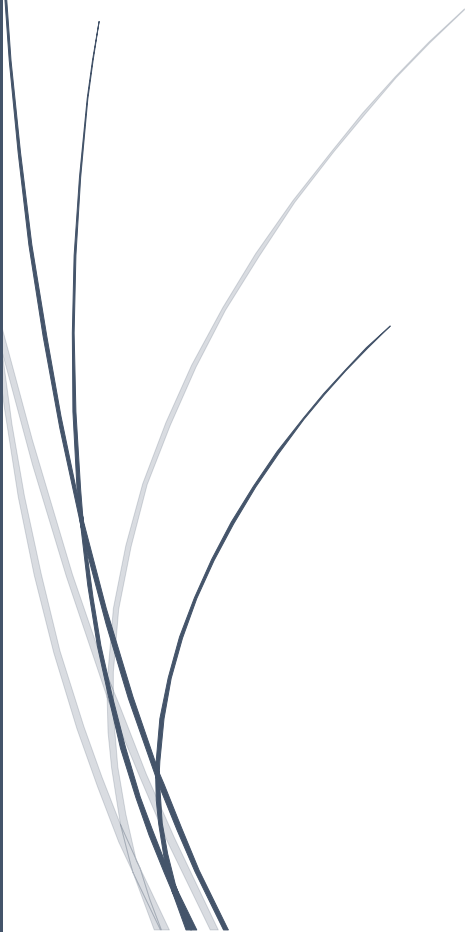


A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from the bar, containing the date.

23/01/2019

Plateformes de développement IoT

Rapport

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Baptiste Beaupère, Alexandra Hoffman
ESAIP IR2019

Table des matières

| | |
|--|----|
| Introduction..... | 2 |
| I. Prise en main | 2 |
| 1) Objectif..... | 2 |
| 2) Réalisation | 2 |
| Mise en place..... | 2 |
| Compilation des premiers programmes..... | 3 |
| II. Noyau Linux | 4 |
| 1) Test sur QEMU..... | 4 |
| 2) Validation sur Raspberry | 7 |
| III. Projet | 8 |
| 1) Objectif..... | 8 |
| 2) Détails du projet | 8 |
| 3) Mise en place de la base de données..... | 11 |
| 4) Restitution des données sur Android | 12 |
| Annexes | 14 |

Introduction

Ce rapport présentera les trois projets effectués du 21 au 23 janvier 2019 :

- La prise en main d'un Raspberry
- La mise en place d'un noyau Linux
- La réalisation d'un projet de récupération et restitution de données

I. Prise en main

1) Objectif

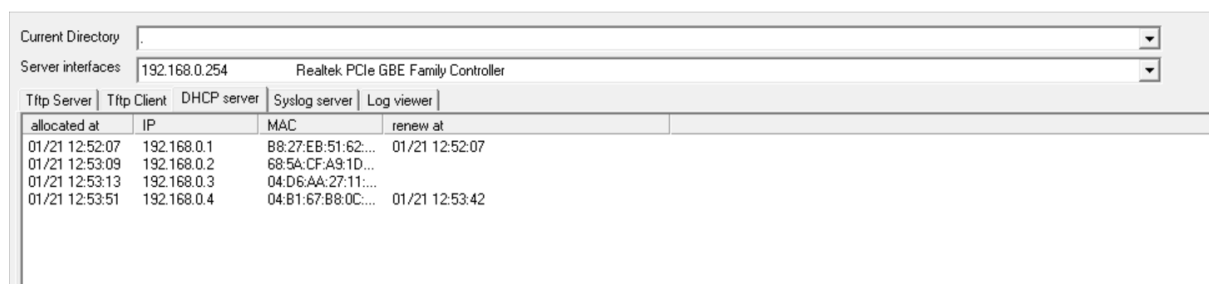
Le premier TP avait pour objectif de nous connecter à la Raspberry à distance pour ensuite compiler et exécuter un programme sur cette dernière. Puisque les objets connectés sont rarement équipés d'écran et de clavier, nous devons tout d'abord trouver un moyen d'interagir avec la carte depuis un accès distant, pour ensuite la configurer et enfin exécuter le programme voulu.

