Royaume du Maroc

Agence Nationale de Réglementation des Télécommunications

Institut National des Postes et Télécommunications

MINI-PROJET

Commandes Arduino via Bluetooth et GSM, et Collecte de données par module Wifi

Encadré par :

Professeur BOUYAHYAOUI Abdelmalik

Préparé par :

AIT SAID Noureddine EL AOUMI Khaoula

Année universitaire : 2015/2016

SOMMAIRE

RESUME **3**

ABSTRACT **3**

REMERCIEMENTS **4**

I. INTRODUCTION **5**

II. PARTIE DOMOTIQUE : COMMANDES PAR BLUETOOTH **5**

1. OBJECTIFS 5
2. MONTAGE ET PROGRAMMATION 5
3. CORRESPONDANCE DES PINS 6
4. PARAMETRAGE DU MODULE 6
5. TRANSFERE DES COMMANDES 7
6. LA RAISON DU CHOIX DE LA METHODE 8
7. PROGRAMME GLOBAL 8
8. LES PERSPECTIVES DU PROJET 8
9. PRESENTATION DE LA PLATEFORME MIT APP INVENTOR ET REALISATION DE L'APPLICATION ANDROID 9
   - *a. La raison du choix de MIT App Inventor 9*
   - *b. Phase Programmation 10*
   - *c. Fonctionnement de l'application 12*

III. PARTIE DOMOTIQUE: COMMANDES PAR GSM **13**

1. PRESENTATION DU GSM 13
   - *a. Mise en route 13*
   - *b. Montage Module Gsm - Arduino 14*
   - *c. Domaines d'applications 15*
   - *d. Configuration du module GSM 15*
2. REALISATION DE LA PARTIE COMMANDES PAR GSM 16
   - *a. L'objectif 16*
   - *b. Les commandes AT utilisées 16*
   - *c. Les fonctions utilisées 16*
   - *d. Le programme final 16*

IV. PARTIE IOT : COLLECTE DE DONNEES PAR WIFI **17**

1. PRESENTATION DU MODULE 17
2. MONTAGE ET TEST DE FONCTIONNEMENT 17
   - *a. Connecteurs de l'ESP8266 ..... 17*
   - *b. Caractéristiques 18*
   - *c. Test de fonctionnement 18*
3. PLAN A : ENVOI DE COMMANDES PAR WIFI ET CONCEPTION D'UN SERVEUR WEB 19
   - *a. Objectif 19*
   - *b. Problèmes rencontrés 19*
4. PLAN B : UTILISER UNE PLATEFORME CLOUD POUR STOCKAGE DE DONNEES 20
   - *a. La plateforme www.ThingSpeak.com 20*
   - *b. Les requêtes HTTP 21*
   - *c. Utilisation du composant LM35 pour la mesure de la température 22*
   - *d. Automatisation des commandes AT 22*

Pour que la collecte des données soit automatique, il faut mettre le code dans la fonction `loop()`. *Les résultats obtenus :* *22*

V. ANNEXE 23

1. CONNECTION AU MODULE BLUETOOTH PAR LIAISON SERIE NON MATERIELLE (SOFTWARESERIAL) 23
2. PROGRAMME GLOBAL DE LA PARTIE BLUETOOTH 23
3. CONNEXION AU MODULE GSM PAR LIAISON SERIE SOFTWARE 24
4. LE PROGRAMME GLOBAL DU MODULE GSM 24
5. UTILISATION DU COMPOSANT LM35 POUR LA MESURE DE LA TEMPERATURE 27
6. AUTOMATISATION DES COMMANDES AT MODULE WIFI 27

Résumé

Dans le cadre du mini-projet de la première année du cycle ingénieur, on a choisi de concevoir des systèmes domotiques qui ont pour but de contrôler et de commander le fonctionnement de machines domiciles. Ainsi que la collecte des données (température, pression, localisation …etc.). Pour le premier but rien n'est plus pratique qu'une application Android qui communique avec une Arduino par Bluetooth ou par SMS. Et pour la deuxième partie on choisit d'instaurer une plateforme Cloud open source et on utilise un module Wifi pour s'y connecter.

Abstract

For the mini-project of the first year in engineering cycle, we've chosen to design some domotic systems that can help control and manage house machines. We also worked on data collection using open source platforms. For the first purpose we choose to conceive an Android application which controls Arduino via Bluetooth. And for the second one we choose to use a Wi-Fi module.

Remerciements

Avant d'entamer ce rapport, nous profitons de l'occasion pour remercier tout d'abord notre professeur Monsieur BOUYAHYAOUI Abdelmalik qui n'a pas cessé de nous encourager pendant la durée du projet, ainsi pour sa générosité en matière de formation et d'encadrement. Nous le remercions également pour l'aide et les conseils concernant les missions évoquées dans ce rapport, qu'il nous a apporté lors des différents suivis, et la confiance qu'il nous a témoigné.

Nous tenons à remercier nos professeurs de nous avoir incités à travailler en mettant à notre disposition leurs expériences et leurs compétences.

I. Introduction

Dans le cadre du mini-projet de la première année du cycle ingénieur, on a pour objectif de contrôler à distance des machines domiciles, en utilisant plusieurs technologies, à savoir : Bluetooth (et Android), GSM, et WiFi.

Trois objectifs sont visés dans ce mini-projet, et seront classés selon 2 parties :

- **Partie domotique** :
  - Commander des machines à travers un module Bluetooth, via une application programmée sur la plateforme MIT APP INVENTOR.
  - Commander des machines par des SMS envoyés au module GSM.
- **Partie Internet of Things (IoT)** :
  - Collecter des données (température, pression … etc) et les stocker sur une plateforme cloud, notamment ThingSpeak.com.

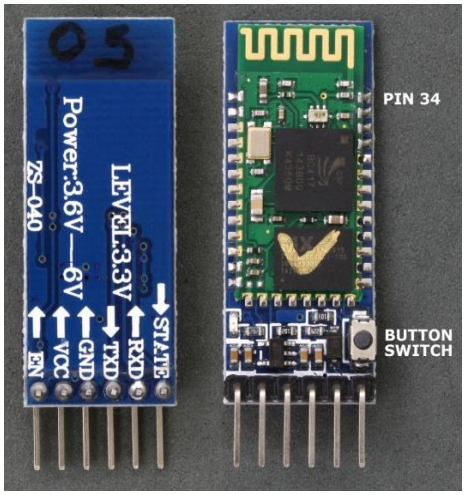
II. Partie domotique : commandes par Bluetooth

1. Objectifs

Notre objectif pour cette partie sera de commander des machines à partir d'une application Android connectée à l'Arduino par Bluetooth.

2. Montage et Programmation

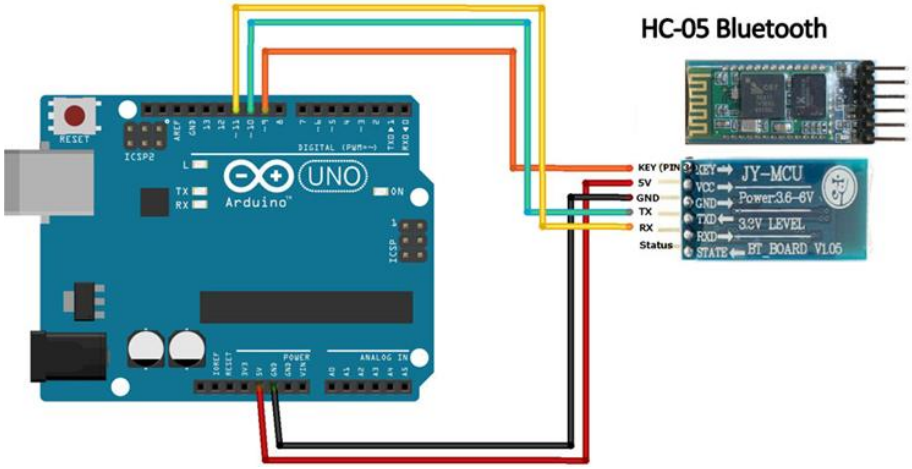
Pour assurer la connexion Bluetooth nous utilisons le module HC-05 ZS-040 suivant :



Comme on veut faire de la configuration sur le module, alors on doit le connecter avec l'Arduino via une liaison serial (Software aux ports 10 et 11).

3. Correspondance des pins

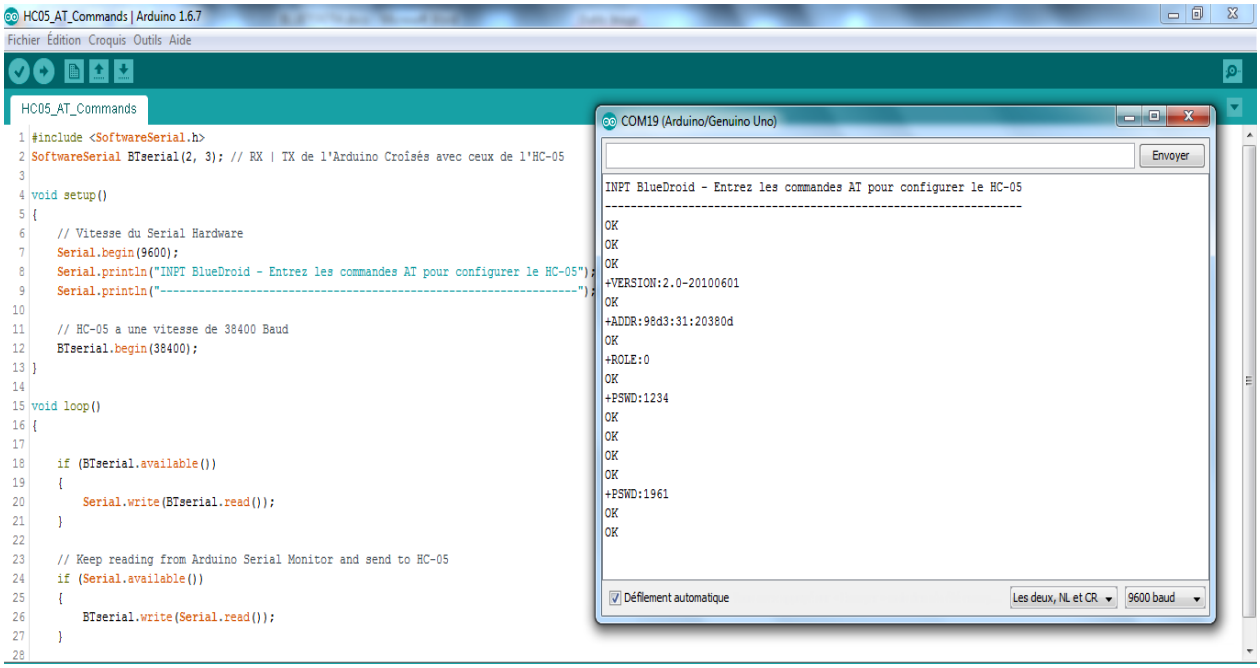
HC-05	Arduino UNO	
Rx	Tx (pin 11)	Liaison série de type Software
Tx	Rx (pin 10)	
GND	GND	
Vcc	5V	
State et En	Non connectés	



