RAPPORT DE PROJET
“Sextoy connecté”

Département Informatique, Microélectronique, Automatique

Tuteurs :
Xavier Redon
Alexandre Boé

Étudiants :
Cédric Roussel
Thomas Stievenard

Remerciements

Dans un premier temps, nous tenons à remercier l'équipe pédagogique du département "Informatique, Microélectronique et Automatique" de Polytech Lille pour les compétences de bases qui nous ont permis d'avancer au mieux dans ce projet.

Nous remercions également M. REDON et M. BOÉ, nos tuteurs de projet, qui ont su se rendre disponible pour nos questions et problèmes.

Nous remercions également M. FLAMEN pour l'aide ainsi les conseils apportés pour la conception des cartes électroniques de ce projet.

Sommaire

Introduction

Au cours de notre 2ème année d'étude en spécialité "Informatique, Microélectronique et Automatique" à Polytech Lille, nous avons été amené à choisir un projet que nous effectuons tout au long du second semestre de cette année.

Ce projet est le moyen d'appliquer des notions amorcées en cours, d'en découvrir de nouvelles et de s'exercer à la gestion d'un projet. Nous avons donc dû élaborer un cahier des charges, une liste de matériel dimensionnée à notre problématique, une liste de tâches que nous nous sommes répartie en fonction de nos affinités.

Le sujet que nous avons choisi s'intitule "Sextoy connecté", il consiste à permettre aux couples vivant à distance, ou aux curieux, de pratiquer des rapports intime, à distance et de manière sécurisée.

Dans un premier temps, nous présenterons le projet et son cahier des charges. Ensuite nous nous attarderons sur la conception du projet, notamment sur la conception des prototypes, la programmation embarquée et android, ainsi que leur liaison. Enfin, nous tirerons un bilan du projet, sa gestion, les problèmes rencontrés, les erreurs commises, etc.

I. Présentation du projet

I.1. Objectif et cahier des charges

L'objectif de ce projet est de développer un système d'échange entre deux partenaires distants afin de rendre plus facile la vie de couple à distance.

Pour ce faire, le dispositif sera constitué de deux sextoys connectés (un par appareil génital désiré) à leur téléphone portable respectif.
Une application mobile permettra aux utilisateurs de s'échanger des données de manière sécurisé, ceci au travers d'un serveur embarqué dans le téléphone portable.

Les données échangées pourront être l'accélération ou la température. Elles seront traitées et transmises au sextoy qui adaptera sa vitesse, son retour vibratoire ou sa position en conséquence.

L'appareil masculin (dédié à la femme), doit être composé d'un servomoteur, un vibreur, un sonde de température, un accéléromètre.

L'appareil féminin (dédié à l'homme), doit être composé de 4 servomoteurs, de vibreurs, et d'un accéléromètre.

Les Deux appareils doivent disposer d'un micro-contrôleur(Arduino mini), un module de transmission/réception bluetooth afin de se connecter au téléphone.

L'application mobile doit mettre en place une communication cryptée avec ou sans serveur, effectuer un échange de données en bluetooth avec le sextoy connecté. Elle peut aussi effectuer une vidéoconférence, enregistrer les meilleurs moments et éventuellement pouvoir les partager, effectuer une recherche de partenaire aléatoire. Et si il restait encore du temps, mettre en place un réseau social interne à Polytech.

II. Conception des sextoys

La conception de sextoy nécessite de respecter à la fois des normes d'hygiène mais également d'ergonomie.

II.1. Utilisation de silicone

Nous avons décidé de nous renseigner sur l'utilisation de silicone pour effectuer certaines pièces du sextoy. Cependant, au vu du temps que nous avons passé sur les différentes parties, nous n'avons pu mettre en applications les informations obtenues.

Néanmoins, nous gardons en tête les conseils de Mario Sanz, ingénieur chercheur à l'INRIA.

Ce dernier nous a expliqué les différents points importants sur lesquels nous baser afin de choisir au mieux un produit en accord avec le projet. Ne serait-ce que pour le choix de la dureté, il existe une échelle propre au silicone ("Shore hardness scale"), nous pensions au départ rechercher un silicone le plus mou possible, cependant, il faut savoir que le plus le silicone est mou, plus il a tendance à être "collant". Finalement, après discussion, un shore de 40A (correspondant à la dureté d'une gomme) suffisamment mince ferait l'affaire pour la conception des prototypes.

Il nous a également expliqué que le silicone peut être travaillé à l'aide d'additifs et sous cloche à vide afin d'améliorer le moulage, l'absence de "bulles" sur la surface de la pièce et la souplesse. Nous pouvons cependant travailler sans tout cela car il n'est pas spécialement important, pour notre projet, de rechercher des propriétés physiques homogènes dans tout le sextoy.

Pour ce qui est de la création de moules, nous pouvons les réaliser via imprimante 3D. Il existe un bon nombre de tutoriels sur internet pour s'y référer en cas de soucis.

