Rapport de projet
Réception de données
en temps réel

Remerciements

Avant de commencer ce rapport, je tiens à remercier les personnes qui nous ont permis de faire de ce projet une réussite.

- **M. TOUATI et Mme. SEDDIKI pour leurs conseils qui nous ont souvent sortie de l'impasse, leurs cours et la confiance qu'ils nous a accordés.**
- **Toute l'équipe pédagogique pour leur investissement et leurs patiences.**
- **L'université Paris 8 pour nous avoir permis tout au long de l'année de suivre cette formation.**

SOMMAIRE

I. INTRODUCTION ET CONTEXTE

Lors du second semestre qui a duré 4 mois, mon collègue et moi avons travaillé sur un projet.

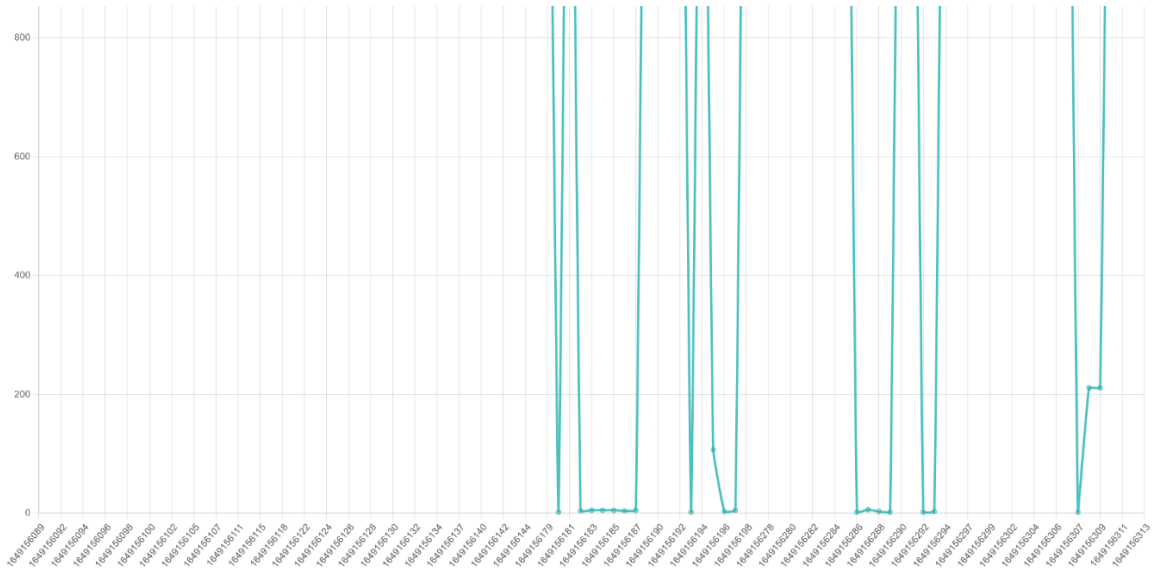
La finalité de celui-ci est de créer une application Android réceptionnant les données d'un capteur en temps réel.

Cette application devra afficher un graphique affichant la dite donnée en fonction du temps.

Pour mener à bien ce projet, nous avons eu accès au matériel suivant :

- Une EasyPic v7
- Un capteur ultrason
- Un capteur de température
- Un module Bluetooth HC-05
- Des fils et 2 câbles USB, un pour l'alimentation, l'autre pour l'UART

Afin de rendre compte le plus rigoureusement possible de notre travail, nous allons présenter plus en profondeur le matériel que nous avons utilisé, ainsi que les tenants et aboutissants du projet.



II. BESOIN ET OBJECTIF DU PROJET

1. Matériel

a. EasyPic v7

Cette carte est une plateforme éducative comprenant plusieurs périphériques, un support permettant de recevoir un microcontrôleur de la gamme PIC. Un ingénieux dispositif de switch permet de réguler l'alimentation des microcontrôleurs.

Cette carte comprend également une suite logicielle avec de nombreuses librairie la rendant programmable. Le langage utilisé à cette fin est le C.