Voir programme 1 de l'annexe

4. Paramétrage du module

Comme c'est montré dans l'image si dessous, on a affiché la version, l'adresse qi caractérise le module, On a changé son nom qui est par défaut « HC-05 » à « **INPT BlueDroid** » et son mot de passe qui est par défaut « 1234 » à « **1961** ».

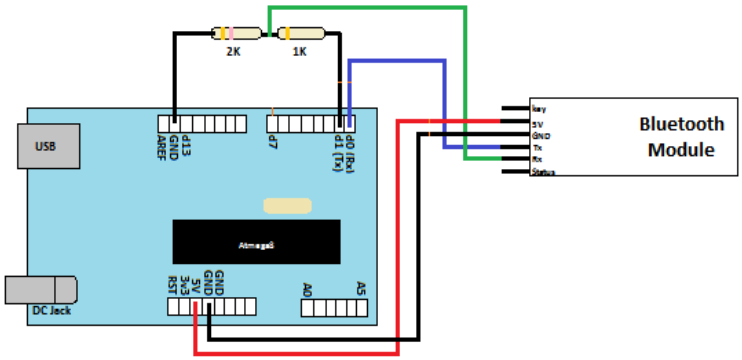


Les commandes AT utilisés :

- AT+NAME=INPT BlueDroid : Permet de redéfinir le nom sous lequel apparait le module sur le Smartphone.
- AT+VERSION? : Donne la version du module.
- AT+ADDR? : Donne l'adresse du module (identifiant).
- AT+PSWD? : Affiche le mot de passe actuel.
- AT+PSWD=1961 : Change le mot de passe en 1961.

5. Transfère des commandes

Après avoir paramétré le module, on le connecte à l'arduino et cette fois directement sur la liaison série Hardware (pins 0 et 1) comme suit :



A chaque fois que l'utilisateur choisit une machines parmi celles listés dans la liste, il clique sur Allumer ou su Eteindre, ainsi on émet un nombre (sur 8bits) parmi ceux du tableau ci-contre.

A chaque pin est associé un nombre qui servira à l'allumage et un autre servira à l'éteindre.

PIN	PIN_ON = Pin+30	PIN_OFF = Pin+60
2	32	62
3	33	63
4	34	64
5	35	65
6	36	66
7	37	67
8	38	68
9	39	69
10	40	70
11	41	71
12	42	72
13	43	73

Exemples :

- Pour allumer le pin 8 on envoie 38 (=8+30)
- Pour éteindre le pin 9 on envoie 69 (=9 + 60)

Supposons par exemple qu'on a choisit d'éteindre la machine liée au pin 10. L'application alors envoie le nombre (10 + 60) = 70 à l'Arduino, après la réception l'Arduino teste si il est entre 32 et 61 alors il s'agit d'un allumage et le

pin concerné sera le nombre moins 30, sinon si le nombre est >= 61 alors il s'agit d'éteindre une machine et le pin concerné sera le nombre émis moins 60.

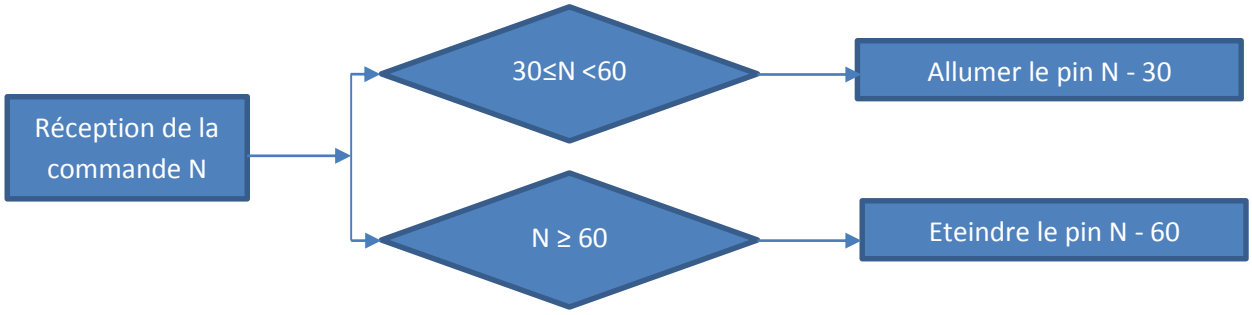
6. La raison du choix de la méthode

On a besoins de transmettre, à chaque fois, deux informations : le pin concerné et l'opération désirée. On a décidé d'associer à chaque pin 2 nombres, un pour allumer et l'autre pour éteindre.

C'est un simple choix du au fait que l'émission des nombres sur 8bits est la méthode la plus simple à faire, mieux que les instructions composées par exemple, et on peut savoir simplement si l'opération désirée est d'allumer (nombre émis entre 32 et 61) et le pin concerné (nombre émis – 30) ou d'éteindre (nombre émis supérieur à 62) et le pin concerné (nombre émis – 60). Les choix de 30 et 60 sont arbitraires. Elle permet aussi une flexibilité au cas où on veut par exemple passer vers une Arduino plus grande (Mega par exemple) et donc de pouvoir étendre le nombre de pins (jusqu'à 30 machines). Et de garder le même code sans changement.

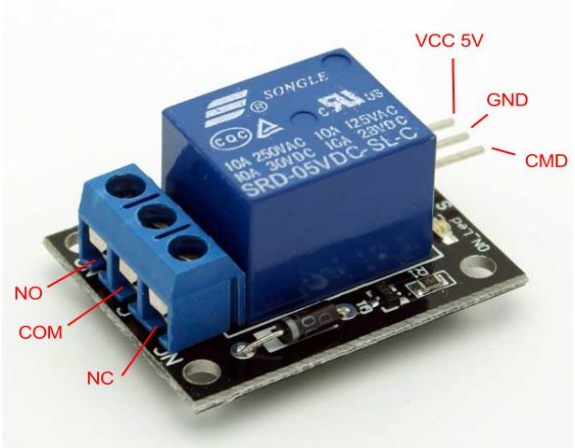
7. Programme global

Voir programme 2 de l'annexe



8. Les perspectives du projet

Tout ce qui reste maintenant c'est de brancher des relais sur les pins entre 2 et 13, auxquels on attachera les machines qu'on veut contrôler. Les relais sont un bon moyen pour controler des équipements qui marchent sur des tensions variables élevées (220V~230V) par des tensions continues très faibles de l'ordre de 5V.



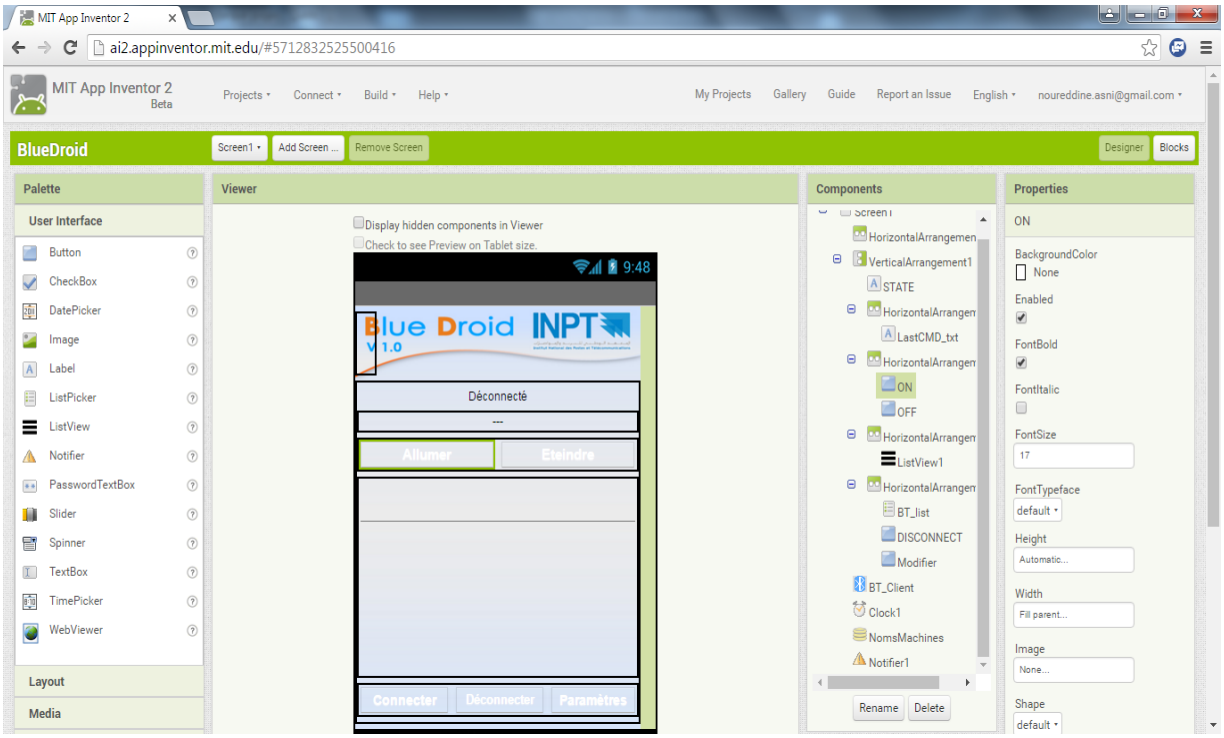
Les relais: Au lieu de mettre les LEDs, on reliera les pins au pin CMD du relais, si le pin est à l'état HIGH (5V), la connexion entre NC et NO est établie. Sinon, s'elle est à l'état LOW (0V) la connexion est coupée, et par conséquent la machine est éteinte.