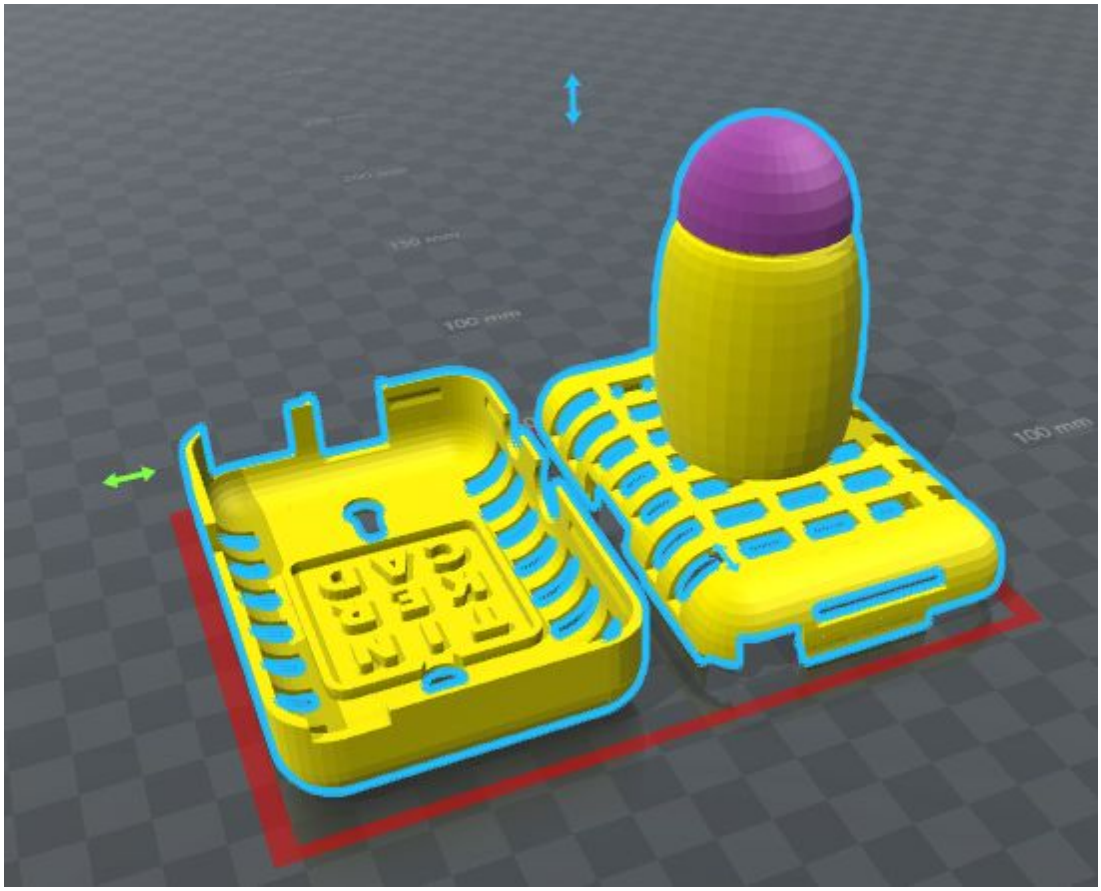
II.2. Conception 3D

Finalement, nous avons décidé de nous lancer dans l'impression 3D de ce à quoi auraient pu ressembler les sextoys. Et avons imprimé la partie "pénis connecté".

Pour ce faire, nous avons utilisé l'application Tinkercad, utilisable via navigateur web. L'avantage d'un tel outil est sa grande communauté qui permet un accès simple et rapide à une multitudes de projets dont on pourrait s'inspirer pour générer nos pièces.

Pour imprimer le prototype, nous avons mis à profit l'accès au Fabricarium de Polytech Lille.

Ce prototype se divise en 3 pièces : un boîtier (pour y loger le PCB), un cylindre (afin de reproduire la forme désirée) et une "tête" qui devait à l'origine être un moule en silicone (pour y loger le PCB du capteur de température)



Design 3D du sextoy

II.3. Conception des PCB

Afin de diminuer un maximum la taille des cartes électroniques, nous utilisons principalement des composants CMS.

La conception des cartes peut-être effectué à l'aide de logiciels de CAO tels qu'Altium ou Eagle. Dans notre cas, nous avons utilisé Eagle.

Ce travail consiste à créer une librairie regroupant l'intégralité des composants nécessaires et, ensuite, de créer un schéma du circuit. Une fois le schéma terminé, il ne restera plus qu'à faire le routage et exporter les fichiers au format "Gerber".

Pour ce projet, nous aurons besoin de deux cartes, une par prototype. Chaque carte embarquera des composants similaires, c'est pourquoi nous nous contenterons d'abord d'un seul PCB pour effectuer des tests. Si ces derniers sont concluants, nous pouvons graver le second.

Arbitrairement, nous avons choisi de commencer par le PCB du pénis. Ce dernier était censé regrouper : un support pour l'arduino, un accéléromètre, un module bluetooth et un capteur de température.

Cependant, compte tenu du fait que le capteur de température doit être au plus prêt de la zone intéressée, nous avons décidé de le faire sur un pcb à part, qui serait relié au PCB principal mais potentiellement coulé dans un moule en silicone.