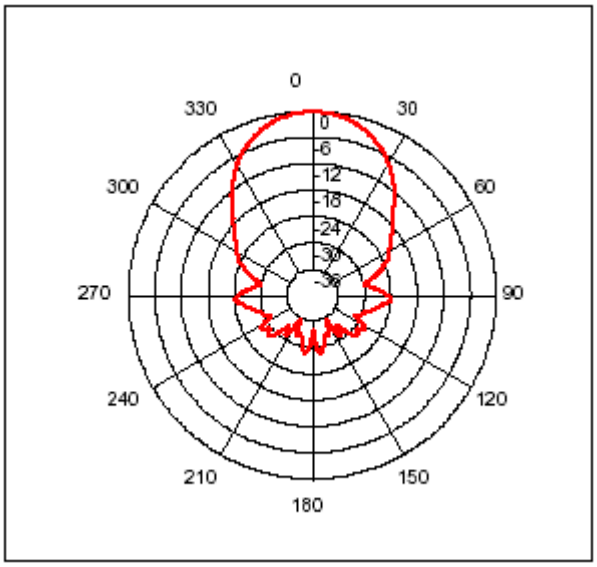
Elle peut également embarquer des dispositifs appelés « click » possédant une/des librairies installé/téléchargeable. N'ayant pas utilisé de « click », nous n'en parlerons pas plus.

b. Capteur ultrason SRF 04

Ce capteur a été fait pour palier à certains problèmes d'autres capteurs standardisés. Parmi ceux-ci, on trouve :

- Le prix
- La consommation
- La distance
- La taille
- Le prix

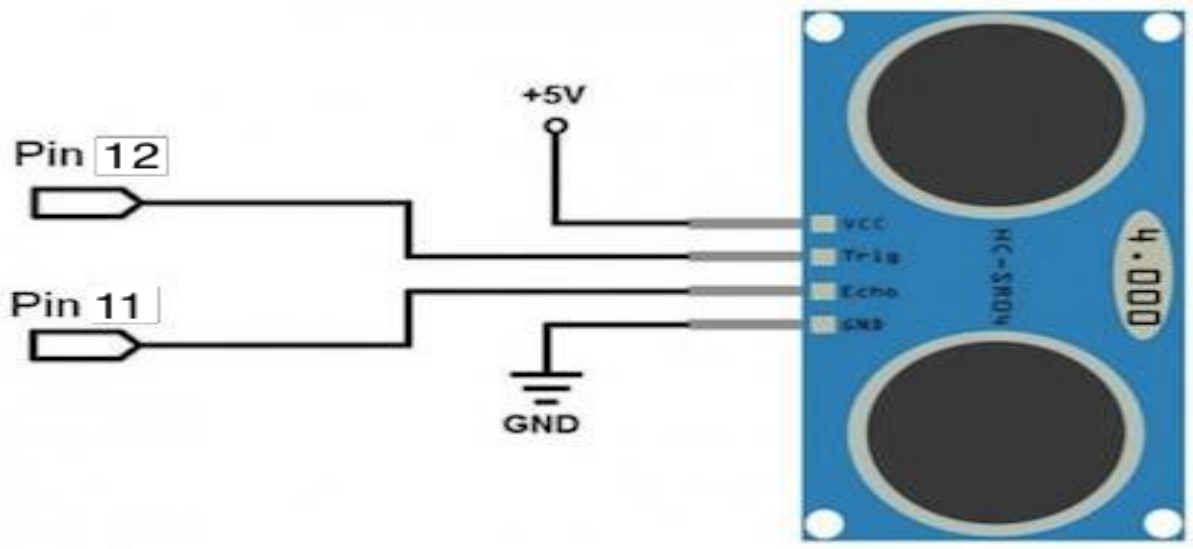
Le principe de fonctionnement du capteur srf04 est entièrement basé sur lavitesse du son. Il peut détecter des objets de 3cm à 3m de distance en suivant un paterne bien précis.