9. Présentation de la plateforme MIT APP INVENTOR et réalisation de l'application ANDROID

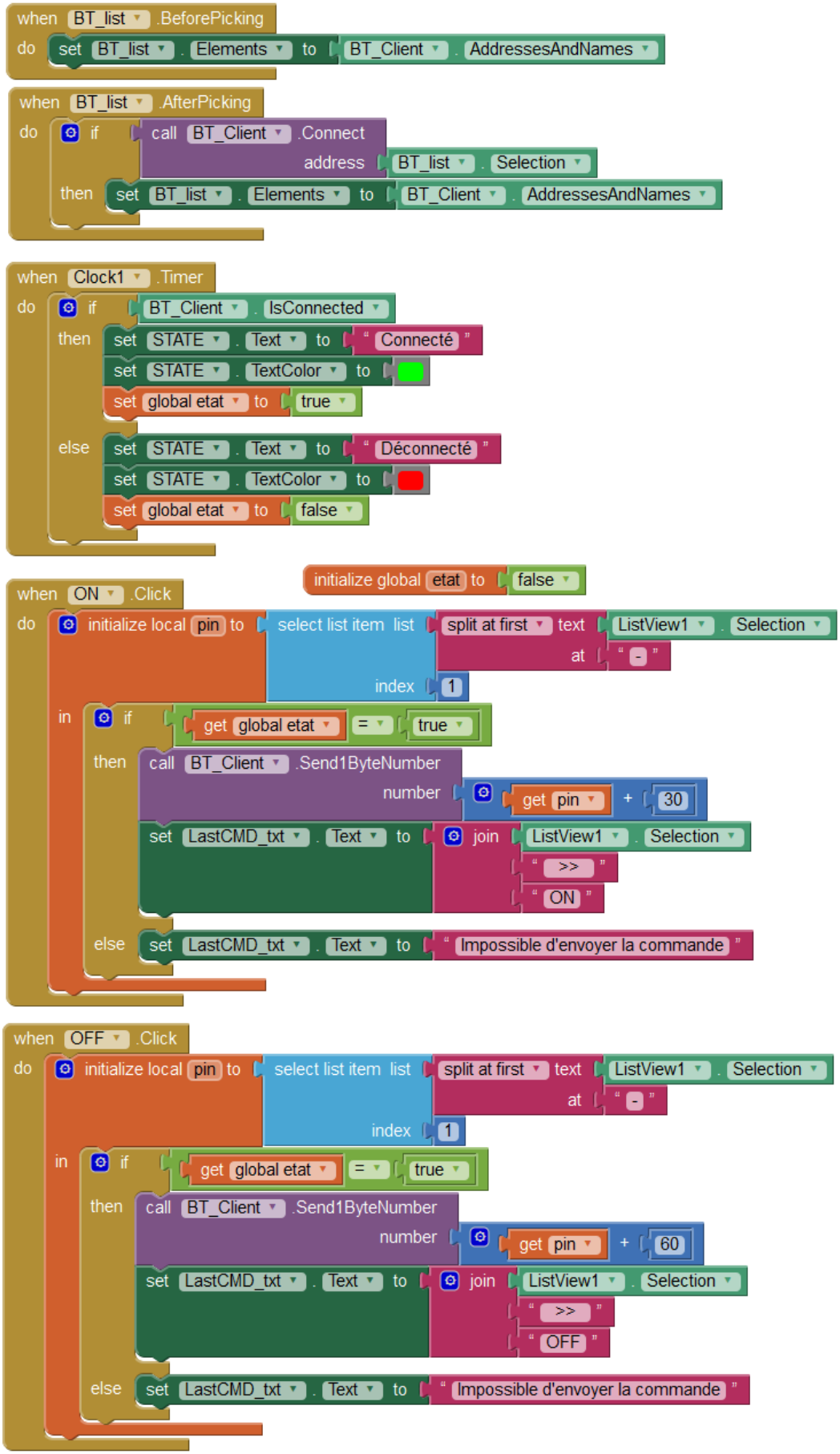
a. La raison du choix de MIT App Inventor

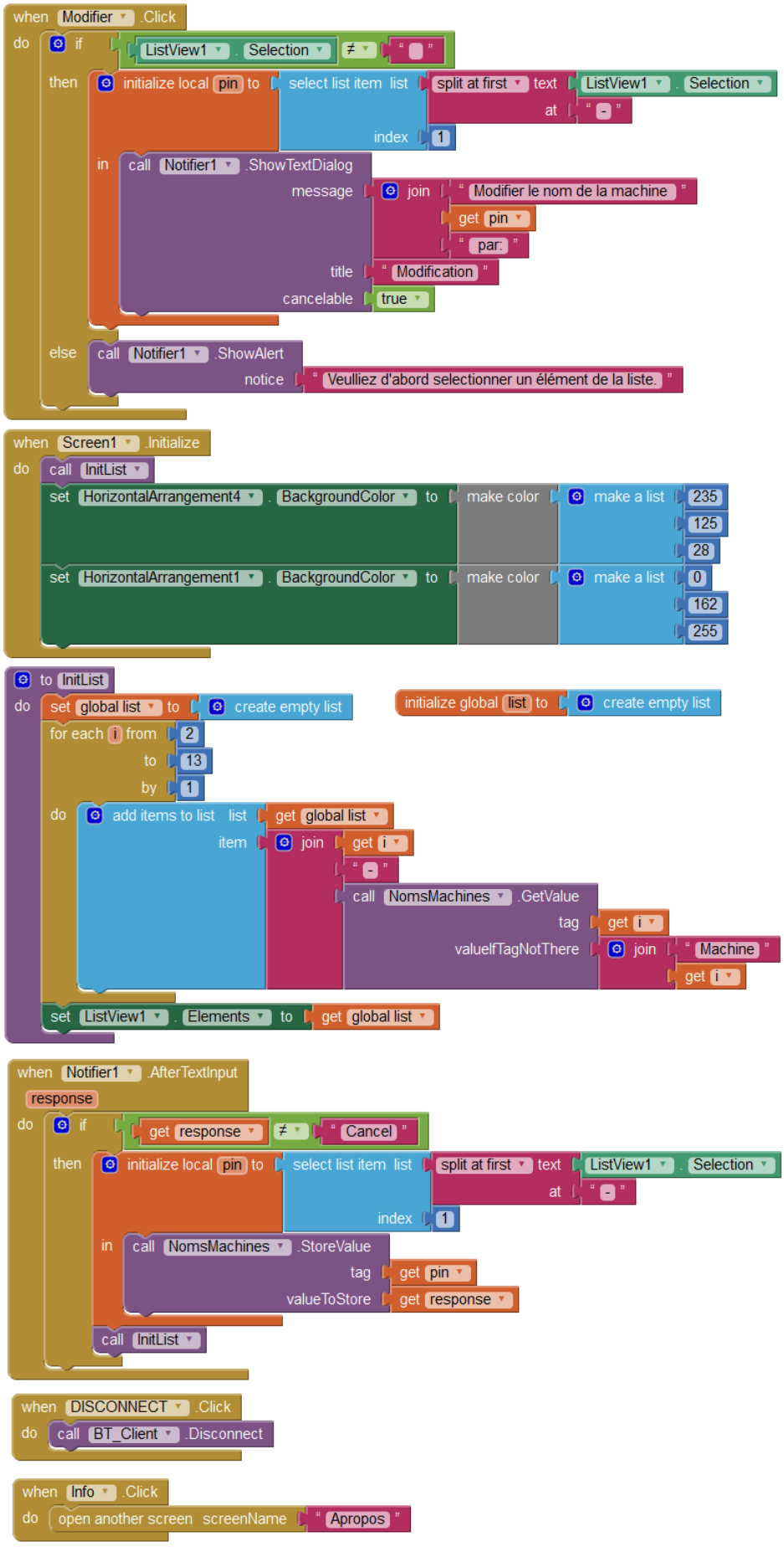
Sur la plateforme MIT App Inventor 2, en se connectant gratuitement par adresse gmail, on peut créer des projets d'applications Android, en combinant des modules (de gestion des périphériques Bluetooth, appels, SMS, gyroscope, appareil photo, enregistreur audio, reconnaissance vocale …), et puis on programme chacun des modules insérés, c'est ainsi qu'on peut créer des application pertinentes par rapport à la méthode classique qui consiste à coder chaque partie par langage Java qui prends beaucoup de temps à mettre en place et pour effectuer les tests.

D'autre part la plateforme offre la possibilité de tester l'application en temps réel, au fur et à mesure qu'on la modifie, soit sur un émulateur installé sur le PC, soit directement sur le Smartphone, ce qui constitue un autre argument pour le choix de cette plateforme.