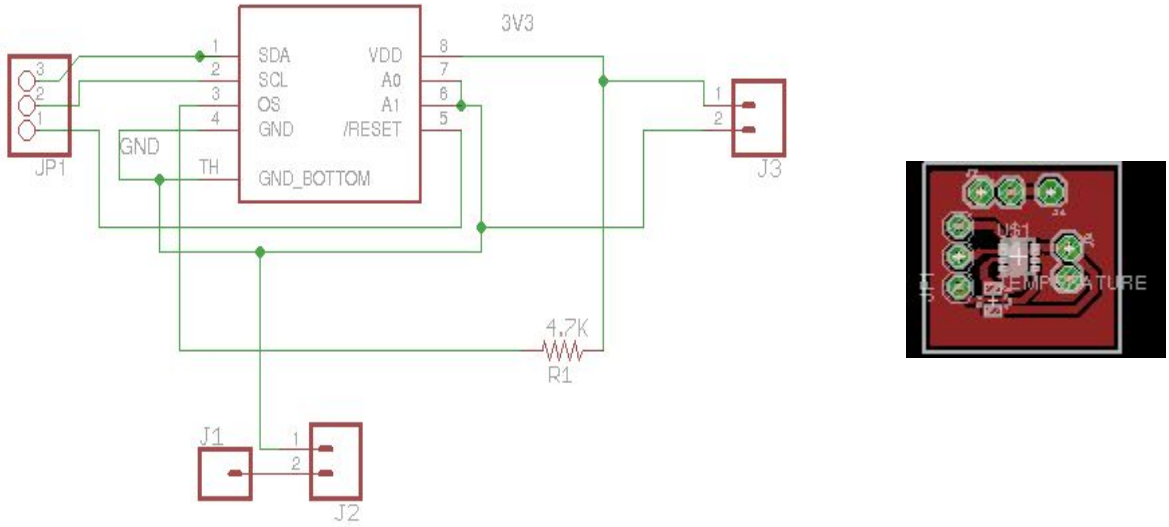
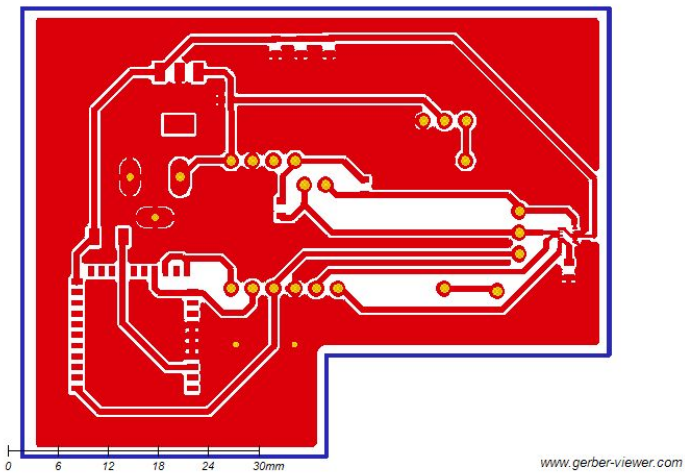


Schéma et PCB du capteur de température

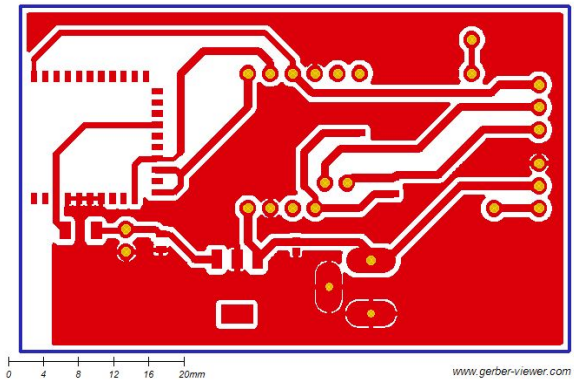
Nous avons expérimentés des problèmes dûs aux dimensions de l'accéléromètre (tout à droite sur le PCB), l'utilisation d'un boîtier CMS de type TDFN10 2x2 était optimiste pour la réalisation du projet, compte tenu du matériel à disposition.



Première version du PCB pour le pénis

C'est pourquoi une seconde carte a été gravée, cette fois sans l'accéléromètre et en suivant quelques conseils pour une réalisation optimale.

Notamment, nous avons veillé à enlever le plan de masse de sous le module bluetooth, afin d'être sûr que l'antenne de ce dernier capte les transmissions.



Version finale du PCB du pénis (sans accéléromètre)

