y=ad_sample();
      ad_init(5);
      data2z=ad_sample();

      received = get_serial();
if(recieved=='d')
{
      send_serial(data1x);
      send_serial(data1y);
      send_serial(data1z);
      send_serial(data2x);
      send_serial(data2y);
      send_serial(data2z);
}
}
return0;
}
```

MAKEFILE

```
export CC = avr-gcc

export MCU = atmega328p
export TARGET_ARCH = -mmcu=$(MCU)

export CFLAGS =  -Wall -I. -DF_CPU=16000000 -Os #-g
export LDFLAGS = -g $(TARGET_ARCH) -lm -Wl,--gc-sections #       -Os

TARGET = usb
TERM = /dev/ttyUSB1
CPPFLAGS = -mmcu=$(MCU)
PGMER = -c stk500v1 -b 57600 -P $(TERM)
PGMERISP = -c stk500v1 -b 115200 -P $(TERM)
ARVDUDECONF= -C /usr/local/arduino/arduino-0022/hardware/tools/avrdude.conf
export DUDE = /usr/bin/avrdude -F -v -p $(MCU) $(AVRDUDECONF)

C_SRC = $(wildcard *.c)
OBJS = $(C_SRC:.c=.o)

all: $(TARGET).hex

clean:
	rm -f *.o *.hex *.elf

%.o:%.c
	$(CC) -c $(CPPFLAGS) $(CFLAGS) $< -o $@

$(TARGET).elf: $(OBJS)
	$(CC) $(LDFLAGS) -o $@ $(OBJS)

$(TARGET).hex: $(TARGET).elf
	avr-objcopy -j .text -j .data -O ihex $(TARGET).elf $(TARGET).hex
	avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -O ihex $(TARGET).elf eeprom.hex

upload: $(TARGET).hex
	stty -F $(TERM) hupcl # reset
	$(DUDE) $(PGMER) -U flash:w:$(TARGET).hex
```

CODE PROCESSING

Fichier 1 : initialisation / gestion des écrans

```
import SimpleOpenNI.*;
import processing.serial.*;

Serial myPort;

int currentScreen=0;
PFont font;
color backColor;
color textColor;

//mouseInfo needs
int lastMouseX=0;
int lastMouseY=0;
int distanceChoosed=0;

//comSelect
int comNum;
int comMax=-1;
StringList dev;

//animView
int[] inByte = newint[6];
int[] last = newint[3];
int edges=50;
int size=20;
int edges_lunched=0;
int nbThreads=0;
int nbThreadsMax=8;



SimpleOpenNI  context;

voidsetup()
{
  size(1024, 768);
  context = new SimpleOpenNI(this);
  if (context.isInit() == false)
  {
    println("Can&apos;t init SimpleOpenNI, maybe the
camera is not connected!");
    exit();
    return;
  }

  // mirror is by default enabled
  context.setMirror(true);

  // enable depthMap generation
  context.enableDepth();

  // enable ir generation
  context.enableRGB();

  font = loadFont("DINPro-Medium-48.vlw");
  textFont(font);
  textSize(24);
  backColor= color(100, 100, 100);
  textColor= color(250, 250, 70);

  background(backColor);

}
```

```
voiddraw()
{
  switch(currentScreen) {
  case 0:
    mouseInfo();
    break;
  case 1:
    comSelect();
    break;
  case 2:
    animDraw();
    break;
  default:
    background(backColor);
    break;
  }
}

// Actions entree user


voidmousePressed() {
  switch(currentScreen) {
  case 0:
    newClic(mouseX, mouseY);
    break;

  case 1:
    break;

  case 2:
    break;

  default:
    background(70);
    break;
  }
}

voidkeyPressed() {
  switch(currentScreen) {


  case 0:
    if
((key==ENTER||key==RETURN)&&distanceChoosed!=0)
{//return pour les macs
      currentScreen++;
      background(backColor);
    }
    break;

  case 1:
    if (keyCode==UP) comDrawLine(-1);
    elseif (keyCode==DOWN)comDrawLine(1);
    elseif (key==&apos;l&apos;||key==&apos;L&apos;)
comList();
    if ((key==ENTER||key==RETURN)&&comMax!=-1) {
      myPort = new Serial(this,
Serial.list()[comNum], 57600);
      currentScreen++;
      smooth();
      noStroke();
      last[0]=128;
      last[1]=128;
      last[2]=128;
      background(backColor);
      fill(backColor);
    }
    break;
  case 2:

    break;
  default:
    background(255);
    break;
  }
}
```

Fichier 2 : Config distance

```
void mouseInfo() {

  fill(backColor);
  noStroke();
  rect(640, 0, 384, 480);
  strokeWeight(1);

  fill(textColor);
  // update the cam
  context.update();
  int[]   depthMap = context.depthMap();
  // draw depthImageMap
  image(context.depthImage(), 0, 0);
  text("Cliquez sur une zone de la depthImage pour pointer le danseur", 640+10, 10, 280, 500);
  if (lastMouseX!=0) {
    ellipse(lastMouseX, lastMouseY, 16, 16);//representer le clic
    String actu = "Distance actuelle : "+str(depthMap[lastMouseX + lastMouseY * context.depthWidth()]);
    text(actu, 640+10, 460);
  }
}

void newClic(int lastClicX, int lastClicY) {
  background(backColor);

  int[]   depthMap = context.depthMap();

  if (lastClicX<=640&&lastClicY<=480) { // clic effectue dans la zone de depthImage

    distanceChoosed=depthMap[lastClicX + lastClicY * context.depthWidth()];
    String s = "Distance choisie : "+str(distanceChoosed);
    if (distanceChoosed!=0) {
      text(s, 640+10, 510);
      text("Vous pouvez valider avec enter", 640+10, 600);
      lastMouseX=lastClicX;
      lastMouseY=lastClicY;
    } elsetext("Attention distance invalide", 640+10, 550);
  } elsetext("DANS LA ZONE DE DEPTHIMAGE!", 640+10, 400, 300, 600);
}
```

Fichier 3 : Config série