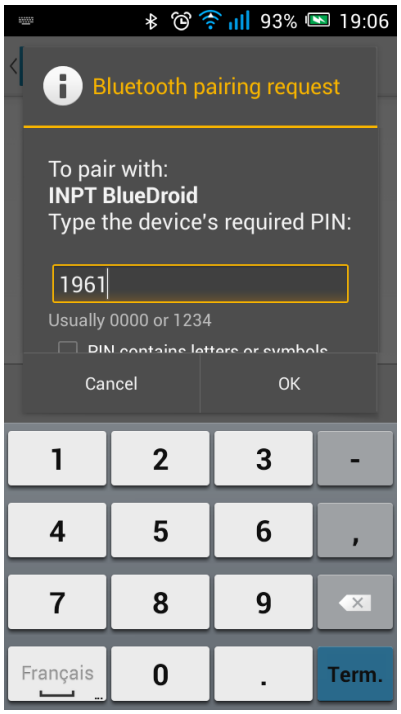
b. Phase Programmation

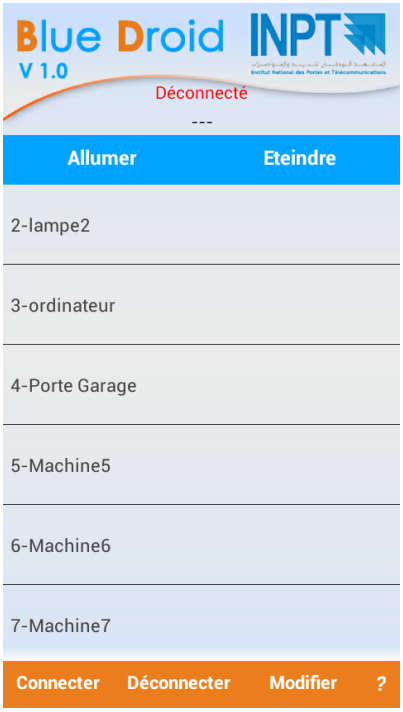


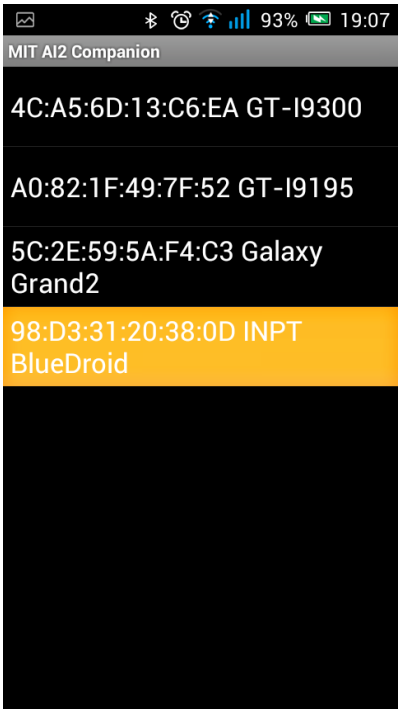

when Modifier .Click
do if ListView1 . Selection ≠ " "
then initialize local pin to select list item list split at first text ListView1 . Selection
at " - "
index 1
in call Notifier1 .ShowTextDialog
message join " Modifier le nom de la machine "
get pin
" par: "
title " Modification "
cancelable true
else call Notifier1 .ShowAlert
notice " Veulliez d'abord selectionner un élément de la liste. "
when Screen1 .Initialize
do call InitList
set HorizontalArrangement4 . BackgroundColor to make color make a list 235 125 28
set HorizontalArrangement1 . BackgroundColor to make color make a list 0 162 255
to InitList
do set global list to create empty list
for each i from 2 to 13 by 1
do add items to list list get global list
item join get i " - "
call NomsMachines .GetValue
tag get i
valueIfTagNotThere join " Machine "
get i
set ListView1 . Elements to get global list
initialize global list to create empty list
when Notifier1 .AfterTextInput
response
do if get response ≠ " Cancel "
then initialize local pin to select list item list split at first text ListView1 . Selection
at " - "
index 1
in call NomsMachines .StoreValue
tag get pin
valueToStore get response
call InitList
when DISCONNECT .Click
do call BT_Client .Disconnect
when Info .Click
do open another screen screenName " Apropos "

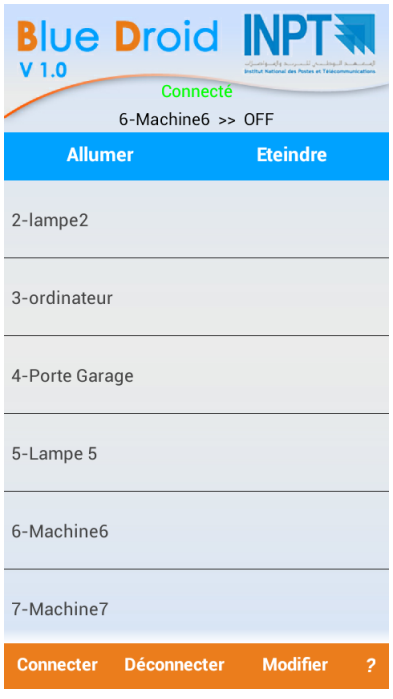
c. Fonctionnement de l'application

- On active le Bluetooth, et on se connecte avec le module qu'on a renommé **INPT BlueDroid** et on entre le nouveau mot de passe qu'on a défini : **1961**.
- On ouvre l'application et on clique sur Connecter
- Une liste contenant les différents équipements Bluetooth auxquels le Smartphone est apparié, on choisit INPT BlueDroid.
- Après la connexion on est prêts à contrôler nos équipement en cliquant sur celui qu'on veut parmi la liste, et on appuyant sur Allumer ou Eteindre. Ainsi l'application envoie la commande à l'Arduino et pourra donc la traiter.









III. Partie domotique: commandes par GSM

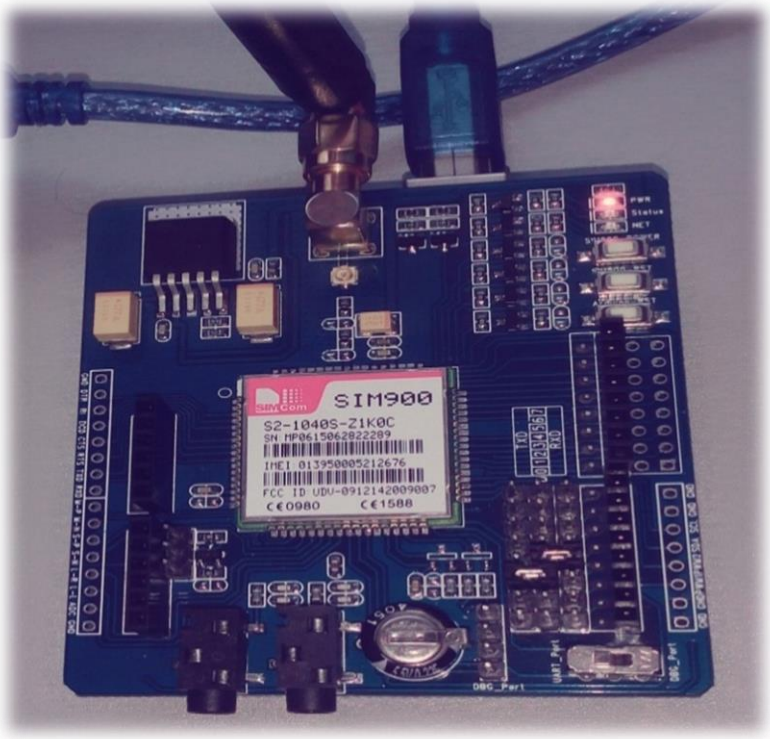
On s'intéresse dans cette partie à contrôler des machines par SMS reçus dans le module GSM. La forme qu'on a choisie pour les messages de commandes est la suivante :

#CMD1**#**CMD2**#** ….. **#**CMD(n-1)**#**CMDn*****

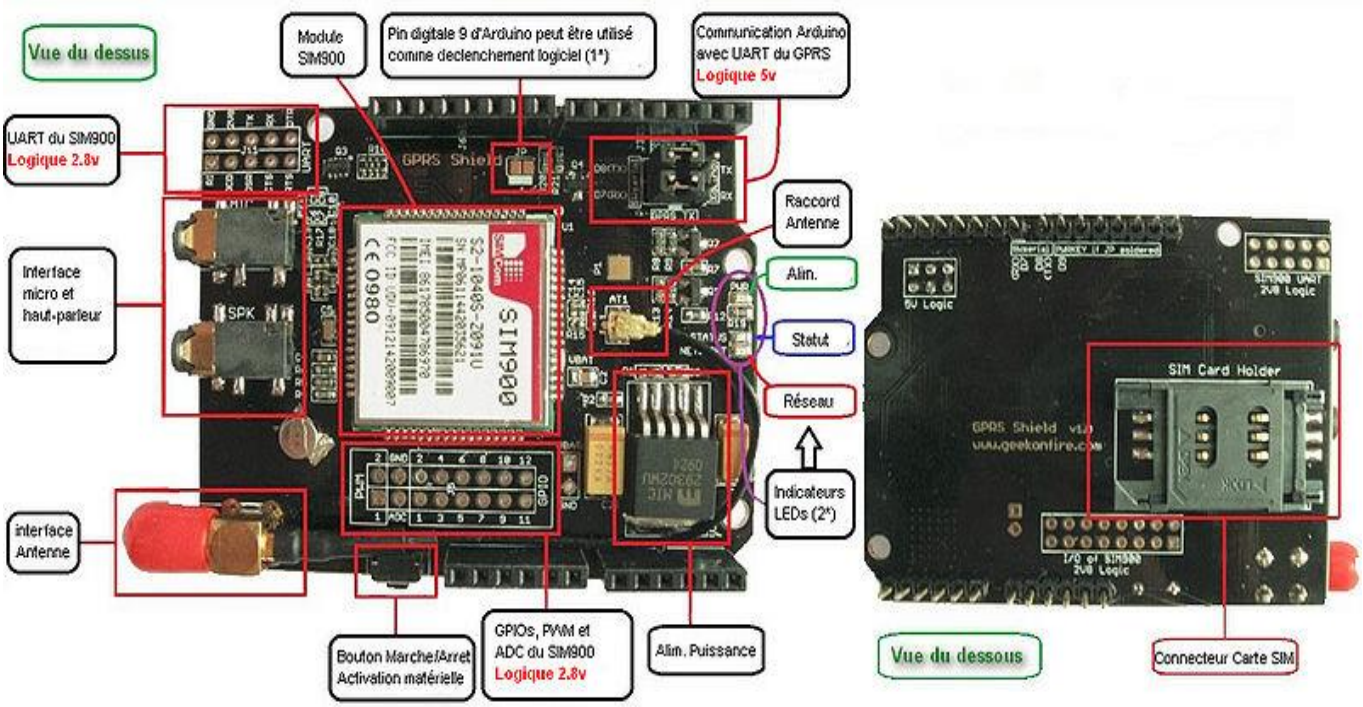
On va essayer de trouver des méthodes pour pouvoir extraire et exécuter de manière correcte les commandes reçus.