2) Réalisation

Mise en place

Lors de la première utilisation de la Raspberry, celle-ci possède rarement un moyen de communiquer avec l'extérieur directement. Nous avons donc dû trouver un moyen pour communiquer avec la carte depuis un autre poste.

Pour ce faire, le moyen le plus adapté nous a semblé être la connexion SSH. Pour l'activer sur la Raspberry, il nous a suffi de créer un fichier nom « SSH » dans la partition /boot de la carte SD. Une fois le SSH activé sur la carte, il a fallu trouver un moyen de lui attribuer une adresse IP sur le réseau. Pour cela, nous l'avons reliée directement à notre PC via un câble Ethernet en configurant un DHCP sur l'ordinateur. La carte étant par défaut configurée pour fonctionner en adressage IP dynamique, celle-ci demande automatiquement une adresse à l'ordinateur au démarrage. Nous avons pour cela utilisé TFTP64, visible sur la capture ci-dessous.



The screenshot shows the TFTP64 application window. At the top, there are fields for 'Current Directory' and 'Server interfaces' (192.168.0.254, Realtek PCIe GBE Family Controller). Below these are tabs for 'Tftp Server', 'Tftp Client', 'DHCP server', 'Syslog server', and 'Log viewer'. The 'DHCP server' tab is active, displaying a table of DHCP leases.

| allocated at | IP | MAC | renew at |
|----------------|-------------|--------------------|----------------|
| 01/21 12:52:07 | 192.168.0.1 | B8:27:EB:51:62:... | 01/21 12:52:07 |
| 01/21 12:53:09 | 192.168.0.2 | 68:5A:CF:A9:1D:... | |
| 01/21 12:53:13 | 192.168.0.3 | 04:D6:AA:27:11:... | |
| 01/21 12:53:51 | 192.168.0.4 | 04:B1:67:B8:0C:... | 01/21 12:53:42 |

Figure 1: Capture de TFTP64

Une fois que la carte a obtenu une adresse IP, nous avons pu y accéder via SSH et modifier les autres paramètres réseaux pour lui attribuer une adresse IP fixe (plus simple d'utilisation), une passerelle par défaut et un serveur DNS pour accéder à internet (ici notre ordinateur portable).

Une fois la connexion établie, nous avons créé une machine virtuelle Ubuntu sur notre ordinateur, placée sur le même réseau que la Raspberry pour pouvoir compiler plus facilement les programmes et faciliter l'exercice suivant qui consiste en la manipulation du noyau Linux.

Le schéma suivant détaille les branchements effectués :

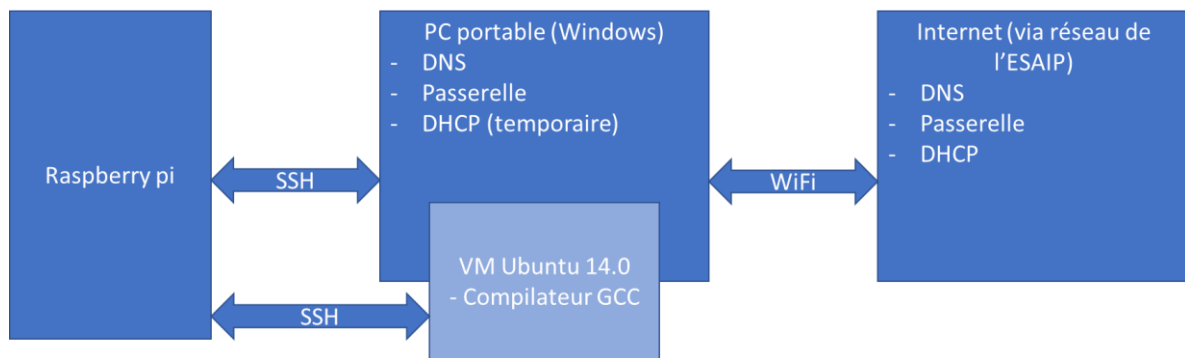


Figure 2: détails des interactions

Compilation des premiers programmes

Une fois l'accès à distance correctement configuré sur la carte, nous avons pu compiler et exécuter nos premiers programmes. Le premier programme lancé était le suivant :