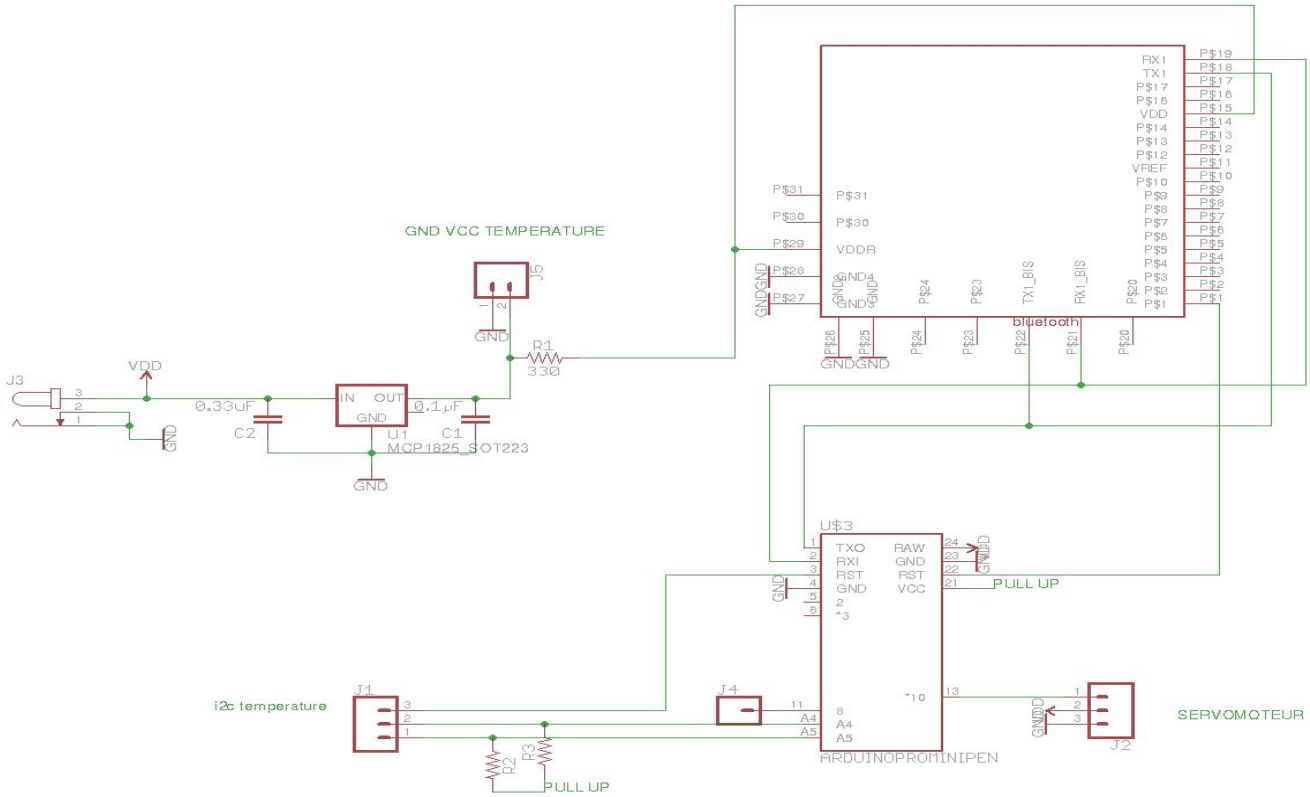


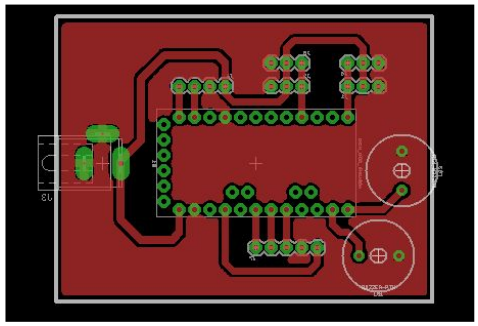
Schéma de la version finale du PCB du pénis

Les points importants du schéma de ce PCB sont la présence de résistances en PULL-UP sur les voix SCL, SDA (A4, A5) du bus I2C et la résistance R3 permettant de driver le courant désiré en alimentation du module Bluetooth.

Une fois le PCB du pénis réalisé et les composants correctement soudés, nous avons pu passer aux tests qui nous ont fait remarquer que le module bluetooth était défaillant. C'est pourquoi, pour le PCB du vagin, nous avons décidé d'aller au plus simple. Nous n'utiliserons pas

le module bluetooth initialement prévu mais un module utilisable en traversant. (HC-06) De même pour l'accéléromètre, nous utiliserons un accéléromètre analogique en traversant.

Aux vues de ces modifications, il n'est plus nécessaire d'avoir un régulateur 3.3V comme pour le PCB du pénis car ce dernier servait surtout à s'assurer que le module soit correctement alimenté. Au lieu de ça, si nous avons besoin de 3.3V, nous mettrons à profit le convertisseur interne à l'arduino pro mini.



PCB du vagin

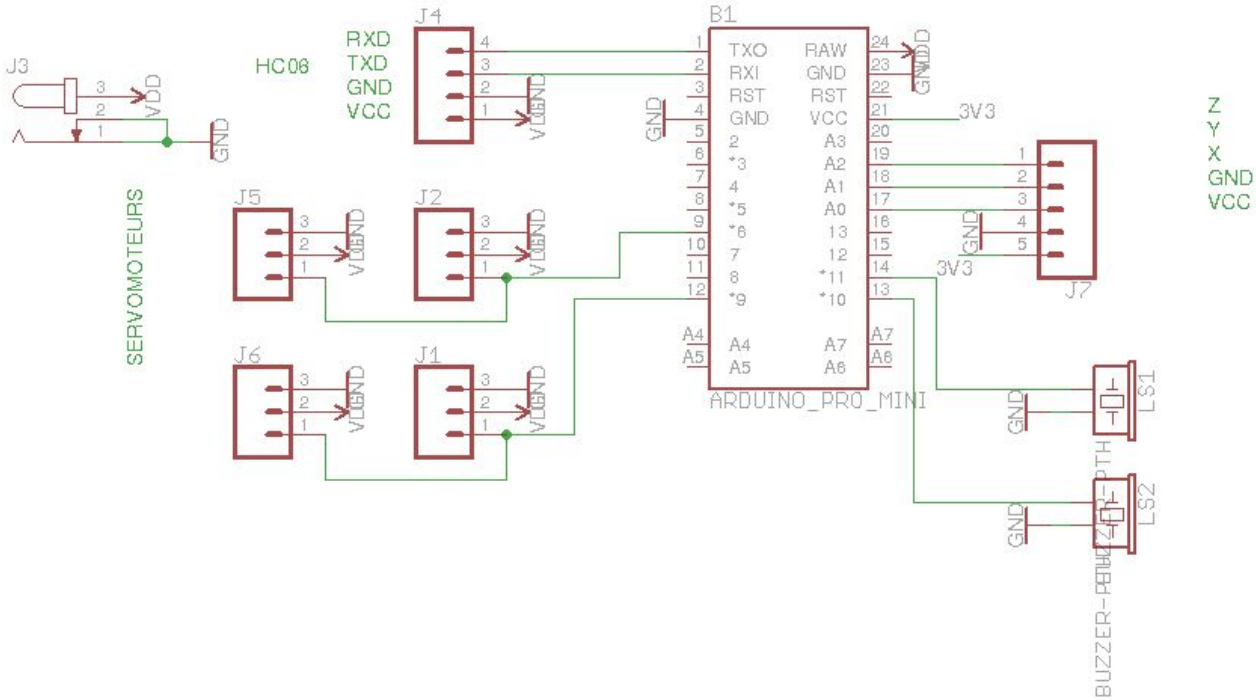


Schéma du PCB du vagin

Finalement, le schéma consiste simplement à préparer des zones enfichables directement reliées aux broches utilisées sur l'arduino et les niveaux de tensions utilisés.

