Rapport de projet de 4ème année

Informatique, Microélectronique, Automatique

Matelas connecté

Élèves : OBEISSART Morgan – ROBIC Vincent

Encadrants : A. Boé – T. Vantroys – N. Rolland

Année scolaire : 2015 – 2016

POLYTECH LILLE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

Sommaire

Introduction

Dans le cadre de notre quatrième année au sein du département Informatique, Microélectronique et Automatique de Polytech Lille, nous avons choisi de nous pencher sur l'un des sujets de projets proposés par des personnes extérieures mais également intérieures à l'école. Ce sujet a pour objectif de développer un démonstrateur de matelas connecté dans le but d'améliorer le confort du sommeil de l'utilisateur.

Nous avons réalisé entièrement le projet en binôme, de l'élaboration du cahier des charges à la réalisation finale du projet, avec l'aide de nos encadrants. Nous souhaitons donc particulièrement remercier Monsieur Boé, Monsieur Vantroys, Monsieur Redon ainsi que toutes les autres personnes qui nous ont apporté leur aide pour mener à bien ce projet.

Dans un premier temps, nous commencerons par présenter plus en détail le projet que nous avons choisi. Puis, nous reviendrons sur le déroulement de notre travail. Enfin, nous ferons un bilan pour donner les améliorations que nous avons apporté au projet et pour exprimer notre ressenti vis-à-vis de ce projet.

I. Présentation du projet

1. Objectif du projet

L'objectif du projet est de développer un démonstrateur de matelas connecté. Ce matelas aura pour but d'améliorer le confort du sommeil de l'utilisateur, chose de plus en plus importante de nos jours, grâce à la possibilité de régler la fermeté du matelas.

Nous envisageons alors deux solutions : soit la fermeté du matelas est choisie globalement, soit elle évolue en fonction du poids du dormeur.

2. Description du projet

En utilisant les nouvelles technologies et des matériaux spéciaux il est possible de contrôler à distance la fermeté du matelas, d'où la désignation de matelas connecté.

La conception de ce nouveau matelas devra prendre en compte les contraintes de fabrication du matelas ainsi que les contraintes d'autonomie de la solution. Il conviendra de :

- Définir le cahier des charges en fonction des spécifications du fabricant de matelas,
- Concevoir et réaliser un prototype,
- Intégrer ce prototype dans un matelas.

Ce projet est mené en collaboration avec une entreprise locale "Matelas no stress" située à Tourcoing. C'est d'ailleurs l'un des raisons pour laquelle nous avons décidé de choisir ce sujet.

3. Choix techniques : matériel et logiciel

Pour la commande du matelas à distance, nous avons décidé d'utiliser une application Android. Pour développer cette application, nous avons utilisé Android Studio, qui est un environnement de développement pour développer des applications Android.

Pour faire le lien entre l'application Android et le matelas, nous utiliserons un module « RFduino » qui intègre sur le même circuit un microcontrôleur et un module Bluetooth 4.0 Low Energy, ou BLE (avec antenne intégrée) qui sera notre standard de communication. L'avantage de ce module est que le

microcontrôleur est compatible avec l'interface de programmation IDE Arduino : c'est donc cette interface que nous utiliserons (suggestion faite par Monsieur Boé).

Enfin, nous mettons ici la liste complète du matériel utilisé, la fonction de chaque composant non abordé dans cette première partie sera détaillé tout au long de ce rapport :

- Module RFduino
- Ruban de 4 Néopixels
- Capteur de température DS18B20 1-Wire
- Elément Peltier
- Diffuseur thermique
- Relais
- Jauges de déformation
- Velostat
- Autres « petits » composants (diodes, résistances, transistors…)

Nous avons donc présenté l'objectif de notre projet et en avons fait une courte description. Nous connaissons désormais les choix techniques que nous avons mis en place. Revenons maintenant sur le déroulement de notre projet plus en détails.