1. Présentation du GSM

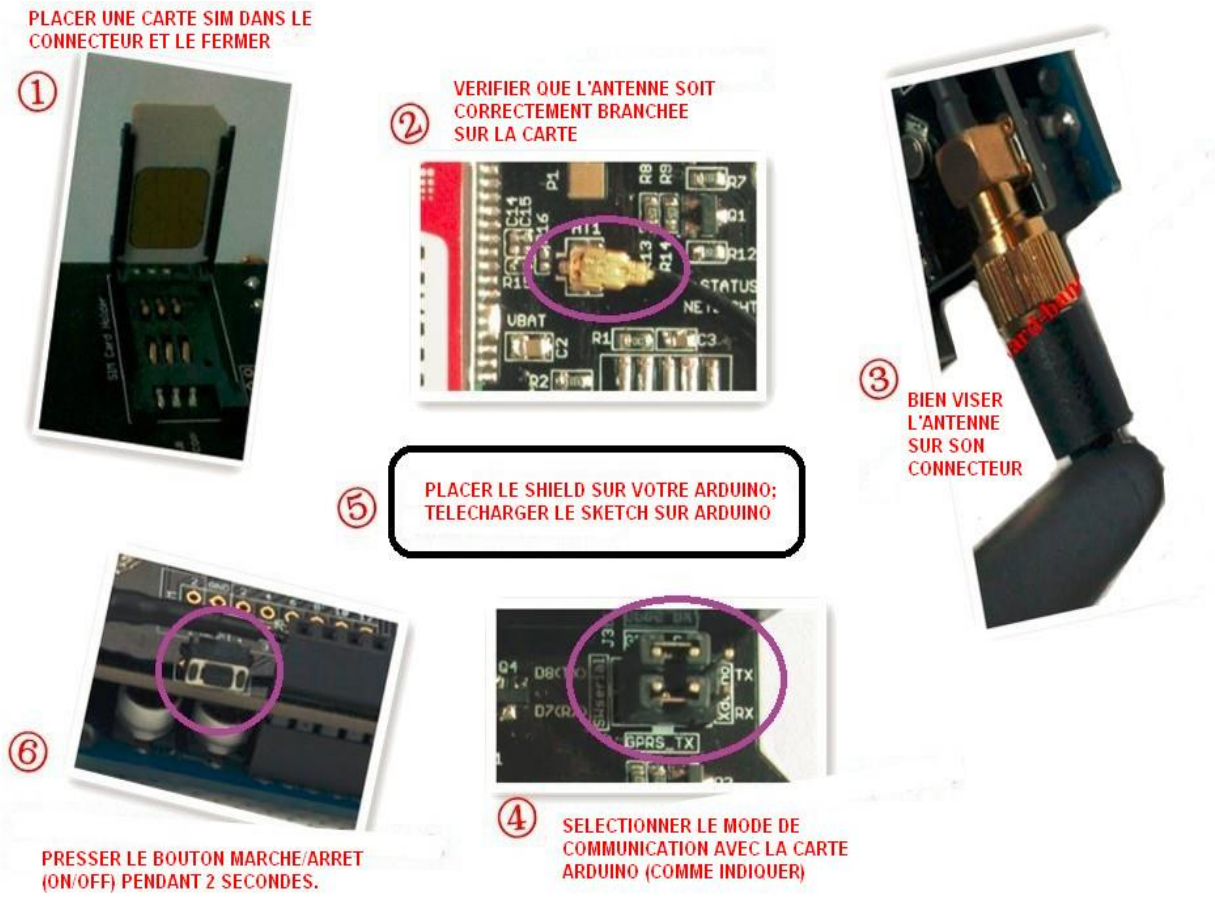
a. Mise en route



- **Le shield GSM/GPRS** est basé sur un module SIM900 de SIMCOM, il est compatible avec **Arduino** et ses clones. Il permet à votre Arduino de communiquer en utilisant le réseau GSM.
- Le shield permet d'envoyer des SMS, MMS et GPRS en envoyant des **commandes AT** à l'UART. Les commandes AT supportées sont GSM 07.07 ,07.05 et SIMCOM enhanced AT Commands.
- Le shield dispose aussi des 12 GPIOs, 2 sorties PWM et du convertisseur Analogique/Digital ADC du module SIM900.
- Voici la description matérielle de ce shield :



b. Montage Module Gsm - Arduino



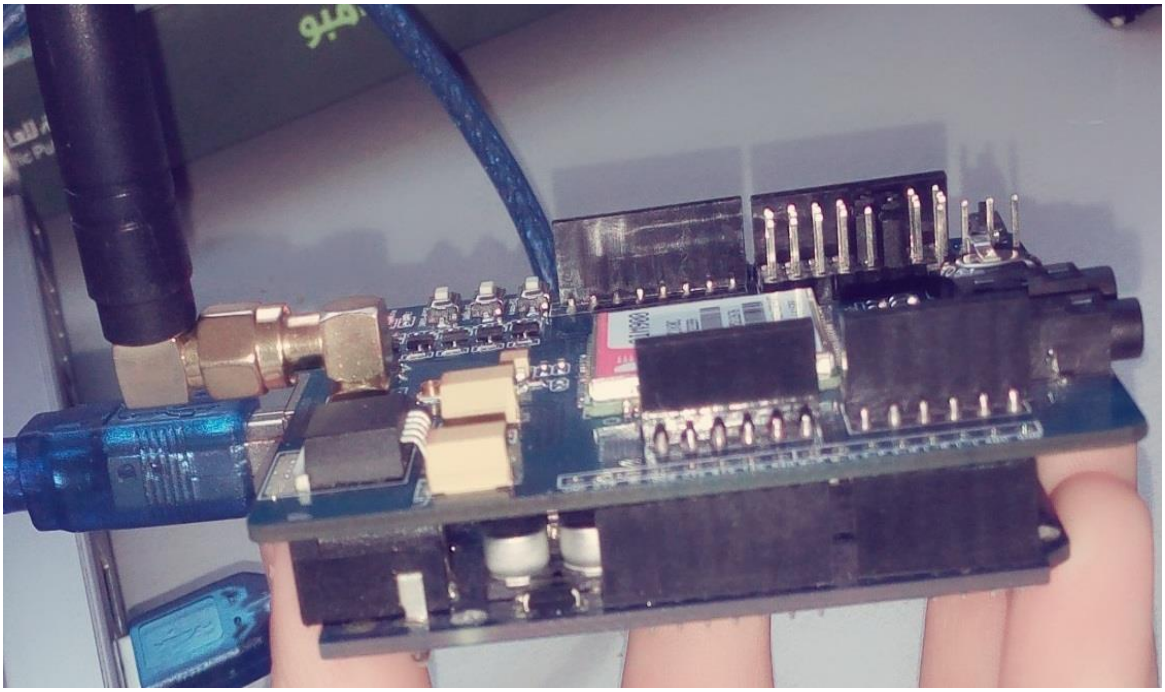
Donc, Le module **GSM** est interfacé avec la carte **Arduino** au moyen d'une simple liaison série (UART). La carte Arduino commande le module GSM en utilisant des **commandes AT** (ou commandes de Hayes) constituent un langage de commande développé afin de commander les modems.

c. Domaines d'applications

Ce montage est utilisé dans des différents domaines comme :

- Dialogue Machine to Machine (M2M).
- Commande d'appareils à distance.
- Réseau de senseurs à distance, station météo distante.
- Système de suivit de véhicule.

d. Configuration du module GSM



L'UART permet d'établir une connexion série avec le module SIM900. A l'aide de cette connexion série, le programme de notre Arduino sera capable d'envoyer des commandes "textuelle" au module (les commandes "AT"). Le module réagira à ces commandes et fournira le service demandé et une réponse sur la liaison série.

Le programme qui assure la connexion :

Voir programme 3 de l'annexe

2. Réalisation de la partie commandes par GSM

a. L'objectif

On s'intéresse dans cette partie à lire des messages qui contiennent n commandes sous la forme suivante : **#cmd1#cmd2# ... #cmdn***, extraire et exécuter ces commandes qui sont transférées de la même manière que dans la partie commandes par Bluetooth :

- ✓ pour **allumer**, on envoie **Pin+30**
- ✓ pour **éteindre**, on envoie **Pin+60**

Attention :

Dans cette partie, on ne doit pas utilisé LES PINS 0, 1, 2, 3 et le Pin 9 qui est attaché à PWRKEY (utilisé pour l'alimentation).

b. Les commandes AT utilisées

AT+CMGL="ALL" : lire tous les messages

AT+CMGDA="DEL ALL" : supprime tous les messages

c. Les fonctions utilisées

1. **GetMessage ()**

Une fonction qui extraire les commandes à partir du message et retourne une chaine de caractères.