Voilà comment se déroule une prise de mesure :

- On envoie une impulsion HIGH de 10µs sur la broche TRIGGER du capteur.
- Le capteur envoie alors une série de 8 impulsions ultrasoniques à 40KHz(inaudible pour l'être humain, c'est quand plus agréable qu'un biip).
- Les ultrasons se propagent dans l'air jusqu'à toucher un obstacle et retournent dans l'autre sens vers le capteur.
- Le capteur détecte l'écho et clôture la prise de mesure.
- Le signal sur la broche ECHO du capteur reste à HIGH durant les étapes 3 et 4,ce qui permet de mesurer la durée de l'aller-retour des ultrasons et donc de déterminer la distance.

N.B. Il y a toujours un silence de durée fixe après l'émission des ultrasonspour éviter de recevoir prématurément un écho en provenance directement du capteur

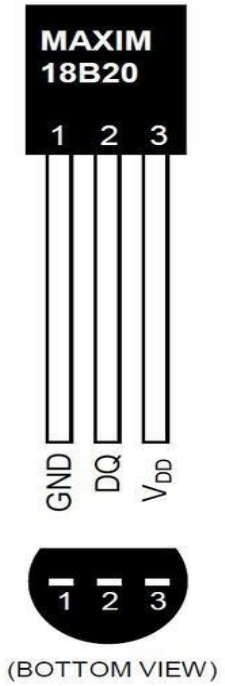


c. Capteur température

Il s'agit d'un capteur de température :

- Qui réalise en interne la conversion analogique numérique
- Peut-être connecté simplement sur 2 fils (technique dite One-Wire)
- Qui réalise une mesure de température sur la plage -55°C à + 125°C sanscomposant supplémentaire (polyvalent donc...)
- Qui renvoie le résultat de la mesure sous forme de données série sur 12 bitsplusieurs capteurs peuvent être utilisés simultanément sur les 2 mêmes fils

Grâce à un système d'adressage de chaque capteur qui possède une adresse unique(64 bits ROM soit 8 octets). À noter que l'octet de poids faible contient le code du type de composant.



.

Ce capteur présente 3 broches :

- Le +5V (Vdd)
- Le 0V ou masse (GND)
- La broche de communication série « One Wire » entrée/sortie (DQ) : Cette broche est un type « drain ouvert » et devra petre connectée au plus par une résistance de 4.7K.

Ce capteur peut être alimenté de 2 façons : On peut l'utiliser de façonclassique avec 3 fils :

- GND au 0V
- Vdd au 5V
- DQ vers une broche numérique du circuit numérique de commande, avec unerésistance de rappel au +

On peut l'utiliser également avec seulement 2 fils grâce à un ingénieux systèmed'alimentation par la broche de données (un condensateur interne se charge et assure l'alimentation du CI lorsque la ligne est à 0V...) :

- GND et Vdd au 0V
- DQ vers une broche numérique du circuit numérique de commande, avec une résistance de rappel au +

Le thermomètre numérique DS18B20 fournit des mesures de température en degrés Celsius avec une valeur codée de 9 à 12 bits. Sa gamme de mesure s'étendde -55°C à +125°C avec une précision de ±0,5°C.

Il dispose d'une fonction d'alarme avec des points de déclenchement haut et bas non-volatiles et programmables par l'utilisateur.

Chaque DS18B20 a un numéro de série unique sur 64 bits, ce qui permet à plusieurs DS18B20 de fonctionner sur le même bus 1-Wire. Il est ainsi simple d'utiliser une EasyPic7 pour contrôler de nombreux DS18B20 répartis sur une grande surface. Parmi les applications pouvant bénéficier de cette fonctionnalité.

On trouve :

- les commandes environnementales HVCA (chauffage – ventilation –climatisation)
- Les systèmes de surveillance de la température à l'intérieur des bâtiments
- la surveillance d'équipement ou de machines
- les systèmes de surveillance et de contrôle de processus industriel.