II. Déroulement du projet

Au début de ce projet, nous avions réfléchi à la manière de changer la fermeté du matelas. Nous avions alors pensé à intégrer dans un matelas plusieurs chambres à air reliées à une pompe qui serait commandée (par un Arduino par exemple) afin de pouvoir vider et/ou remplir ces chambres et donc de changer la fermeté du matelas. Puis, nous avons eu une première réunion avec nos encadrants qui nous ont demandé de réaliser un premier prototype basique, assez rapidement, qui permet de contrôler des Néopixels et d'afficher la température récupérée grâce à un capteur, tout ceci à distance en utilisant une application Android qui communique en Bluetooth avec la RFduino. Le but étant de montrer à l'industriel les compétences que nous pouvions mettre en œuvre afin de l'aider à préciser ses besoins.

Dans la suite, nous verrons donc que nous avons réalisé ces deux fonctions. Malheureusement, nous n'avons pas eu de contact avec l'industriel : nous avons donc améliorer ce premier prototype en améliorant ces deux fonctions et en en ajoutant une troisième.

1. Mise en place du Bluetooth

Avant de développer les différentes fonctions, nous devons d'abord mettre en place le standard de communication Bluetooth. Pour cela, comme pour chaque fonction que nous avons développée, nous avons séparé le travail en deux parties : le développement de l'application Android et celui du code de la RFduino.

1.1 Code RFduino

Côté RFduino, il est très simple de mettre en place ce mode de communication. En effet, la RFduino intégrant un module Bluetooth 4.0, il existe des fonctions que l'on peut utiliser avec l'IDE Arduino. Voici les différentes fonctions que nous avons utilisé :

- **RFduinoBLE.begin() :** permet de lancer le module Bluetooth.
- **RFduino_ULPDelay(uint64_t ms) :** cette fonction place le module dans un delay ultra basse puissance pour le temps indiqué.
- **RFduinoBLE_onConnect() :** cette fonction permet d'exécuter un code lorsqu'une connexion a lieu.

- **RFduinoBLE_onDisconnect() :** cette fonction permet d'exécuter un code lorsqu'une connexion a lieu.
- **RFduinoBLE_onReceive(char *data, int len) :** cette fonction permet de récupérer les données envoyées par l'autre appareil et d'exécuter du code lorsqu'une réception a lieu.
- **RFduinoBLE.send(const char *data, int len) :** cette fonction permet d'envoyer des données via le BLE.

Le fonctionnement du module Bluetooth se fait donc par interruption : il attend indéfiniment un évènement représenté par les fonctions ci-dessus.

1.2 Application Android

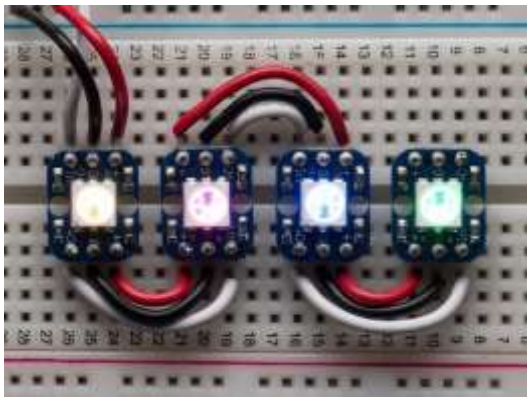
Il est beaucoup plus difficile de mettre en place le Bluetooth du côté de l'application Android. Nous sommes donc parti d'un code déjà existant pour nous aider. Nous avons utilisé l'API BluetoothGatt. Une API (Application Programming Interface) est un ensemble de classes et de méthodes normalisées pour définir un environnement de développement. Cette API fournit donc un environnement qui permet la communication avec des dispositifs possédant le BLE.

Enfin, nous avons intégré les fonctions qui permettent de mettre en place le BLE dans un service. Un service est utilisé pour réaliser des tâches d'arrière-plan de longue durée ce qui correspond exactement à ce que nous voulions faire.

Le protocole de communication étant désormais en place, nous pouvions passer au développement des fonctions.

2. Commande des Néopixels

La première des fonctions est donc la commande des Néopixels via l'application Android. Nous avons utilisé quatre Néopixels RGB v1. Ces LEDS sont individuellement adressables, d'où la nécessité d'utiliser un microcontrôleur.