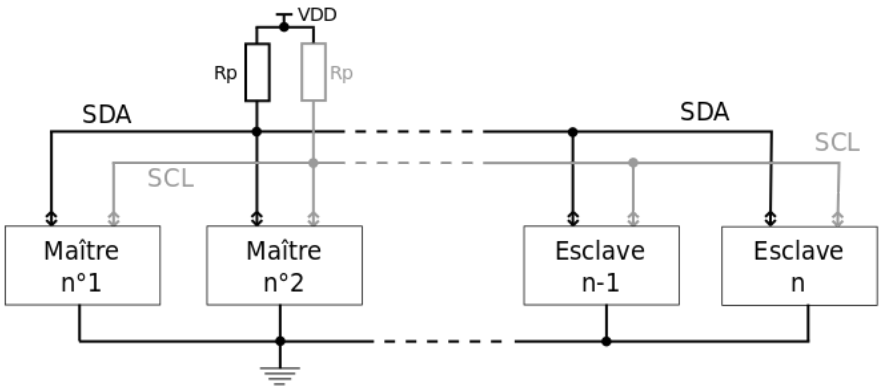
III. Programmation

III.1. Partie embarquée

La partie embarquée est constitué de la gestion de capteurs en IIC, de servomoteurs par PWM et de vibreurs. Nous nous attarderons sur les deux premières gestions dans la mesure où l'activation d'un vibreur reste triviale.

III.1.1. Gestion par IIC

Pour le pénis, nous utilisons un capteur de température fonctionnant en IIC. Ce type de communication consiste en un réseau maître-esclave, il permet d'avoir plusieurs capteurs sur un unique bus de données composés de deux fils : SDA, SCL.



Tout d'abord, nous devons initialiser le bus IIC de l'arduino. Pour ce faire, il suffit de préciser la fréquence désirée, ici 400KHz, dans les registres TWSR (prescaler) et TWBR. (fréquence désirée suivant la fonction TWBR = (F_{cpu} - 16 F_{scl}) / (2 F_{scl} Prescaler))

Comme il peut y avoir potentiellement plusieurs capteurs sur ce bus, il faut respecter un certain protocole afin de récupérer la valeur du capteur de température. Attardons nous sur la lecture d'une valeur. Il faut :

1. Lancer un message de start
2. Envoyer l'adresse du capteur de température (0x98 dans notre cas)
3. Envoyer l'adresse du registre qui nous intéresse (0x00 pour celui contenant la température), ceci placera le pointeur de registre du capteur sur cette adresse
4. Restart la communication pour laisser la main au capteur
5. Envoyer l'adresse du capteur en ajoutant 1 (Ceci permet de stipuler que nous désirons lire la valeur pointée)
6. Lire la partie haute de la température
7. Lire la partie basse de la température
8. Envoyer un message de stop

Afin de s'assurer que les messages sont bien reçus, nous vérifions les statuts du bus IIC se trouvant dans le registre TWSR de l'arduino. Ces derniers nous indiquerons par exemple si nous avons reçu un ACK en cas d'émission, preuve de la bonne réception du message.

Si un statut n'est pas cohérent avec le déroulement normal, nous arrêtons la mesure.

Pour plus de détail, vous pouvez vous référer au code "iic.c".

III.1.2. Gestion des servomoteurs

Afin de procurer une sensation de massage au partenaire masculin, nous désirons commander 4 servomoteurs, deux par deux. Pour ce faire, nous devons générer deux signaux PWM sur 8bits. La création de signaux PWM en AVR se fait via l'utilisation de vecteurs d'interruptions.

Le choix des vecteurs d'interruption est donc important. La PWM se doit d'être régulière, ainsi, les interruptions de ces signaux doivent passer en priorité par rapport à toutes les autres. Les vecteurs d'interruptions d'un Atmega utilisent un ordre de priorité simpliste. Pour un Timer X donné, plus X est petit, plus sa priorité est grande. Ainsi, nous avons décidé d'utiliser les Timer0 et Timer1 pour nos deux signaux PWM.

L'initialisation d'un timer en PWM se fait en configurant les registres TCCRXA, TCCRXB et en activant les pins concernés en tant qu'output. Les initialisations des deux timers sont similaires, c'est à dire qu'on configure les registres pour générer des "FAST PWM" (la consigne est directement la valeur placée dans le registre OCRXA), "NON INVERTING MODE" (le signal carré commence par un état haut) avec un prescaler de 1024 (nous avons remarqué que plus la consigne était lente, plus le servomoteur adhérait à la consigne).

La seule différence entre Timer0 et Timer1 est le fait que la PWM en Timer1 doit être spécifiée sur 8bits (elle peut aller jusque 16bits).

Comme dit précédemment, la consigne est entrée dans le registre OCRXA. Il faut savoir que la consigne est une consigne de position. Nous devons donc utiliser un troisième timer pour la faire varier périodiquement.

Nous utilisons donc le Timer2 en comparaison (CTC mode), avec un prescaler de 1024 et une valeur maximale en OCR2A. Le but est de générer des interruptions le plus lentement possible. Bien que ce timer ne pourra pas empiéter sur les performances des PWM, il était inutile d'imposer une fréquence élevée.

La période d'interruption du Timer2 est donc de 32.64ms. Ce qui reste trop rapide pour effectuer des changements de consignes de positions à répétition (Afin d'induire la sensation de massage). Nous avons donc choisi d'ajouter un délai via l'utilisation d'un compteur. Après quelques tests nous avons jugé que compter jusqu'à 15, pour passer la période à 500ms, était viable pour atteindre l'objectif des servomoteurs.