Le bus 1-wire est basé sur une architecture maître-esclave. Le maître est l'équipement qui contrôle le bus, interroge les périphériques, ou leur envoie des ordres.

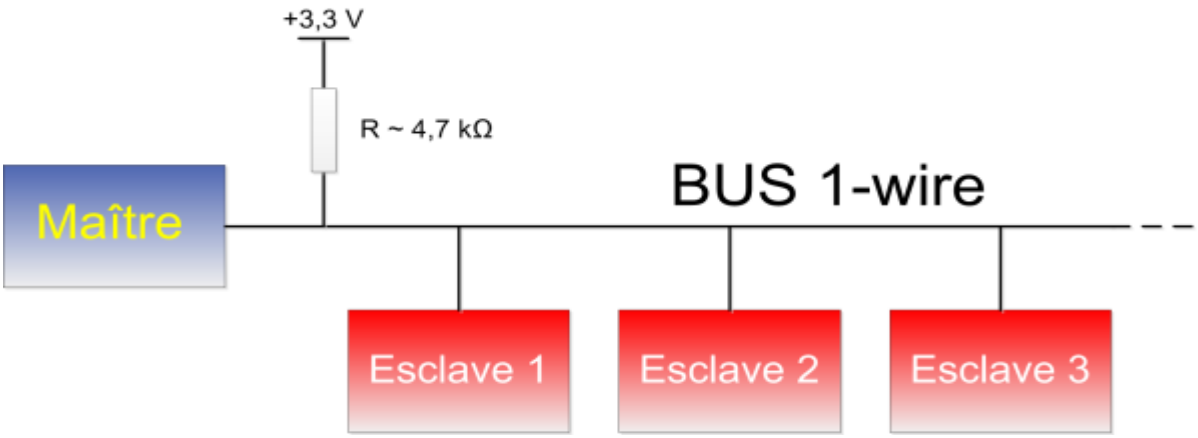
Dans notre cas, le maître sera le PIC16F45K et il n'y aura qu'unesclave, notre capteur de température DS18B20.

Le maître initialise le bus en forçant un zéro pendant plus de 480 µs. Lesesclaves lui répondent en mettent eux aussi leur sortie à zéro pendant un certain temps, indiquant ainsi leur présence.

Le maître déclenche une lecture dite ROM et l'esclave renvoie sonidentifiant unique, gravé lors de la fabrication.

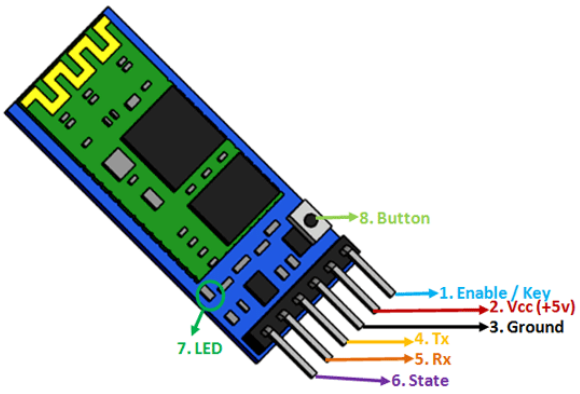
Chaque identifiant a une longueur de 48 bits (8 octets) encadrés par un octet indiquant le type de matériel (ici, le type est : sonde de température =28h) et un octet de CRC qui permet au maître de vérifier qu'il a correctement reçu les informations d'identification du composant.

Les 48 bits permettent d'individualiser à peu près 280 000 milliards decomposants…



d. Module Bluetooth HC-05

Ce module est utilisé pour établir une connexion sans fils entre deux appareil (si les deux le possèdent). Dans notre cas, il est pourvu de plusieurs entrée et sorties.