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main() {
5
6  string prenom;
7  cout << "Test d'écriture et de compilation d'un programme C++ sur Rasperry";
8  cout << "- Compilation sur cible depuis hote distant " << endl ;
9  cout << "Quel est ton prenom ? " ;
10 cin >> prenom;
11 cout << "Bonjour " << p;|
12 }

```

Figure 3: premier programme effectué sur Raspberry

Nous l'avons compilé directement sur la carte grâce au compilateur « GCC » déjà installé.

L'étape suivante était l'utilisation d'un Cross compiler pour permettre de compiler un programme depuis la VM Ubuntu pour ensuite le pousser sur la carte et l'exécuter correctement. Nous avons tenté de réaliser la compilation du cross compiler nous-même sur la VM Ubuntu mais malheureusement, des problèmes de compatibilité nous en ont empêché.

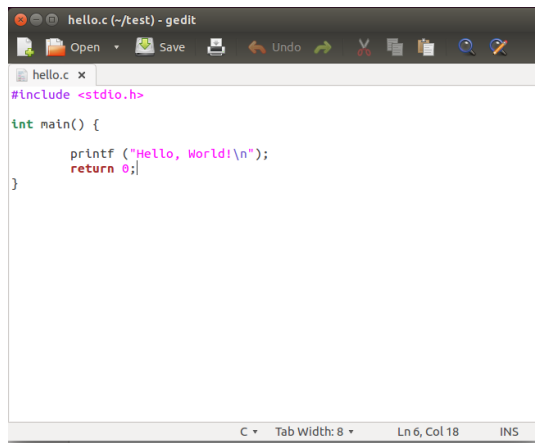
A la place, nous avons téléchargé le cross compiler directement disponible sur le GitHub de Raspbian (<https://github.com/raspberrypi/tools>) Et l'avons décompressé sur notre VM Ubuntu. Le programme que nous souhaitions utiliser était « /master/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian-x64/bin/arm-linux-gnueabi-hf-gcc », qui correspond au crosscompiler pour compiler des programmes en C pour des processeurs du type ARM. En ajoutant cet exécutable à la variable d'environnement, il nous a été possible de compiler le programme souhaité pour l'exécuter ensuite sur la Raspberry.

```

alex@alex-VirtualBox:~/TPKernel/linux-rpi-3.10.y$ export PATH=${PATH}:/home/alex/crosscompiler/tools-master/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian-x64/bin

```

Figure 4: commande pour l'ajout de la variable d'environnement



```
hello.c (-/test) - gedit
#include <stdio.h>

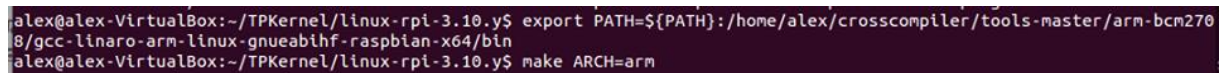
int main() {
    printf ("Hello, World!\n");
    return 0;
}
```

Figure 5 : programme en C

II. Noyau Linux

L'objectif de la deuxième séance était la recompilation du noyau de l'OS Raspbian, pour ensuite le tester sur un environnement QEMU et finalement l'utiliser sur notre carte Raspberry.

Pour la compilation, nous avons téléchargé l'image du noyau depuis le Github de Raspbian. Une fois le fichier téléchargé, nous avons utilisé les commandes « make ARCH=arm menuconfig » et « make ARCH=arm oldconfig » pour configurer correctement le noyau avant de le compiler avec la commande suivante :



```
alex@alex-VirtualBox:~/TPKernel/linux-rpi-3.10.y$ export PATH=${PATH}:/home/alex/crosscompiler/tools-master/arm-bcm270
8/gcc-linaro-arm-linux-gnueabi/raspbian-x64/bin
alex@alex-VirtualBox:~/TPKernel/linux-rpi-3.10.y$ make ARCH=arm
```

Figure 6: commande sur Ubuntu pour la configuration du noyau

1) Test sur QEMU

Nous avons ensuite testé notre image du noyau grâce à QEMU.

Lorsqu'on démarre le noyau Linux, l'écran se fige et le noyau plante : c'est le Kernel Panic.