2.1 Application Android

Comme nous avons pu le dire, nous avons récupéré un code existant pour commencer à développer notre application. Grâce à cela, nous sommes parvenus à développer une application qui permettait de se connecter à la RFduino, de lui envoyer des données, d'en recevoir et de les afficher (ces données étant des chaînes de caractères). Nos Néopixels étant des Néopixels RGB, pour les commander, nous avons choisi au début d'envoyer depuis l'application le code hexadécimal de la couleur que l'on souhaitait afficher. Ainsi, si nous envoyions simplement un code hexadécimal, tous les Néopixels s'allument de cette couleur. Si nous souhaitons changer la couleur d'un seul Néopixel, nous utilisons la syntaxe suivante : **1 : ff0000** (pour allumer le premier Néopixel en rouge par exemple).

2.2 Code RFduino

Comme nous l'avons dit, lorsqu'une donnée est reçue, le code de la fonction RFduinoBLE_onReceive(char *data, int len) est exécutée. Il nous suffit alors de regarde la valeur de la donnée reçue pour commander correctement les Néopixels. Pour rendre cette commande plus facile pour l'utilisateur, nous avons associé trois couleurs avec leurs codes hexadécimaux : le rouge, le vert et le bleu. Ainsi, si l'utilisateur envoie depuis l'application le mot « rouge », alors les Néopixels s'allumeront en rouge, de même pour le vert et le bleu.

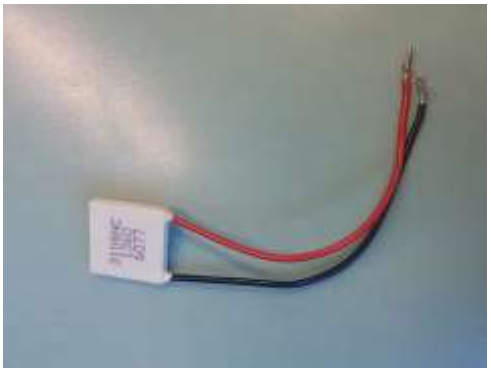
Au début, nous utilisions la bibliothèque Adafruit_NeoPixel qui nous fournissait plusieurs fonctions afin de commander les Néopixels. Cependant, nous nous sommes rendus compte que la couleur que l'on pouvait observer ne correspondait pas tout à fait à la couleur souhaitée : elle avait une luminosité très forte et une forte dominance de blanc. De plus, la différence entre chaque couleur était infime. Nous pensions que cela était dû à une mauvaise alimentation des Néopixels. Après quelques tests, nous nous sommes rendus compte que le problème venait des fonctions utilisées. Nous avons alors supprimé la bibliothèque et créé nos propres fonctions pour résoudre le problème.

Il faut noter que pour changer la couleur d'un Néopixel, il faut d'abord « effacer » l'ancienne couleur pour ne pas avoir une superposition de couleur. Pour cela, nous avons créé les fonctions **void clear()** et **void clear_one(int nombre) :** la première permet d'effacer les couleurs de tous les Néopixels (ce qui revient à les éteindre) alors que la deuxième permet d'effacer la couleur d'un seul Néopixel à la fois.

La fonction de commande des Néopixels à distance était donc opérationnelle. Nous nous sommes donc penchés sur la fonction suivante, à savoir la gestion de la température du matelas.

3. Gestion de la température

Pour la gestion de la température du matelas, nous utilisons un capteur de température DS18B20 1-Wire, un élément Peltier alimenté via un relais ainsi qu'un diffuseur thermique.



3.1. Application Android

Le premier objectif était donc de récupérer la température du matelas. Pour cela, nous utilisons le même principe que pour la commande des Néopixels, à savoir un champ texte dans lequel l'utilisateur peut envoyer une commande sous forme d'une chaîne de caractères pour récupérer la température relevée par le capteur. Pour cela, il faut donc que l'utilisateur envoie le mot « temp ». Dès lors, la température s'affiche en dessous du champ texte, comme on peut le voir sur la photo suivante :

A ce stade, nous parvenions donc à récupérer la valeur de la température. Toutefois, la manière de la récupérer n'est pas très pratique puisque nous utilisions le même champ texte que pour la commande des Néopixels : cela peut ne pas être clair pour n'importe quel utilisateur. Nous verrons par la suite que nous avons amélioré ce point.