- Vdd au 5V
- La masse (GND)
- Tx pour écrire
- Rx pour lire
- Enable/Key
- State est une sortie de test

Dans notre cas, nous utilisons seulement Vdd et GND afin de l'allumer, ainsi que Tx pour envoyer du texte. La broche Tx n'est pas branché au hasard. On la branche à la partie qui envoie les données à l'UART sur la carte EasyPic v7, dans notre cas « RC7 ». Une fois branché, il faut maintenant coupler le module à un autre appareil afin d'établir une communication. Il a pour nom « HC-05 » et a pour mot de passe en général « 1234 » ou « 0000 ». Pour ce qui est de la synchronisation, le baud rate est généralement à 38400, cependant, nous l'utiliserons à 9600.

2. Logiciel

Dans cette partie, nous allons détailler les codes que nous avons fait en micro-c pour la EasyPic, mais aussi les code PHP et Android pour l'affichage et l'exploitation des données.

a. UART

Pour l'affichage des données sur le terminal UART de mikroC pro et sur l'afficheur 7-segment de la EasyPic, nous avons procédé ainsi :

- ***Pour le capteur Ultrason :***

  Tout d'abord nous avons utilisé deux fonction d'affichage qui sont « afficheChiffre » et « afficheNombre », la première permet d'afficher un chiffre sur un segment et cela grâce au registre LATDX_bit, qui est le registre de sortie du port D, et on utilise ce port la car c'est lui qui est directement relié au LED de l'afficheur selon une configuration donné , et selon l'interrupteurs qu'on allume ou pas sur ce port D cela affiche un chiffre sur l'afficheur, et la deuxième fonction sert a afficher un nombre sur les quatre digit c'est-à-dire afficher les dizaine, centaine et millième en décomposant le nombre digit par digit et en utilisant des appels à la fonction « afficheChiffre » a des positions différentes et un délai de 1ms entre les différents appels pour qu'on est l'impression qu'il s'affiche tous en même temps.

  Puis on passe à la fonction « main » ou on a indiqué à la carte comment acquérir et envoyer les données à l'uart et nous avons procédé ainsi : dans un premier temps, nous avons branché le capteur sur le Port B (le trigger sur RB1 et l'écho sur RB0) ce qui fait que dans le code nous avons déclaré le port B grâce à l'instruction TRISB (le registre d'E/S du port B), mais surtout, on a initialisé la connexion avec l'uart grâce à la fonction « UART1_Init » qui est fourni par la bibliothèque « UART » de mikroC et on a établi une communication cadencé à 9600 bps.

  Ensuite, nous avons utilisé un « while (1) » pour faire en sorte que le capteur ne s'arrête jamais de capter des données et avoir ainsi du temps réel, et nous avons aussi utilisé un « switch(étape) » pour dire au capteur quoi faire étape par étape.

  Etape 0 : nous demandons au capteur d'envoyer une impulsion HIGH par le trigger grâce à « LATB1_bit =1 » qui envoie cette impulsion au port B1 qui est reliée à la branche trigger, puis on passe à l'étape suivante.

  Etape 1 : Nous arrêtons l'envoie d'impulsion avec « LATB1_bit = 0 », puis on passe à l'étape suivante.

  Etape 2 : on met une condition dans une boucle qui dit que tant que l'echo est activée (en écoute) on incrémente « compte » qui représente le temps et dès que cette condition n'est plus respectée, c'est-à-dire que la branche echo arrête d'écouter car elle a reçu l'écho de l'impulsion alors on sort de la boucle et on passe à l'étape suivante.

  Etape 3 : on calcul la distance parcourus par l'écho grâce au temps et a la vitesse de propagation de ce dernier qu'on connait déjà, et on mets ca dans la variable « **nombreaffiche = (float)compte*10/58.82**», puis on passe a l'affichage sur l'afficheur 7-segments en appelant la fonction « **afficheNombre((int)nombreaffiche)** », et enfin on affiche aussi les résultats dans l'uart grâce à la fonction « **UART1_Write_Text(distChar)** », après que on est converti ses deniers en chaine de caractères que la fonction write puisse écrire « **IntToStr((int)nombreaffiche, distChar)** ».