```

QEMU - Press Ctrl-Alt to exit mouse grab
RPC: Registered tcp NFSv4.1 backchannel transport module.
NetWinder Floating Point Emulator V0.97 (double precision)
Installing knfsd (copyright (C) 1996 gki@monad.sub.de).
ifconfig: (C) 1996 Red Hat, Inc.
ROMFS MTD (C) 2007 Red Hat, Inc.
msgmni has been set to 34
Block layer SCSI generic (bsg) driver version 0.4 loaded (major 254)
io scheduler noop registered
io scheduler deadline registered
clcd-pll1x dev:20: PLL10 rev0 at 0x10120000
clcd-pll1x dev:20: Versatile hardware UGA display
Console: switching to colour frame buffer device 80x60
brd: module loaded
PCI: enabling device 0000:00:0c:0 (0100 -> 0103)
sym0: <895> rev 0x1 at pci 0000:00:0c:0 irq 93
sym0: No NVRAM, ID 7, Fast-40, LVD, parity checking
sym0: SCSI BUS has been reset.
scsi0: 0:0:2:0: CD-ROM QEMU QEMU CD-ROM 2.0. PQ: 0 ANSI: 5
scsi target0:0:2: Tagged command queuing enabled, command queue depth 16.
scsi target0:0:2: Beginning Domain Validation
scsi target0:0:2: Domain Validation skipping write tests
scsi target0:0:2: Ending Domain Validation
sr0: 0:0:0:0: cdrom: Uniform CD-ROM driver Revision: 3.20
cdrom: Uniform CD-ROM driver Revision: 3.20
physmap platform flash device: 040000000 at 34000000
physmap flash: 0: Found 1 x32 devices at 0x0 in 32-bit bank. Manufacturer ID 0x00
0000 Chip ID 0x00000000
Intel/Sharp Extended Query Table at 0x0031
Using buffer write method
smc91c11x v1.1, Sep 22 2004 by Nicolas Pitre <nico@fluxnic.net>
eth0: SMC91C11x FD (rev 12) at c22ca000 IRQ 57 [nowait]
eth0: Ethernet address: 82:54:00:12:34:56
mousedev: PS/2 mouse device common for all mice
TCP: cubic registered
NET: Registered protocol family 17
UFP: Support v0.3: Implementor 41 architecture 1 part 20 variant b rev 5
input: AT Raw Set 2 keyboard as /devices/fpga:06/serial0/input0
input: LMP Generic Explorer Mouse as /devices/fpga:07/serial1/input1
UFS: Cannot open root device "lfs3" or unknown-block(31,3): error 5
Please append a correct "root=" boot option; here are the available partitions:
 0000 1648758 sr0: driver: sr
lfs0 65536 mtdblock0 (driver?)
Kernel panic - not syncing: UFS: Unable to mount root fs on unknown-block(31,3)
CPU: 0 PID: 0 Comm: swapper Not tainted 3.10.3800 #1
[<c001933c>] (unwind_backtrace+0x8/0x9c) from [<c0016548>] (show_stack+0x10/0x14)
[<c0016548>] (show_stack+0x10/0x14) from [<c031f268>] (panic+0x7c/0x1b8)
[<c031f268>] (panic+0x7c/0x1b8) from [<c041fed8>] (mount_block_root+0x1b0/0x25c)
[<c041fed8>] (mount_block_root+0x1b0/0x25c) from [<c0420164>] (mount_root+0xe4/0x18c)
[<c0420164>] (mount_root+0xe4/0x18c) from [<c04202f0>] (prepare_namespace+0x164/0x184)
[<c04202f0>] (prepare_namespace+0x164/0x184) from [<c041fb4c>] (kernel_init_freeable+0x160/0x1b0)
[<c041fb4c>] (kernel_init_freeable+0x160/0x1b0) from [<c031d9fc>] (kernel_init+0/0x100)
[<c031d9fc>] (kernel_init+0x0/0x100) from [<c0013218>] (ret_from_fork+0x14/0x3c)

```

Figure 7 : Kernel Panic au lancement du noyau Linux

Cette erreur est due à un problème de compatibilité avec le Kernel (arm_linux : 3.10, SD : 4.0.1), Libc(arm_linux : 6, SD :6). Le programme /init n'arrive pas à se lancer, le démarrage ne peut donc pas s'effectuer.

Suite à ce test, nous avons apporté des modifications au code du noyau pour empêcher le Kernel Panic : nous avons créé une boucle affichant un message (ici « toto ») avant la commande causant le Kernel Panic (voir figures 8 et 9). La ligne en question se trouvait dans le fichier « do_mount », fichier appelé par la fonction « main.c »