```
void comSelect() {
  textSize(32);
  text("Choisir un port serie", 400, 24);
  textSize(24);
  String s="-Pressez &apos;L&apos; pour actualiser la liste\n";
  s+="-Pressez haut et bas pour changer la selection\n";
  s+="-Pressez enter pour valider";
  text(s, 500, 300, 500, 500);
}

void comDrawLine(int countChange) {
  if (comMax!=-1) {//ne rien faire si la liste est vide

    if (countChange==0) comNum=0;
    elseif ((countChange==-1)&&(comNum>0)) comNum--;
    elseif ((countChange==1)&&(comNum<comMax)) comNum++;

    strokeWeight(3);
    stroke(backColor);

    //effacer les anciennes lignes
    for (int i=0; i<=comMax; i++) {
      if (i>29)  line(200, 24*(i-28)+2, 224, 24*(i-28)+2);
      elseif (i>9) line(25, 24*(i+2)+2, 48, 24*(i+2)+2);
      elseline(25, 24*(i+2)+2, 35, 24*(i+2)+2);
    }
    //tracer la nouvelle ligne
    stroke(255, 0, 0);
    if (comNum>29)  line(200, 24*(comNum-28)+2, 224, 24*(comNum-28)+2);
    elseif (comNum>9) line(25, 24*(comNum+2)+2, 48, 24*(comNum+2)+2);
    elseline(25, 24*(comNum+2)+2, 35, 24*(comNum+2)+2);


    fill(backColor);
    noStroke();
    rect(470, 550, 500, 100);
    strokeWeight(1);
    fill(255);

    textSize(32);
    fill(textColor);
    text("Selection actuelle : "+dev.get(comNum), 500, 600);
    textSize(24);
  }
}

void comList() {
  background(backColor);
  int count=0;
  dev = new StringList();
  dev.insert(0, Serial.list()); //on recopie la liste serial dans dev, eviter de la recharger X fois


  for (String i : dev ) {
    fill(textColor);
    String s=str(count)+" : "+i;
    if (count<30) text(s, 24, 24*(count+2));
    elsetext(s, 200, 24*(count-28));
    comMax=count;
    count++;
  }
  comDrawLine(0);
}
```

Fichier 4 : Gestion de l'animation

```
void animDraw() {

  getSerial();

  //var for
  float sumX = 0;
  float sumY = 0;
  float count = 0;

  int index;
  int[]   depthMap = context.depthMap();

  // update the cam
  context.update();
  for (int y=0; y < context.depthHeight (); y+=2)
  {
    for (int x=0; x < context.depthWidth (); x+=2)
    {
      index = x + y * context.depthWidth();
      if ((depthMap[index] <
(distanceChoosed+50))&&(depthMap[index]
>(distanceChoosed-50))) {
        sumX += x;
        sumY += y;
        count++;
      }
    }
  }

  fill(0, 0, 0, 20);
  rect(0, 0, 1024, 768);
  fill(255);

  //launch threads
  if (edges_lunched==0&&(nbThreads<nbThreadsMax)) {
    if (inByte[3]<60||inByte[3]>200)
thread("changeSize");
    if (inByte[2]<45||inByte[2]>210)
thread("changeEdges");
  }

  //draw points
  if (count != 0) {
    float avgX = sumX/count;
    float avgY = sumY/count;
    fill(inByte[0], 0, inByte[1]);
    polygon(avgX*1.6, avgY*1.6, size, edges); //*1.6
pour passer de 640*480 en 1024*768
  }
}

void polygon(float x, float y, float radius, int
npoints) {
  float angle = TWO_PI / npoints;
  beginShape();
  for (float a = 0; a <TWO_PI; a += angle) {
    float sx = x + cos(a) * radius;
    float sy = y + sin(a) * radius;
    vertex(sx, sy);
  }
  endShape(CLOSE);
}
```

```
void getSerial() {
int temp;
  myPort.write(&amp;apos;d&amp;apos;);

  while (myPort.available () > 0) {
    for (int i=0; i<2; i++) {//pour x y du premier
capteur filtrage
      inByte[i] = myPort.read();
      temp=inByte[i]-last[i];
      if (temp>100) inByte[i] = last[i]+30;
      if (temp<-100) {
        inByte[i] = last[i]-30;
      }
      last[i]=inByte[i];
    }
    for (int k=2; k<=5; k++) {// enregistrement des
autres valeurs
      inByte[k] = myPort.read();
    }
  }
}


void changeEdges() {
  edges_lunched=1;
  size+=4;
nbThreads++;
  while (edges>5) {
    edges--;

    delay(7);
  }
  size+=4;

  delay(300);

  size-=4;

  while (edges<50) {
    edges++;
    delay(10);
  }
  size-=4;
  edges_lunched=0;
  nbThreads--;
}

void changeSize() {
  nbThreads++;
  for (int i=0; i<10; i++) {
    size++;
    delay(20);
  }
  for (int i=0; i<10; i++) {
    size--;
    delay(13);
  }
  nbThreads--;
}
```