- ***<u>Pour le capteur de Température :</u>***

  Tout d'abord, on a utilisé les deux mêmes fonctions que pour l'ultrason afin d'afficher les résultats sur l'afficheur 7-segments, mais on plus on utilise la fonction « Display_Temperature » qui prend en argument les données envoyées par le capteur et les converties en valeurs décimal pour pouvoir les afficher sur l'uart avec « UART1_Write_Text () »

  Puis la fonction « main » ou on configure le Prot E comme port digital et on configure RE2 en input car on aura besoin de se dernier pour se connecter au bus 1-wire de notre carte et capter les données du capteur de température qui est branchée sur ce dernier, puis on initialise une connexion avec l'uart cadencé à 9600 bps « UART1_Init(9600)», et enfin on utilise une boucle while(1) pour capté et afficher en continue les données et avoir du temps réel, et dans cette boucle nous initialisant une ligne de communication one-Wire grâce à la commande « Ow_Reset » depuis le port E2, puis on envoie une commande ROM « Ow_Write(&PORTE, 2, 0xCC) » pour parler à tout le BUS. Comme nous n'avons qu'un seul capteur, c'est parfait ! Ensuite, on utilise la commande «Ow_Write(&PORTE, 2, 0x44) » pour demander d'effectuer une conversion "CONVERT_T" c'est-à-dire convertir la donnée en température, après ça on réutilisant la commande « Ow_Reset » car avant chaque communication il faut "initialiser" la ligne, puis on refait une lecture du code 64 bits en ROM (si 1 seul capteur) "SKIP_ROM" avec la commande « Ow_Write(&PORTE, 2, 0xCC) », et on demande la lecture de la mémoire scratchpad de notre capteur avec la commande « Ow_Write(&PORTE, 2, 0xBE) ».

  Enfin, après avoir attendu le temps de conversion suivant la résolution. Celle-ci se trouve dans 2 Octets car elle est composée de 12bit. On retrouvera donc un LSB (Les Signifiant Byte) et un MSB (Most signifiant Byte).

  Grace à une première lecture, on récupère nos 8bits de poids faible. Ensuite, on fait une deuxième lecture pour récupérer la deuxième partie. Comme c'est censé être nos bits de poids fort on effectue un décalage de 8 à gauche (cela revient a multiplié par 256) avec « temp = (Ow_Read (&PORTE, 2) << 8) + temp ».

  Pour finir, on fait un appel à la fonction « Display_Temperature(temp) », pour afficher les résultats du capteur sur l'afficheurs et l'uart.

b. PHP

Dans cette partie, nous allons détailler les codes que nous avons faits pour l'affichage des données en PHP, et pour ce faire nous avons divisé le code sur plusieurs fichiers :