2. **extractCMD(String msg)**

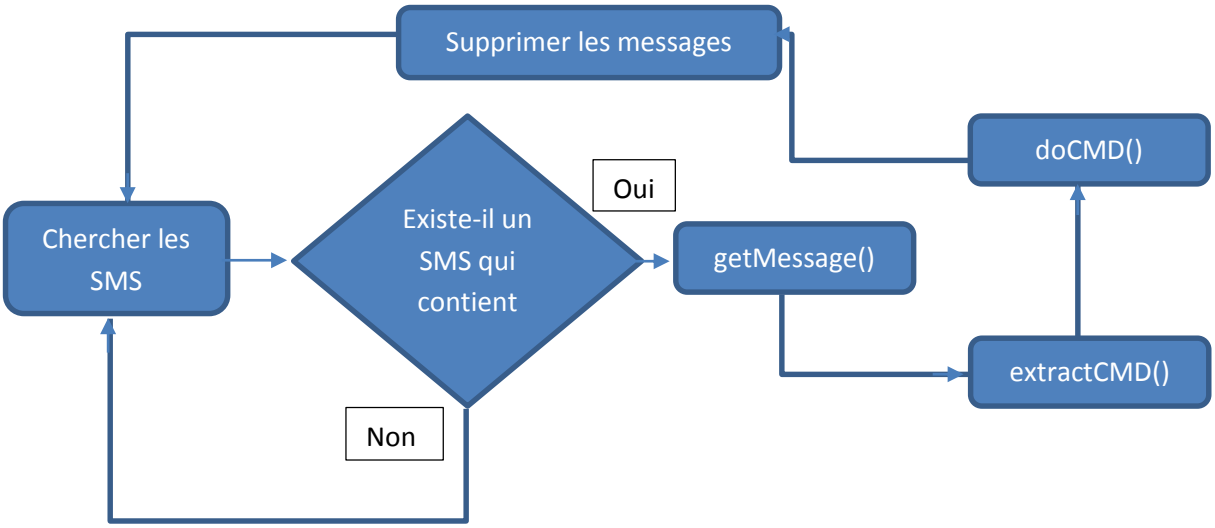
Elle prend comme argument la chaine de caractères retournée par GetMessage () et elle remplie un tableau des commandes qui sera au maximum de taille 13.

3. **doCMD()**

Elle va exécuter les commandes qui se trouvent dans le tableau commandes en trouvant le pin indiqué par l'utilisateur et le mode d'exécution (allumer ou éteindre).

d. Le programme final

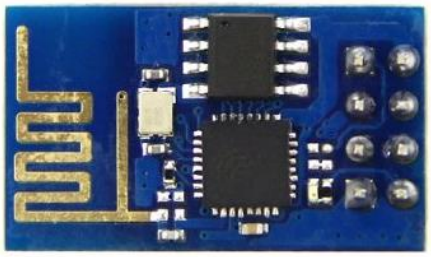
Voir le programme 4 de l'annexe



IV. Partie IoT : Collecte de données par WiFi

1. Présentation du module

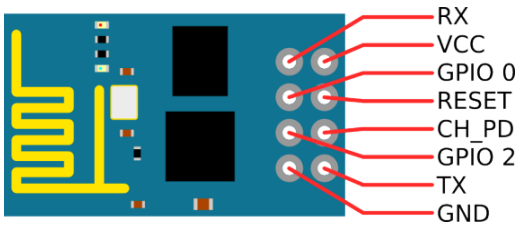
Le modèle ESP-01 du module Wifi ESP8266



L'ESP8266 est un microcontrôleur à cœur Tensilica L106 (RISC 32bit) produit par la société chinoise Espressif Systems. Il existe plusieurs variétés qui diffèrent par le nombre d'entrées sorties GPIO. Le module "ESP-01", c'est la variété qu'on va utiliser, embarque un ESP8266 qui se présente sous la forme d'un circuit imprimé minuscule, équipé d'un connecteur double 4 broches avec une antenne intégrée et une puce de mémoire flash.

2. Montage et test de fonctionnement

a. Connecteurs de l'ESP8266



- Deux broches pour l'alimentation en 3,3V (Vcc et Gnd)
- Deux broches pour la connexion série (Tx et Rx)

- Une broche RST (reset) pour réinitialiser la puce en le connectant à la masse
- Une broche CH_PD (chip power-down) qui doit être alimenté en 3,3V pour activer le Wifi
- Deux broches GPIO pour vos I/O (GPIO0 et GPIO2)

b. Caractéristiques

- Il a des ports GPIO, I2C, ADC, SPI, PWM tout comme un microcontrolleur.
- Horloge de fréquence 80mhz (bien plus que l'Arduino qui a seulement 16mhz)
- 64kbytes mémoire programme de type RAM
- 96kbytes mémoire de données de type RAM
- 64kbytes de démarrage ROM
- Une architecture RISC
- Une pile protocolaire TCP/IP (ipv4)

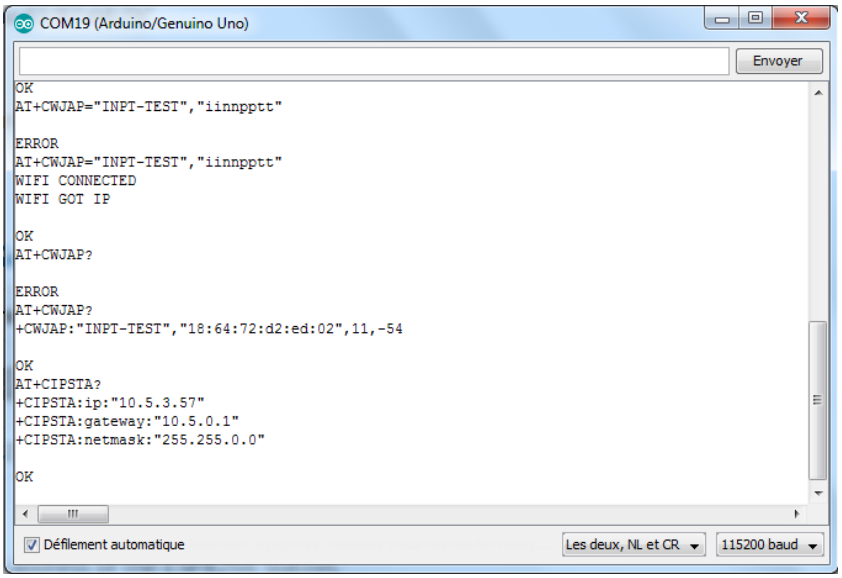
Pour pouvoir programmer son microcontrôleur, on utilise en général un connecteur serial – usb appelé FTDI, on écrit après le programme en Python, et on le transmit vers le module. C'est la méthode la plus utilisée et qui rends le module indépendant de tout autre microcontrôleur externe tel qu'Arduino. L'inconvénient c'est que ce module possède uniquement 2 broches entrée sortie, donc insuffisant pour notre cas.

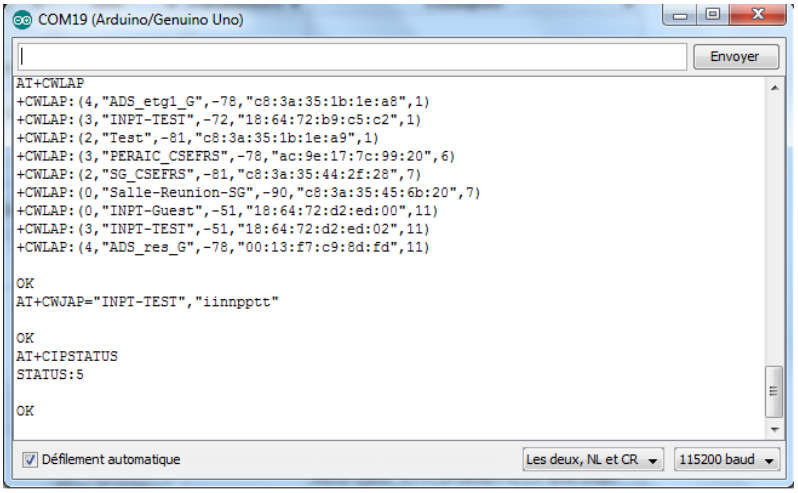
Arduino doit communiquer avec le module via une liaison serial, par des commandes AT.