Il nous faut maintenant pouvoir gérer la température du matelas. Après quelques recherches, nous avons décidé d'utiliser un élément à effet Peltier. L'effet Peltier (aussi appelé effet thermoélectrique) est un phénomène physique de déplacement de chaleur en présence d'un courant électrique. L'effet se produit dans des matériaux conducteurs de natures différentes liés par des jonctions. L'une des jonctions se refroidit alors légèrement, pendant que l'autre se réchauffe. Néanmoins, notre élément Peltier demande un courant assez fort pour fonctionner de manière efficace : jusqu'à 8,5 ampères. Nous ne pouvions donc évidemment pas l'alimenter avec la RFduino. Nous avons donc utilisé une alimentation externe qui pouvait délivrer jusqu'à 6 ampères. L'utilisation d'un relais associé à un transistor s'est également révélé nécessaire.

Sur l'application, nous utilisions encore le même champ texte pour changer la température. Il fallait pour cela utiliser la syntaxe « consigne=25 » pour mettre la valeur consigne à 25°C par exemple. Là encore, nous avons améliorer ce point, comme nous le verrons par la suite.

La gestion de la température en elle-même se fait donc au niveau de la RFduino.

3.2. Code RFduino

Nous avons utilisé les bibliothèques **OneWire** et **DallasTemperature** qui permettent de commander facilement le capteur de température utilisé. Ainsi, lorsque nous recevons le mot « temp », nous appelons les fonctions **requestTemperatures()** et **getTempCByIndex(0)** qui permettent de récupérer la valeur relevée par le capteur. Nous utilisons ensuite simplement la fonction **RFduinoBLE.send(const char *data, int len)** pour envoyer cette température à l'application.

Pour la gestion de la température, nous avons utilisé une variable globale afin qu'elle soit accessible n'importe où dans le code. Ainsi, lorsque nous recevons un mot correspondant à la syntaxe mentionnée dans la partie précédente, nous affectons la bonne valeur à cette variable globale. Il suffit ensuite de comparer la valeur relevée par le capteur avec la valeur consigne. Pour cela, nous travaillons par interruption : périodiquement, nous relevons la valeur de la température, puis nous la comparons

à la valeur consigne (l'utilisation d'une interruption permet de ne pas utiliser de fonction bloquante dans notre code). Voici la fonction qui nous permet de mettre en place cette interruption :

```
void timer_config(void)
{
  NRF_TIMER1->TASKS_STOP = 1;   // Stop timer
  NRF_TIMER1->MODE = TIMER_MODE_MODE_Timer;  // taken from Nordic dev zone
  NRF_TIMER1->BITMODE = (TIMER_BITMODE_BITMODE_16Bit << TIMER_BITMODE_BITMODE_Pos);
  NRF_TIMER1->PRESCALER = 14;   // SysClk/2^PRESCALER) =  16,000,000/16 = 1us resolution
  NRF_TIMER1->TASKS_CLEAR = 1; // Clear timer
  NRF_TIMER1->CC[0] = 60000; // Cannot exceed 16bits
  NRF_TIMER1->INTENSET = TIMER_INTENSET_COMPARE0_Enabled << TIMER_INTENSET_COMPARE0_Pos;  // taken from Nordic dev zone
  NRF_TIMER1->SHORTS = (TIMER_SHORTS_COMPARE0_CLEAR_Enabled << TIMER_SHORTS_COMPARE0_CLEAR_Pos);
}
```

Si les deux valeurs ne sont pas égales, nous activons le relais grâce au transistor afin d'alimenter l'élément Peltier. Nous avons décidé d'alimenter ce dernier avec 1.6 ampères ce qui est suffisant pour avoir une bonne réaction de l'élément. Pour des raisons de sécurité, nous avons fixé la partie chauffante de l'élément à un diffuseur thermique : nous ne pouvions donc utiliser que la partie froide, mais en réalité, cet élément est réversible. Mais le but était d'être capable d'alimenter un moyen de chauffage/refroidissement tout en sachant que l'élément Peltier ne serait pas forcément la meilleure des solutions : il aurait fallu faire ce choix avec l'industriel.

Nos deux fonctions principales, à savoir la commande des Néopixels et la gestion de la température, étaient donc opérationnelles. Malheureusement, à ce stade, nous n'avions toujours pas de nouvelles de l'industriel et nous ne pouvions donc pas avancer sur le projet. Nos encadrants nous ont alors donné une autre tâche qui consiste à détecter le mouvement d'une personne sur une chaise ou sur un lit.