```

main.c x ops_fstype.c x journal.c x do_mounts.c x
#else
const char *b = name;
#endif

get_fs_names(fs_names);
retry:
for (p = fs_names; *p; p += strlen(p)+1) {
    for (;;) {
        printk("toto \n");
    }
    int err = do_mount_root(name, p, flags, root_mount_data);
    switch (err) {
        case 0:
            goto out;
        case -EACCES:
            flags |= MS_RDONLY;
            goto retry;
        case -EINVAL:
            continue;
    }
}

```

Figure 8: Modification du code pour empêcher le Kernel Panic



Figure 9: Résultat de la modification du code

2) Validation sur Raspberry

Une fois le test sur Qemu validé, nous avons copié le fichier « kernel.img » pour remplacer celui déjà présent sur notre Raspberry. En reprenant la même configuration qu'avant le noyau démarre. Seuls les paramètres d'affichage de l'invite de commande étaient différents, nous les avons configurés de la manière suivante pour pouvoir brancher un écran sur la carte et ainsi lire les chaînes de caractères renvoyées par le programme.

- Application des étapes précédentes à l'environnement Raspberry
⇒ Modification des configurations nécessaire (figures 10 et 11)

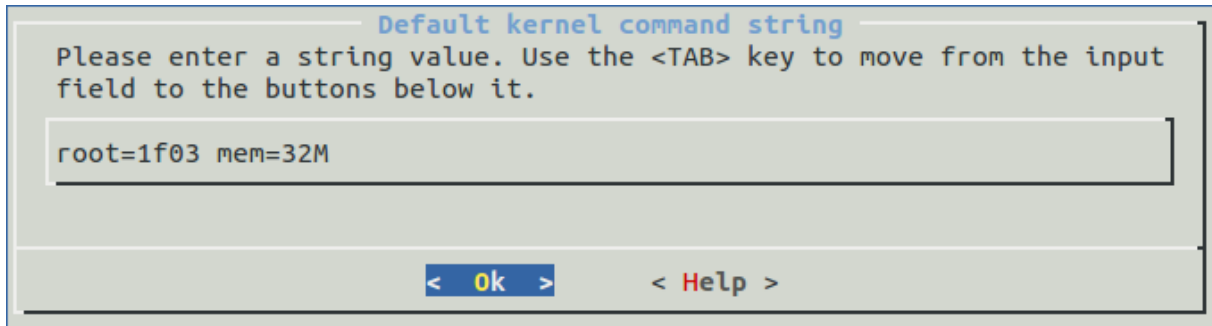


Figure 10 : Modification des configurations 1/2

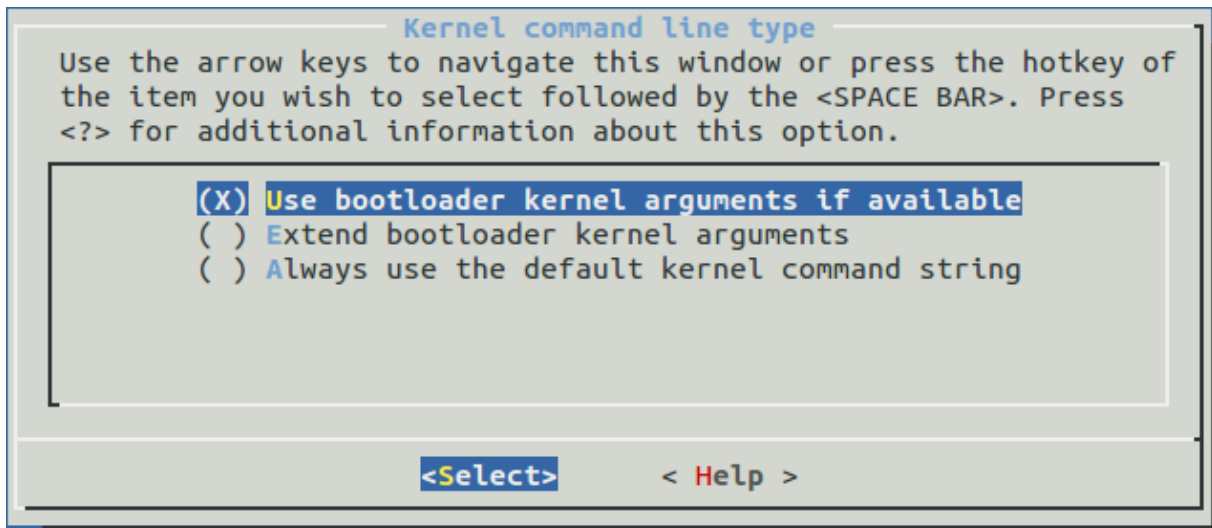


Figure 11 : Modification des configuration 2/2

III. Projet

1) Objectif