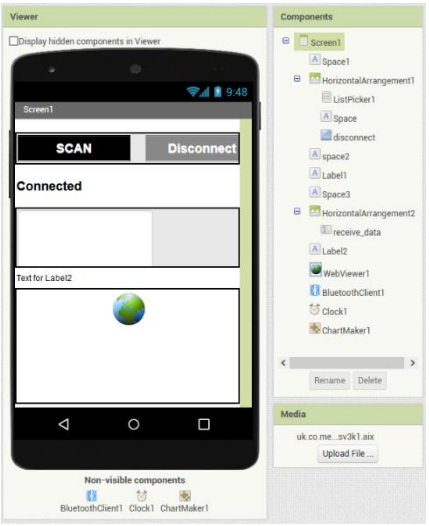
- ***Un fichier SQL :*** ou on a créé notre table SQL « température » qui contient une colonne (username) qui contiendra la donnée captée par le capteur, un timestamp pour afficher l'heure en milliseconde et enfin une colonne (c_date) pour avoir la date, on a aussi créé une procédure « add_ins » pour faire en sorte qu'il n'est pas plus de 100 valeurs qui seront sauvegardé dans la base, on a ensuite créé notre base de données sur MySQL en utilisant ce fichier comme source en utilisant la commande MySQL « source chemin de la src/graphe. SQL ».
- ***Un fichier connection.php :*** ou on a initialisé une connexion avec notre base de données en utilisant l'outil PDO de MySQL et PHP, qu'on a configuré en persistent « PDO :: ATTR_PERSISTENT => true » pour éviter que notre programme se déconnecte de la base de données et se reconnecte ensuite à chaque acquisition de donnée ce qui évite de ralentir et de bloquée l'affichage des données dans le graphe.
- ***Un fichier add_temp.php :*** ou on a établi une connexion avec notre easypic (soit filaire soit avec le Bluetooth), et ceci grâce à la fonction « dio_open » qui permet d'ouvrir le port série auquel les données sont envoyées puis grâce à « dio_read » de pouvoir les lire, nous avons par la suite préparer une requête SQL pour trier les résultats obtenu par ordre croissant par rapport au timestamp et limité l'affichage a 10 données, comme ça on a des données qui s'affiche par ordre chronologique ce qui est meilleurs pour les utiliser afin de dessiner un graph, enfin nous avons utilisé une boucle pour stocker les valeurs obtenu sous forme de tableau « $arr » puis on retourne une chaîne de caractères contenant la représentation JSON de la valeur « $arr ».
- ***Un fichier index.php :*** qui représente notre fichier PHP principale où on appelle le fichier « add_temp.php » pour remplir la BDD avec les données envoyée par le capteur et les afficher sur la page PHP et cela grâce au script ajax qui fait ce traitement de manière dynamique afin d'avoir du vrai temp réel et pour finir on a mis ses données dans un chart « Canvas » afin de les afficher sous forme de graphe dynamique qui les affiches en fonction du temps « timestamps »

c. Android

Afin de compléter le projet, après avoir affiché un graphe sur PHP, il faut afficher ce même graphe, mais sur un téléphone, encore une fois en temps réel.

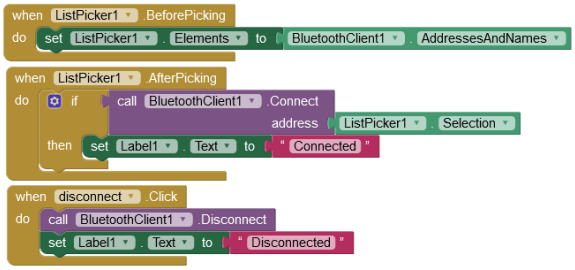
Pour cela, plusieurs options se sont offertes à nous. Utiliser Android Studio et coder une application, qui est l'option qui nous offre le plus de liberté. Où utiliser « MIT App Inventor » qui à contrario nous offre moins de liberté, mais simplifie la création de l'application. Par manque de temps, nous avons préféré cette deuxième option.

App inventor utilise un ingénieux système de block empilé afin d'afficher des modules et de programmer ceux-ci.

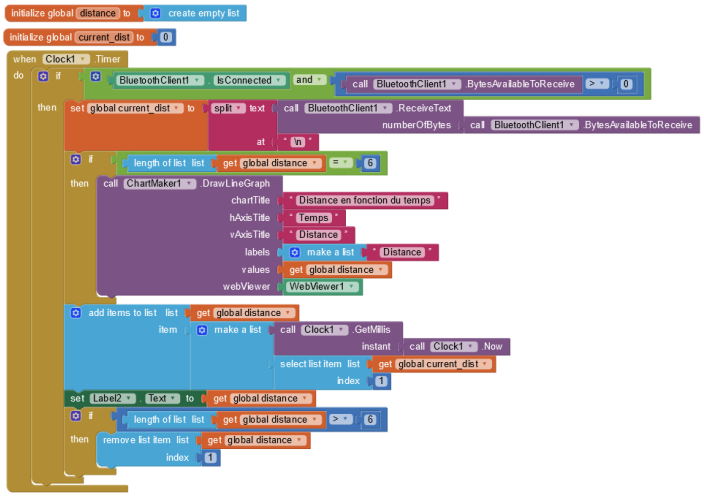


Notre application utilise plusieurs modules. Un pour se connecter au Bluetooth nous permettant de recevoir des messages, un autre affichant les données du capteur et pour finir un « WebViewer » permettant d'afficher des figures au format web. Dans notre cas, il contiendra un graphe créé via une extension ajouté dans le projet du nom de « ChartMaker ».