4. Détection d'un mouvement

Le principal but est d'aider le corps médical à surveiller les mouvements de personnes handicapées qui doivent rester allongés sur un lit ou assis sur une chaise. En effet, certaines de ces personnes ne sont pas capables de se mouvoir elles-mêmes ce qui peut leur causer une nécrose. La nécrose est une forme de dégât cellulaire qui mène à la mort prématurée et non programmée des cellules dans le tissu vivant.

Nous avons décidé de faire nos tests sur une chaise, en sachant qu'il est assez facile d'intégrer le même dispositif dans un matelas. Au début, nous avons utilisé des jauges de déformations. Même si les résultats étaient assez satisfaisants, les jauges que nous avions étaient très petites. Nous avons alors décidé d'utiliser un Velostat disponible à l'école.

Un Velostat est une feuille conductrice sensible à la pression. Lorsque la pression change, sa résistance varie également.

4.1. Code RFduino

Le montage que nous avons utilisé est très simple : il s'agit d'un montage diviseur de tension. Nous utilisons pour cela une résistance de 6.8 KOhms et nous alimentons le circuit sous 5V. Selon la pression, nous avons pu voir que le Velostat a une résistance qui varie de 190Ohms à 540KOhms. Ce qui nous donne une tension en sortie du pont diviseur variant de 130mV à 4.95V. Il nous suffit alors, grâce à la RFduino, de récupérer la valeur de cette tensions grâce à la fonction **analogRead(pin).** Pour détecter un mouvement, nous travaillons également par interruption : périodiquement, nous relevons la valeur de cette tension, puis nous la comparons avec la valeur précédente. Si la différence entre ces deux valeurs est supérieure à 30, alors nous considérons qu'il y a un mouvement. Cette valeur seuil a été déterminé grâce à des tests : elle correspond à un mouvement assez « ample » pour considérer que le patient n'aura pas de problème. Bien sûr, nous pouvons changer ce seuil. Si un tel mouvement est détecté, nous envoyons le mot « mouvement » à l'application.

Si, au contraire, un mouvement n'est pas détecté pendant un certain temps (pour nos tests, quelques secondes mais nous pouvons changer cette durée), alors nous envoyons le mot « alerte » à l'application. Pour détecter cette immobilité, nous utilisons un compteur qui s'incrémente à chaque interruption où il n'y a pas de mouvement. Si le compteur atteint la valeur limite que l'on se fixe selon le temps que l'on souhaite, alors l'alerte est déclenchée.

4.2. Application Android

Nous n'avons pas créé d'activité supplémentaire pour cette fonction. Nous avons décidé d'alerter la personne responsable via une notification. Lorsque le mot alerte est reçu, une notification apparaît sur le smartphone. Voici le code qui correspond à la création de cette notification :

```
if (data1.contains("Alerte") &&  VarGlobale.alerte==0) {
    manager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    //API level 11
    Intent Notifintent = new Intent(MainActivity.this, MainActivity.class);

    PendingIntent pendingIntent = PendingIntent.getActivity(MainActivity.this, 1, Notifintent, 0);

    Notification.Builder builder = new Notification.Builder(MainActivity.this);
    builder.setAutoCancel(true);
    builder.setTicker("Alerte");
    builder.setContentTitle("Alerte");
    builder.setContentText("Il faut bouger la personne dans le fauteuil !");
    builder.setSmallIcon(R.drawable.ic_stat_name);
    builder.setContentIntent(pendingIntent);
    builder.setOngoing(true);
    //builder.setSubText("This is subtext...");   //API level 16
    builder.setShowWhen(true);
    builder.build();

    myNotification = builder.getNotification();
    manager.notify(11, myNotification);
    VarGlobale.alerte=VarGlobale.alerte+1;
}
else if(data1.contains("Mouvement"))
    VarGlobale.alerte=0;
```

Les fonctions intéressantes ici sont :

- **setAutoCancel(true) :** cela permet d'effacer automatiquement la notification après avoir cliqué dessus.
- **setContentIntent(pendingIntent) :** cela permet de revenir à la page contenu dans le pendingIntent lorsque l'on clique sur la notification. Ici par exemple, si la notification a été déclenché sur la page MainActivity, alors lorsque l'utilisateur cliquera sur cette notification, il reviendra sur cette page, peu importe où il se trouve au moment où il clique.
- **setShowWhen(true) :** cela permet d'afficher l'heure à laquelle la notification a été déclenchée.