III.1.3. Détermination de l'excitation

Pour déterminer l'excitation de l'utilisateur, chaque prototype possède son capteur dédié. Nous utilisons toujours le Timer2 pour mesurer de manière périodique les valeurs des capteurs. En fonctions des valeurs, nous jugeons un niveau d'excitation.

Pour le prototype pénis, le capteur de température mesure la température interne de la partenaire. En sachant que la température initiale est de 37°C et qu'elle oscille de 2°C en fonction de l'excitation et de la période, nous avons considéré que chaque 0.5°C, nous augmentons d'un niveau d'excitation.

Pour le prototype du vagin, nous utilisons finalement un accéléromètre analogique. Ce dernier a une valeur fluctuant beaucoup et nous souhaitons mesure l'entrain de l'utilisateur. C'est pourquoi nous sommes parti sur une lecture de 20 valeurs par axe avant d'en déduire une estimation de la dérivée avant de calculer un niveau d'excitation propre à l'utilisateur.

III.2. Application mobile

Toute l'application mobile a été créé sur Android studio, un IDE créé par Google pour développer des applications Android. Il permet d'éditer les fichiers Java et les fichiers de configuration d'une application Android. Le logiciel dispose aussi d'une fonction Debug, qui permet de faire tourner l'application et d'afficher des informations importantes, comme les messages d'erreur et la ligne de code Java qui produit l'erreur.

Afin d'avoir une application fonctionnelle il faut manipuler:
- les fichiers Java, qui définissent le fonctionnement du code
- les fichiers .xml se trouvant dans le dossier /res , en particulier ceux se trouvant dans le dossier /res/layout, qui définissent l'affichage et les éléments interactifs(bouton, texte éditable) de l'application
-le "manifest" qui détermine les permissions de l'applications, et déclare les activités de l'application.

Les activités sont les seuls programmes Java de l'application pouvant afficher sur l'écran du téléphone.

III.2.1. Gestion des SMS

III.2.1.a. Envoi de SMS

Dans le but de synchroniser deux téléphones et de communiquer les valeurs nécessaires à la transmission, nous avons choisi d'utiliser des SMS.

Dans un premier temps nous avons créé une application pour envoyer les SMS:
L'application était très simple, l'envoi de SMS se faisait en deux lignes :

```
// initialisation d'une variable sms gérant les sms
SmsManager sms = SmsManager.getDefault() ;
// envoie un sms comportant le texte de la variable message
// de type String au numéro contenu dans la variable num de type String
sms.sendTextMessage(num, null, message, null, null);
```

Il suffit ensuite d'utiliser deux "EditText", un pour entrer le numéro de téléphone et le second pour le message à envoyer. Et enfin ajouter un objet "Button" qui déclenche la récupération des informations sur les EditText et l'envoie du sms

III.2.1.b. Réception de SMS

Dans un deuxième temps il fallait recevoir un SMS, ou plutôt savoir qu'un nouvel SMS avait été reçu . Pour cela il faut utiliser un thread qui est appelé par interruption, avec la réception d'un SMS comme vecteur d'interruption.
Ce thread effectue ce qui est demandé dans sa fonction onReceive(), comme par exemple lire les informations contenues dans le SMS:

```
public class IncomingSms extends BroadcastReceiver {
    final SmsManager sms = SmsManager.getDefault();
    public void onReceive(Context context, Intent intent) {
      //Ce qui doit etre fait lors de la reception de SMS
    }
}
```

Malheureusement ce programme ne fonctionnait pas sur ma carte SIM, mais fonctionnait sur une autre carte. Ce problème fut très long à résoudre, car aucune erreur n'était présente sur Android Monitor, et tous les codes trouvés sur internet étaient similaires au mien. C'est un peu par chance que j'ai réussi à résoudre ce problème, en utilisant une carte SIM présente dans le téléphone prêté par Polytech.

Comme ce code ne peut pas être utilisé par tous les utilisateurs à cause de ce problème, nous avons choisi ne pas utiliser la réception de SMS, en demandant plutôt à l'utilisateur de rentrer les informations nécessaires.

III.2.2. Transmission UDP

Afin d'effectuer un échange de données entre les deux utilisateurs, pour échanger les informations provenant des sextoys mais aussi pour effectuer une vidéo-conférence, nous avons choisi d'utiliser le protocole UDP.

Une première application permettait de tester l'envoi de paquet UDP, pour cela un serveur UDP a été utilisé sur un PC. Le serveur attendait un message et l'affichait quand il le recevait.

Afin d'envoyer un paquet UDP, il faut créer un "DatagramSocket" et un "DatagramPacket" dans une fonction asynchrone. Le "DatagramSocket" prend en paramètre le port d'émission, le "DatagramPacket" prend en paramètre le message, sa taille, l'adresse Ip du destinataire et le Port d'émission, ensuite avec ces deux lignes :

```
ds.setBroadcast(true);
```

```
ds.send(dp);
```
le paquet UDP est envoyé.

Pour recevoir un paquet, il faut comme pour l'envoi créer un "DatagramSocket" et un "DatagramPacket" dans une fonction asynchrone, avec le port de réception comme paramètre pour le Socket et un buffer et sa taille pour le Packet, enfin avec :
```
dsocket.receive(packet);
buff = packet.getData();
```
on reçoit un paquet et on met les données dans une variable buff.

Une fois que tout fonctionnait comme il le fallait, nous avons pu créer une application qui comportait l'envoi et la réception de paquet. En mettant l'application sur deux téléphones on pouvait tester si tout fonctionnait. Les tests ont seulement été effectué sur un réseau local, on ne sait pas si des box internet pourraient empêcher la transmission. L'application est composé d'un thread d'écoute, d'un thread d'envoi et d'un thread principal gérant l'affichage et appelant le thread d'envoi lorsque que le bouton de l'application est appuyé.