Les détails de ces commandes sont au lien :
https://room-15.github.io/blog/2015/03/26/esp8266-at-command-reference.

c. Test de fonctionnement

Après avoir relié le module avec Arduino par la liaison série, on peut voir les résultats des commandes sur le moniteur série :





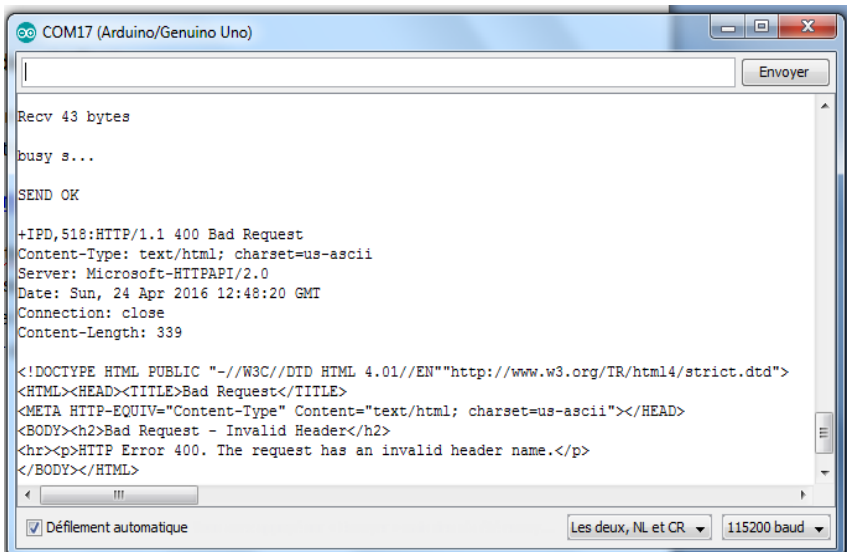
3. Plan A : Envoi de commandes par WIFI et conception d'un serveur WEB

a. Objectif

Notre premier objectif était de concevoir un serveur web, qui servira d'intermédiaire entre l'Arduino et l'application Android. Après des jours de recherche, on a essayé de le programmer en utilisant le langage de programmation web PHP. Ce serveur devait être sous forme d'une page web qui contiendra un fichier texte où on va stocker les commandes, et puis en se connectant sur ce serveur avec ESP8266, en lui envoyant des requêtes http pour extraire ces commandes du fichier texte puis les traiter et les exécuter après.

b. Problèmes rencontrés

Au début tout fonctionne bien quand on fait les tests à partir d'un explorateur internet (Mozilla Firefox par exemple) sur PC. Mais quand on essaye de s'y connecter à partir du module wifi on rencontre un problème : la réponse du serveur est complètement différente genre « Bad request error », parfois » invalid header name » … etc.



D'après les conseils des experts qu'on a contacté, cette solution ne peut pas marcher de cette manière pour plusieurs raisons :

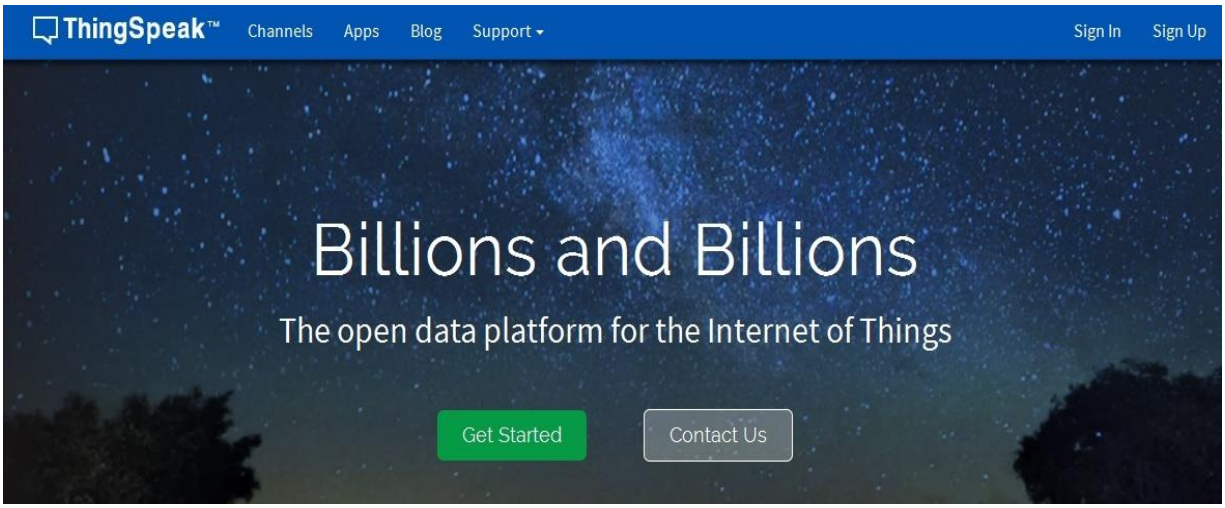
- Le fichier contenant les commandes n'est pas sécurisé et n'importe qui peut y accéder.
- Le serveur web peut tomber en panne puisqu'il n'est hébergé que sur un site d'hébergement gratuit.
- Aucun moyen d'authentification n'est disponible sur le serveur.

Pour remédier à ce problème, on doit penser à utiliser des solutions cloud, or le développement de solutions pareilles depuis le début est un peut compliquée pour des débutants. D'où la nécessité d'opter pour des solutions cloud déjà prêtes et d'essayer de les utiliser comme il faut.

4. Plan B : Utiliser une plateforme Cloud pour stockage de données

Les problèmes qu'on a rencontré nous ont poussé à changer de méthode : Au lieu d'essayer de concevoir un serveur nous même, utiliser des outils prêts. Et au lieu de transférer des commandes, on opte pour la collecte de données sur cette plateforme.

a. La plateforme www.ThingSpeak.com



Selon ses développeurs, "ThingSpeak est une plateforme open source orientée à Internet de Objets (IoT), elle contient des applications et des API pour stocker et récupérer des données à partir des objets connectés en utilisant le protocole HTTP via Internet ou via un réseau local.

ThingSpeak permet la création d'applications d'enregistrement à partir de capteurs, applications de géolocalisation, et un réseau social constitué de l'ensembles des objets connectés avec des mises à jour de statut.

ThingSpeak a intégré le soutien de MATLAB logiciel de calcul de MathWorks. Il permet aux utilisateurs d'analyser et de visualiser les données téléchargées à l'aide de Matlab sans nécessiter l'achat d'une licence Matlab de Mathworks.

Plus de détails : https://www.mathworks.com/help/thingspeak/

b. Les requêtes HTTP

L'***HyperText Transfer Protocol***, plus connu sous l'abréviation **HTTP** — littéralement « protocole de transfert hypertexte » — est un protocole de communication client-serveur développé pour le World Wide Web.

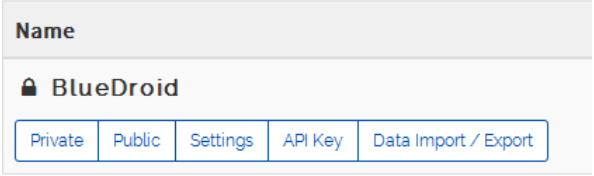
HTTP est alors une sorte de langage qui permet à un client (nous à travers un navigateur) de communiquer avec un serveur connecté au réseau (le serveur HTTP installé sur le serveur d'un site, par exemple Apache).

Il existe plusieurs méthodes :

- **GET**
  C'est la méthode la plus courante pour demander une ressource. Une requête **GET** est sans effet sur la ressource, il doit être possible de répéter la requête sans effet.

- **HEAD**
  Cette méthode ne demande que des informations sur la ressource, sans demander la ressource elle-même.
- **POST**
  Cette méthode doit être utilisée lorsqu'une requête modifie la ressource.
- **OPTIONS**
  Cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.
- **CONNECT**
  Cette méthode permet d'utiliser un proxy comme un tunnel de communication.
- **TRACE**
  Cette méthode demande au serveur de retourner ce qu'il a reçu, dans le but de tester et d'effectuer un diagnostic sur la connexion.
- **PUT**
  Cette méthode permet d'ajouter une ressource sur le serveur.
- **DELETE**
  Cette méthode permet de supprimer une ressource du serveur.

La plateforme ThingSpeak.com, utilise quelques unes parmi ces requêtes pour pouvoir faire une mis à jour (update) des données. En effet sur ce site on peut créer des channels (dans notre cas appelé BlueDroid).



Chaque Channel, contient 8 zones de données appelées (Fields) plus une latitude et une longitude (pour la géolocalisation), et un status.

Exemple :

Pour mettre à jour la zone 1 (field1) avec une valeur **X** et le field2 par une valeur **Y** on doit envoyer la requêtes suivante :

```
GET
https://api.thingspeak.com/update?api_key=90AQPXEAO4B70153&field1=X&field2=Y
```

***api_key:** c'est l'identifiant de la chaine sur laquelle on travaille, on le retrouve on cherchant dans les paramètres de la chaine.

***Field1/field2:** Champs d'informations qu'on va actualiser chaque fois. Chaque champs peut représenter un type d'informations par exemple : field1 pour la température, field2 pour la pression …etc.

c. Utilisation du composant LM35 pour la mesure de la température

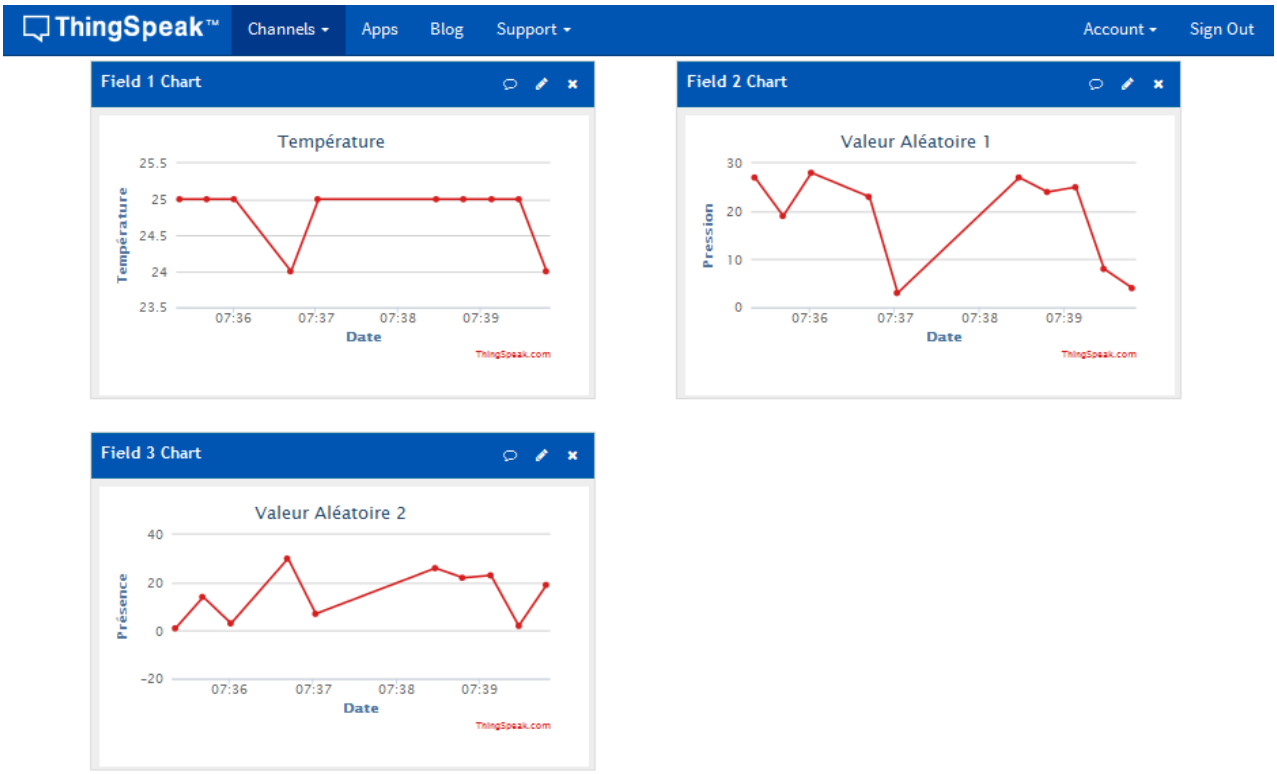
En liant le pin Analog Out avec le pin A0 des entrées analogiques on peut mesurer une valeur entre 0 et 1023, qu'on doit mapper entre 0 et 5V. O n convertit ainsi cette valeur en millivolts. Sachant que chaque 10mV représente 1°C, on peut donc mesurer la température.