Si une notification est créée alors que la précédente n'a pas été vue par l'utilisateur, alors il n'y aura pas deux notifications distinctes : il n'y aura qu'une seule notification mais l'heure de déclenchement sera mise à jour.

Nos trois fonctionnalités sont désormais opérationnelles. Mais nous avons pu voir que certaines améliorations pouvaient être apportées. Nous avons réalisé certaines de ces améliorations, comme nous allons le voir dans la partie suivante.

III. Bilan du projet

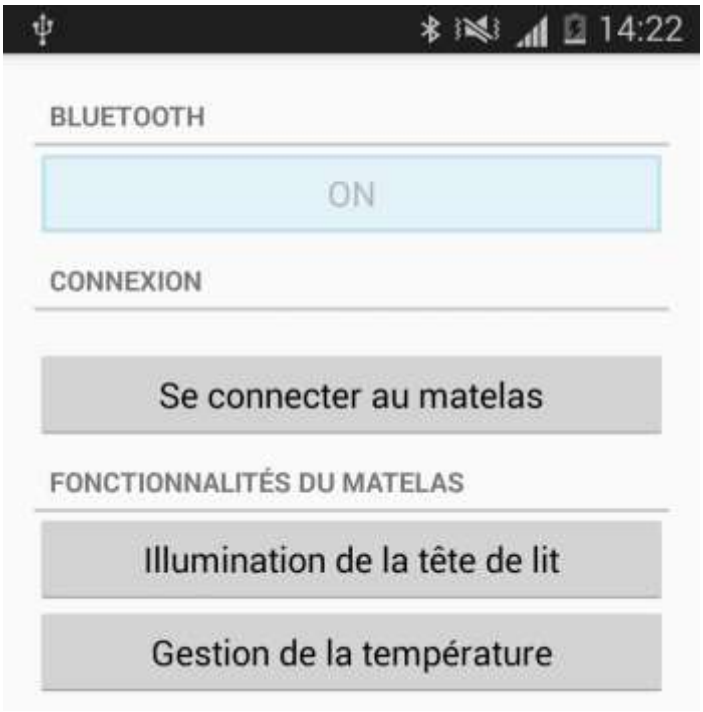
1. Améliorations apportées

A ce stade, notre application ne contient qu'une seule activité (c'est-à-dire une seule vue). C'est donc moins facile d'utilisation : toutes les fonctions se commandent de la même page. Nous avons donc décidé de séparer l'application en trois pages distinctes : la page principale, la page de commande des Néopixels et celle de la gestion de la température. Pour la détection de mouvement, nous n'avons pas créé d'activité à part puisque cela se fait sous forme de notification.

1.1 Page principale

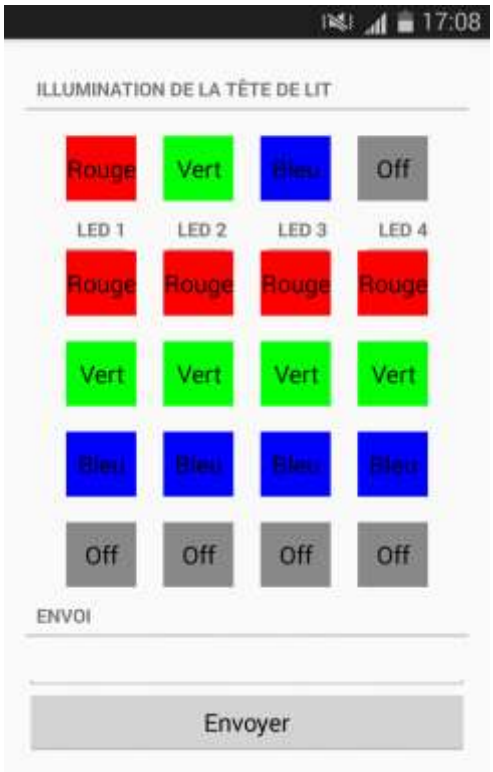
La page principale permet d'allumer le module Bluetooth du smartphone, de se connecter à la RFduino, et d'accéder aux deux pages correspondant à la commande des Néopixels et à la gestion de la température.