Ensuite nous avons cherché à faire du streaming vidéo, un code source fut trouvé mais l'application ne faisait qu'envoyer son flux vidéo par paquet UDP, comme beaucoup de temps avait été perdu pour la réception de SMS et dans la recherche de code source pour le streaming, nous avons préféré nous concentrer vers la conception dans une application "principale".

III.2.3. Application "principale"

Une application regroupant les différentes fonctions crées précédemment devait être créé. Afin d'être plus ergonomique, il fallait créer une base de données regroupant les contacts, à partir de cette liste de contacts leur envoyer des SMS avec des champs réservés pour les informations nécessaire à la communication UDP, supprimer des contacts, ou lancer une communication UDP avec un des contacts.

III.2.3.a Base données

Afin de créer une base de données pour les contacts, nous avons utilisé SQLite car cette bibliothèque est utilisée par Android comme base de données embarquée.

Une classe ContactDAO est créé pour des fonctions d'ajout, de suppression, de modification des valeurs dans la base de données mais aussi différentes chaînes de caractères pour la base de données ,comme le nom de la table et des colonnes.

La classe Contact définit le type Contact afin d'en créer une liste par la suite.

La classe ContactActivity est une activité qui va créer une liste de type Contact à partir de la base de donnée et va grâce à la classe ContactAdapter la transformer en une "ListView" ce qui permettra de cliquer sur les cases d'un Contact et d'en charger la page associé.

La classe dispose aussi d'un bouton d'ajout permettant d'ajouter un Contact dans la base de données.

L'application disposait donc d'une base de données, comportant les noms des contacts et leurs numéros de téléphone. Il fallait donc utiliser ce qui avait été créé précédemment pour pouvoir communiquer avec eux.

III.2.3.b. Création des activités

Pour créer les applications précédentes, une seule activités était nécessaire,mais comme beaucoup d'applications, il est nécessaire de créer plusieurs activités :

- une principale, "MainActivity" permettant d'aller dans les contacts ou dans les réglages, même si finalement il n'y pas réellement besoin de réglages et qui est lancé au démarrage de l'application.

- "ContactActivity" qui affiche une liste de contact et un bouton d'ajout de contact.
- "AjoutContact" pour ajouter un contact et ensuite rejoindre "ContactActivity"
- "FicheContact" qui est rejoint lorsque l'on clique sur un contact, et qui permet de choisir entre supprimer un contact, lui envoyer un SMS, ou commencer une communication UDP.
- "mess" permettant de remplir des "EditText" et ensuite envoyer des SMS du type :
  "Contenu de EditText1
  Adresse: Contenu de EditText2
  Port: Contenu de EditText3"
- "call" permettant de remplir des "EditText" correspondant aux paramètres de la transmission UDP, en cliquant sur le bouton "Envoyer" "DeviceList" se lance.

- "DeviceList" affichant la liste des appareils appairés en cliquant sur "PAIRED DEVICES", en cliquant sur un des appareils on lance l'activité "UDP", si on clique sur le bouton "-" "UDP2" se lance.

- "UDP" essayant de se connecter en bluetooth à l'appareil, si il y arrive l'activité continue, sinon elle s'arrête et "DeviceList" est à nouveau affiché. L'activité dispose d'un "ToggleButton" permettant d'activer ou de désactiver la liaison UDP(envoi et réception), cette activité envoi un paquet toutes les secondes grâce à un Timer.
  Un bouton "Envoyer" est disponible pour forcer un envoie de paquet UDP et de données à l'appareil bluetooth.

- "UDP2" disposant de 4 boutons servant à changer le message UDP envoyé, le message permet de changer le niveau d'excitation du sextoys de l'autre utilisateur. Un "ToggleButton" permet d'activer ou de désactiver la liaison UDP(envoi et réception), comme "UDP" cette activité envoi un paquet toutes les secondes

.

Une fois que l'application principale fonctionnait, il ne restait plus qu'a communiquer avec la partie embarquée

III.3. Transmission bluetooth

III.3.1. Partie embarquée

III.3.1.a. Choix du module bluetooth

Le module bluetooth embarqué sur le PCB est un module reprogrammable. Ainsi il était nécessaire de connaître la configuration par défaut du composant, dans le but de savoir s'il fallait ou non prévoir un pcb de reconfiguration.
Le CYBLE-012012 est configuré par défaut avec le firmware "EZ-Serial BLE Firmware". La configuration par défaut de ce firmware est la suivante :

- 115200 baud, 8bits de donnée, pas de bit de parité, 1bit de stop
- "UART flow control" désactivé
- firmware activé en mode "auto-start", qui correspond à une recherche automatique de connection bluetooth et de redirection RX-TX en cas de connection

.

L'utilisation d'une fonction "auto-start" est en adéquation avec le module bluetooth HC-06 utilisé pour les tests de programmation.

Cependant, lors des essais, il s'est avéré que le CYBLE ne se mettait pas en recherche automatique. Nous avons donc voulu le reprogrammer via minicom en utilisant les mots clés indiqués dans la documentation du firmware "EZ-serial BLE Firmware". Il s'est avéré que les messages à 115200bauds du module bluetooth étaient illisibles. Nous avons donc abandonné l'idée d'utiliser ce module et sommes restés sur l'utilisation du HC-06. Cependant, nous n'en avions qu'un à disposition. Un seul prototype pourra donc communiquer avec son appareil.

III.3.1.b. Communication

La communication s'effectue en liaison série à 9600 bauds. Afin de rendre le programme le plus souple possible, nous avons décidé d'effectuer les lectures par interruption. Nous lisons donc une valeur d'excitation du partenaire que nous passons dans une fonction (lovometre(excitation_partenaire)) qui activera par exemple un ou plusieurs vibreur(s), servomoteur(s) en fonction du niveau.