Le code, quant à-lui se décompose en deux gros blocks :



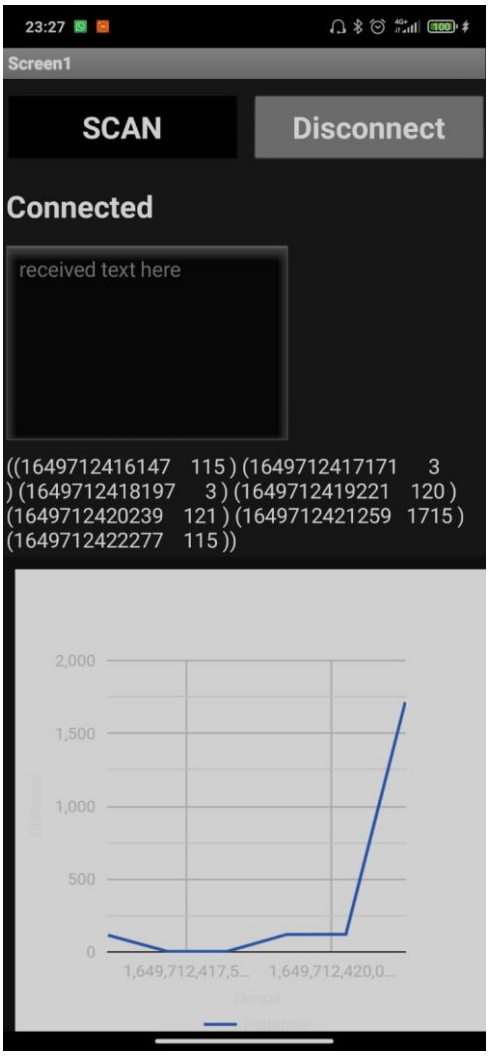
Un premier block propose de se connecter à l'un des modules Bluetooth dans notre entourage. Nous sélectionnerons bien sûr « HC-05 ».



Dans ce deuxième et dernier block une boucle est faite sur l'horloge (boucle infinie donc), chaque seconde, si le module Bluetooth a une donnée qu'il peut recevoir, alors il remplit une liste des valeurs envoyé par le capteur ainsi que la date en milliseconde à l'instant t. Le tableau s'incrémentera en continue, cependant s'il dépasse une taille donnée, la première donnée de celui-ci sera supprimée pour laisser placer à une nouvelle.

À partir de 6 données, un graphe s'affichera. Celui-ci décrire en continue l'afflux de donnée en fonction du temps.

Une fois le programme lancé, il donne un graphe comme ceci :



III. BILAN DU PROJET ET PERSECTIVES D'ÉVOLUTION

Au cours de ce projet, nous avons été amenés à travail avec diverses technologies qui nous étaient jusqu'à lors inconnues. Après de nombreuses tentatives, de nombreux échecs et quelques réussites, nous pensons avoir mené à bien le projet.

En effet, le résultat attendu est bien présent. La carte EasyPic v7 envoie lit les données d'un capteur, le module Bluetooth/UART connecté à la carte envoie les données récupérées vers un autre support, web et/ou Android. Ce dit support quant à lui, récupère les données de manière dynamique, en temps réel et les affiches de manière graphique.

Nous pensons cependant que notre travail peut être amélioré de diverses façon.

La première étant l'affichage des données sur un support web en utilisant le Bluetooth. En effet à cause d'un problème de paramétrage lié à notre module Bluetooth, la liaison est impossible.

Deuxièmement, on devrait pouvoir afficher plusieurs graphiques montrant chacun les données liées à plusieurs capteurs.

Et finalement, notre application Android n'utilise pas de base de données et se contente d'afficher les données en temps réel sans les stocker. Il serait donc intéressant de recoder l'application sur Android Studio afin d'avoir plus de liberté par rapport à ladite application.