Exemple : Mesure et affichage sur le moniteur série

Voir programme 5 de l'annexe

d. Automatisation des commandes AT

Pour que la collecte des données soit automatique, il faut mettre le code dans la fonction `loop()`. Les résultats obtenus :



V. Annexe

1. Connection au module Bluetooth par liaison série non matérielle (SoftwareSerial)

```
#include <SoftwareSerial.h>
SoftwareSerial BTserial(2, 3); // RX et TX de l'Arduino Croîsés avec ceux de l'HC-05
void setup(){
  // Vitesse du Serial Hardware
  Serial.begin(9600);
  Serial.println("INPT BlueDroid - Entrez les commandes AT pour configurer le HC-05");
  Serial.println("-----------------------------------------------------------------");
  // Vitesse du Serial Software HC-05 a une vitesse de 38400 Baud
  BTserial.begin(38400);
}
void loop(){
//Si on reçoit du Serial Hardwaire on transmit au Serial Software et vice versa
// ce qui est reçu par l'un est transmis par l'autre
  if (BTserial.available()){
    Serial.write(BTserial.read());
  }
  if (Serial.available()) {
    BTserial.write(Serial.read());
  }
}
```

2. Programme global de la partie Bluetooth

```
int i = 0;
void setup() {
  Serial.begin(38400);
    for(i = 2; i <= 13; i++){
       pinMode(i, OUTPUT); // définition des pins 2 → 13 comme sortie
    }
}
void loop() {
  while ( Serial.available() > 0)
  {
    byte c = Serial.read(); // L'octet reçu
    if (c >= 32 && c <= 61){
      // Il s'agit d'allumage
      int pin = c – 30 ;
      digitalWrite(pin, HIGH); // Allumer le pin concerné
      Serial.print(pin ) ;
      Serial.println(" >> ON"); // on affiche l'opération sur la fenetre des commandes
    }
    else if (c >= 62) {
      // Il s'agit d'eteindre le pin (c - 60)
      int pin = c – 60 ;
      digitalWrite(pin, LOW);
      Serial.print(pin);
      Serial.println(" >> OFF");
    }
  }
}
```

3. Connexion au module GSM par liaison série software

```
#include <SoftwareSerial.h>
SoftwareSerial gsm(2, 3); // RX, TX

void setup() {
  pinMode(0, INPUT);
  pinMode(1, INPUT);

  Serial.begin(9600); //
  gsm.begin(9600);
}
void loop() {
  if (gsm.available()) {
    Serial.write(gsm.read());
  }
  if (Serial.available()) {
    gsm.write(Serial.read());
  }
}
```

4. Le programme global du module GSM

```
// Objectif: Recevoir un SMS qui contient #cmd1#cmd2# ... #cmdn*
#include <SoftwareSerial.h>

SoftwareSerial gsm(2, 3); // RX, TX
int commandes[13] = {0};

void setup() {
  pinMode(0, INPUT);
  pinMode(1, INPUT);

  Serial.begin(9600);
  gsm.begin(9600);

  /*
     NE PAS UTILISER LES PINS 0, 1, 2, 3, 9
     Pin 9 EST ATTACHE A PWRKEY (utilisé pour l'alimentation)
  */

  // ALLUMER LE MODULE
    pinMode(9, OUTPUT);
    digitalWrite(9, HIGH);
    delay(1000);// on envoie une implusion pour une durée plus que
400µS
    digitalWrite(9, LOW);}
```

```
void loop() {
  gsm.write("AT+CMGL=\"ALL\"\r\n");
  String str = getMessage();

  if (str != "no_cmd") // dans ce cas il y a des commandes à
executer
  {
    extractCMD(str);
    doCMD();
    // Supprimer tout les messages
    gsm.write("AT+CMGDA=\"DEL ALL\"\r\n");
  }
  else
    Serial.println("Pas de commande a extraire");

  delay(1000);
}

void doCMD() {

  for (int i = 0; i < 13; i++)
  {
    if (commandes[i] == 0)
    {
      Serial.print("Fin de commandes\n");
      break;
    }
    else
    {
      if (commandes[i] >= 30 && commandes[i] < 60)
      {
        // On doit élliminer 0, 1, 2 , 3, 9
        //il s'agit d'allumer
        int pin = commandes[i] - 30;
        if (pin != 0 && pin != 1 && pin != 2 && pin != 3 && pin !=
9)
        {
          digitalWrite(pin, HIGH);
          Serial.print(pin);
          Serial.println("    >>    ON");
        }
        else{
          Serial.println("Impossible d'allumer ce pin!!");
        }
      }
      else if (commandes[i] >= 60)
      {
        int pin = commandes[i] - 60;
```

```
        if (pin != 0 && pin != 1 && pin != 2 && pin != 3 && pin !=
9)
        {
          digitalWrite(pin, LOW);
          Serial.print(pin);
          Serial.println("    >>    OFF");
        }
        else
        {
          Serial.println("Impossible d'éteindre ce pin!!");
        }
      }
    }
  }
}

void extractCMD(String msg) //2ème solution
{
  for (int k = 0; k < 13; k++)
    commandes[k] = 0;

  int dizaine, unitee;

  int i = 0; int j = 0;
  for (int i = 0; i < msg.length() ; i++)
  {
    dizaine = (int)msg.charAt(i) - 48;
    unitee = (int)msg.charAt(i + 1) - 48;
    // char '1' --> 49 en ascii on doit retrancher 48
    commandes[j] =  10 * dizaine + unitee;
    j++;
    i += 2;
  }
}

String getMessage()
{
  if (gsm.find("#") == true)
  {
    boolean b = true;
    String msg = "";
    while (b)
    {
      if (gsm.available()) {
        char c = gsm.read();
        if (c == '*')
          b = false;
        else
```

```
          msg += c;
      }
    }
    return msg;
  }
  else
  {
    return "no_cmd";
  }
}
```

5. Utilisation du composant LM35 pour la mesure de la température

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("La température: ");
  Serial.print((String)getTemp());
  Serial.print("°C");
  delay(500);
}

int getTemp()
{
  int val = analogRead(0);
  int mv = (val / 1024.0) * 5000;
  return (mv / 10);
}
```

6. Automatisation des commandes AT module WiFi

```
#include <SoftwareSerial.h>

SoftwareSerial esp(2,3);

void setup() {
  esp.begin(112500);
  Serial.begin(112500);
  delay(10000);
}

void loop() {
  delay(5000);
  digitalWrite(10, LOW);
```

```
  // Etablir une connexion TCP
  startTCP();
  clear_serial();

  // Envoyer une demande d'envoi
  sendReq();
  clear_serial();

  // Génération de nombres aléatoires et de la température
  int x,y,z;
  x = getTemp();
  y = 1 + (rand()%30);
  z = 1 + (rand()%30);

  // Envoi des nombres générés
  sendCMD(x, y, z);

  Serial.print("Temperature : ");
  Serial.print(x);
  Serial.println(" °C");
  Serial.println("-------------------------------------");

  clear_serial();
  // Fermer la connexion TCP
  closeTCP();
  clear_serial();

  digitalWrite(10, HIGH);
  // Attente pendant 5 secondes
  delay(5000);
}


int getTemp()
{
  // Calcul de la température
  int val = analogRead(0);
  // Pour le LM35:  10mV --->> 1°C
  float mv = (val / 1024.0) * 5000;
  return int(mv / 10);
}
void clear_serial()
{
  while(esp.available())
    esp.read();
}
```

```
void startTCP()
{
  esp.print("AT+CIPSTART=\"TCP\",\"184.106.153.149\",80\r\n");
}
void closeTCP()
{
  esp.print("AT+CIPCLOSE\r\n");
}

void sendReq()
{
    esp.print("AT+CIPSEND=100\r\n");
}

void sendCMD(int val, int val1, int val2)
{
    String cmd = "GET
https://api.thingspeak.com/update?api_key=90AQPXEAO4B70153&field1=";
    cmd += val;
    cmd += "&field2=";
    cmd += val1;
    cmd += "&field3=";
    cmd += val2;
    cmd += "\r\n\r\n";
    esp.print(cmd);
}
```