A chaque réception, le prototype répond par le niveau d'excitation de l'utilisateur.

III.3.2. Partie application mobile

III.3.2.a. Détection du bluetooth

Dans un premier temps il faut détecter si le téléphone dispose du bluetooth.
Pour cela:

```
myBluetooth = BluetoothAdapter.getDefaultAdapter();
if(myBluetooth == null)
{
    //Show a message. that the device has no bluetooth adapter
```

```
}
```

Si le bluetooth est disponible les lignes ci-dessous vont lancer une demande de permission

```
else if(!myBluetooth.isEnabled())
{
        //Ask to the user turn the bluetooth on
                                        Intent      turnBTon      =      new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(turnBTon,1);
}
```

Une fois le bluetooth détecté nous pouvons chercher nos périphériques Bluetooth appairés avec:

```
pairedDevices = myBluetooth.getBondedDevices();
```

A partir de pairedDevices nous allons pouvoir afficher la liste, et en utilisant un Adapter pour pouvoir cliquer sur un des éléments de la liste pour charger débuter la connection avec l'appareil choisi.

III.3.2.b. Transmission

Grâce à un "Socket" et un "Adapter" nous pouvons nous connecter avec l'appareil.
Ensuite nous envoyons une variable byte[] message grâce à:

```
btSocket.getOutputStream().write(message);
```

Et nous recevons dans la variable byte[] mmBuffer les messages avec :

```
btSocket.getInputStream().read(mmBuffer);
```

Cette commande n'étant pas bloquante, il faut essayer de lire au bon moment.

Pour cela, nous avons choisi que la partie embarquée n'enverrait ses messages que lorsqu'elle en reçoit, ce qui permet de facilement se synchroniser.

Il fallait aussi choisir une taille de message similaire, nous avons choisi d'utiliser qu'un seul octet, car cela suffisait amplement pour nos tests.

IV. Difficultés rencontrés

Au cours de ce projet, nous avons rencontré un certain nombre de difficultés:

- La réception de SMS a fait perdre beaucoup de temps, le problème de carte SIM n'était jamais évoqué, personne ne semblait avoir eu ce problème malgré de nombreux codes disponibles. Nous nous sommes trop obstinés dessus, alors que nous pouvions faire sans, cela nous aurait permis d'avancer plus tôt sur le protocole UDP et le reste du projet.

- Android Studio permet d'utiliser de nombreuses API, mais la masse est si importante que l'on ne sait pas par où commencer à chercher. Par exemple la demande de permissions peut prendre un peu de temps à être programmée mais celle pour le bluetooth est très simple et peut être faite en deux lignes. Ainsi en trouvant certains codes un programme peut sembler être difficile, mais avec d'autres codes d'un même programme, tout est beaucoup plus simple. Même les "Sample" fait par Google sont assez complexes, et il est parfois préférable de ne pas chercher à les utiliser.

- La conception de PCB dépend de beaucoup de paramètres, nous avons été optimiste en pensant que la module bluetooth marcherait "comme prévu", et en pensant que l'accéléromètre serait soudable aisément à l'aide du four. Ceci nous a coûté un temps précieux en ré-édition de carte électronique, de tentative de soudure et test après soudure (test IIC de l'accéléromètre, tentatives de reprogrammation du module bluetooth). Nous avons donc appris que pour ce genre de travail, il est important d'avoir une solution de secour, moins risquée.

V. Amélioration possibles

Le projet n'a pas été abouti, cependant nous souhaitons insister sur quelques pistes d'amélioration :

- Il serait intéressant d'utiliser plus qu'un capteur par prototype afin de mesurer l'excitation.
- Mettre à profit les conseils de Mario Sanz sur le silicone afin de créer de réels prototypes, utilisables, coulés dans des moules imprimés en 3D.
- La variable de l'heure n'a pas été utilisée, elle aurait été utile pour la synchronisation UDP
- La communication UDP n'a pas été testée avec des boxs internet, ce test aurait permis de savoir si il fallait utiliser un serveur ou si la synchronisation par sms pouvait suffire.
- Il n'y a pas de repère visuel signifiant la réponse de l'autre utilisateur par paquet UDP, hormis l'excitation du sextoys, la vidéoconférence aurait justement pu être utilisé comme repère visuel, pour montrer l'arrêt (volontaire ou non) de communication avec l'autre utilisateur
- L'application ne prend pas en compte les déconnections des périphériques Bluetooth, il faudrait envoyer un paquet UDP pour signifier que l'on arrête la communication, ou essayer de se connecter à nouveau aux périphériques sans quitter la communication Bluetooth.

Conclusion

L'objectif du projet était de réaliser une paire de sextoys connectés communicants entre eux via une application mobile. Nous n'avons pas réussi à mener à bout le projet cependant, nous pouvons tout de même nous féliciter d'avoir réaliser la majeur partie des fonctionnalités utiles au projet. A savoir communication UDP entre deux téléphones, communication bluetooth bidirectionnelle d'un téléphone à son prototype, acquisition de l'excitation d'un utilisateur et transcription de l'excitation du partenaire sur l'utilisateur.

Afin de mener à bien ce projet, nous avons mis en pratique les connaissances acquises lors de notre formation d'ingénieur, mais aussi d'en développer de nouvelles.

Enfin, nous avons également pu mettre à l'épreuve notre gestion de projet et l'avons amélioré. Nous avons appris de nos erreurs et des points positifs. Ce projet aura donc été une expérience importante pour se projeter vers un projet de fin d'étude en cinquième année.