Pour la connexion à la RFduino, nous avions auparavant deux boutons : un qui permet de scanner l'environnement pour trouver le bon appareil et un pour la connexion à cet appareil. Nous avons décidé de regrouper ces deux étapes dans un seul bouton pour que ce soit plus facile pour l'utilisateur. Voici donc la page principale de notre application :



1.2 Commande des Néopixels

Nous n'avons pas changé le principe de la commande, notre but ici était simplement de rendre l'application plus facile d'utilisation. Nous avons donc créé plusieurs boutons pour envoyer les couleurs rouge, vert ou bleu car nous avons associé ces couleurs à leurs codes comme nous avons pu le dire ainsi qu'un bouton pour éteindre les Néopixels. La page se sépare donc en trois parties. La première permet d'allumer tous les Néopixels de la même couleur grâce à ces boutons. Même chose pour la deuxième partie, mais pour un Néopixel à la fois. Enfin, nous avons gardé le champ texte en troisième partie pour envoyer n'importe quelle couleur grâce à son code hexadécimal, en respectant la même syntaxe que celle déjà donnée précédemment. Voici la nouvelle page qui permet de commander les Néopixels :



1.3 Gestion de la température

Pour améliorer cette fonctionnalité, nous avons créé une page qui ne comporte plus de champ texte pour envoyer la température consigne. Nous avons remplacé ce champ par une jauge qui représente mieux le fait de vouloir changer la température. Cette jauge est bornée de 10°C à 40°C et est modifiable par pas de 1°C. Lorsque l'utilisateur lâche le curseur de la jauge, la température consigne

correspondante est envoyée. En revanche, nous n'avons pas modifié la manière d'afficher la température. Voici donc la nouvelle page de la gestion de la température :

2. Bilan personnel

Nous avons pu développer toutes les fonctionnalités qui nous ont été demandées au début de notre projet. Toutes ces fonctionnalités sont opérationnelles comme nous le souhaitions. Nous avons également appris à coder sur de nouveaux supports. D'abord sur l'IDE Arduino que nous n'avions pas beaucoup utilisé jusque-là, et encore moins pour coder sur une RFduino. Puis, surtout, nous avons pu coder une application Android quasi entièrement pour la première fois. Avant ce projet, nous ne savions pas du tout comment faire, ce qui fait que nous avons eu beaucoup de difficultés à arriver au résultat obtenu, mais grâce à nos recherches et à nos cours de Java, nous avons pu acquérir beaucoup de connaissances sur le développement d'application Android.

Le point négatif est l'absence total de contacts avec l'industriel, ce qui nous a beaucoup freiné dans notre projet. De plus, nous avions choisi ce projet notamment parce qu'il y avait un lien avec une entreprise : nous voulions faire l'expérience d'un projet le plus concret possible. Malheureusement, cela n'a pas eu lieu, nous n'avons donc pas pu finaliser notre projet initial qui était le contrôle de la fermeté du matelas. Ce qui fait que notre prototype final donne l'impression d'un catalogue de fonctions : le but était de montrer à l'industriel nos compétences afin de le guider dans la définition de ces besoins. Ce dernier aurait pu nous dire ce qui était possible et ce qui ne l'était pas dans un vrai produit.

Néanmoins, le bilan du projet est globalement positif. Il faut noter que notre application fonctionne sur tous les appareils Android ayant la version 4.4.2 ou une version plus récente.

Conclusion

Nous avons donc mis en place plusieurs fonctions qui pourront servir de base pour développer un matelas connecté.

Durant ces douze semaines de projet, nous avons pu acquérir beaucoup de compétences, notamment dans le domaine du développement d'une application Android. Nous avons également pu mettre en œuvre les connaissances acquises durant les deux premières années de notre formation, notamment dans le domaine de l'électronique. Ce projet nous a permis de nous faire une idée de ce qui nous attend dans nos futurs stages et dans notre métier.

Faute de contacts avec l'industriel, nous n'avons pas pu mener à terme le projet initial, mais le cahier des charges fourni par nos encadrants a été respecté et nous sommes satisfaits du travail que nous avons fourni. Beaucoup d'améliorations peuvent évidemment être apportées, notamment l'intégration dans un vrai matelas. Ce projet pourra peut-être être repris pour le mener à son terme.