L'objectif est de récupérer des données d'une photorésistance sur Raspberry à partir de l'Arduino, et de les transférer à une base de données. Ces données sont restituées par téléphone, par le biais d'une application mobile.

2) Détails du projet

Une photorésistance est branchée en série avec une résistance de 9100 Ω sur une Arduino MEGA2560 (voir figure 12). L'Arduino est elle-même reliée à une Raspberry Pi A+ en I2C. Pour cela, les PIN SDA et SCL sont utilisées pour connecter la Raspberry et l'Arduino (SDA pour les données, SCL pour la clock).

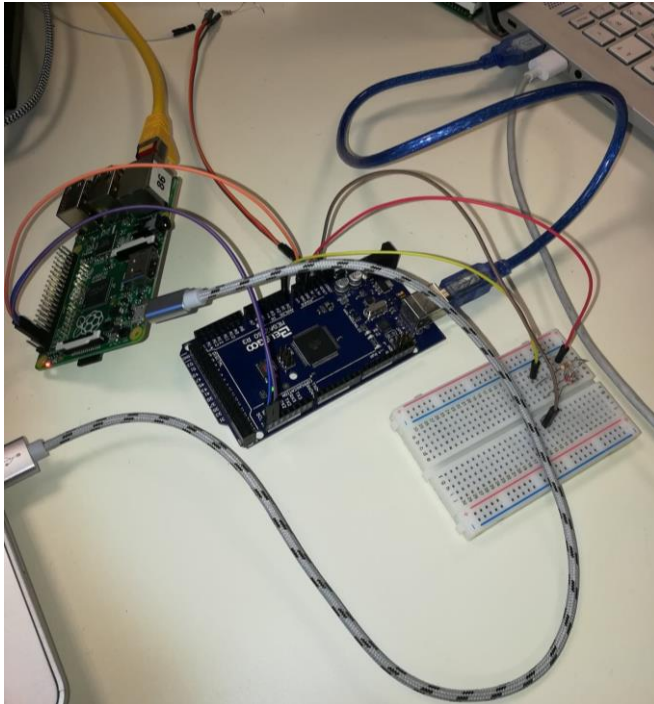


Figure 12: Montage du projet

Connexion des ports :

- Pour l'I2C :
 - SCL sur Arduino relié avec le port 5 de Raspberry
 - SDA sur Arduino avec le port 3 de Raspberry
 - Les masses reliées (Gnd sur Arduino, port 6 sur Raspberry)
- Pour la photorésistance :
 - Plaque d'essai reliée à l'alimentation (3,3V) et à la masse de la carte Arduino afin de créer un circuit fermé
 - Données prélevées sur le port A0 de l'Arduino (port analogique)

Fonctionnement : La Raspberry demande les données à l'Arduino toutes les 30 secondes ; elle les stocke ensuite sur une base de données MongoDB avec l'API MLAB. Cette API sera ensuite utilisée par l'application Android pour récupérer les données et les afficher sous forme de graphique à l'utilisateur.

Le code Python que nous avons créé afin de récupérer les données sur la Raspberry est visible sur la figure 13. La fonction askValues() communique avec l'Arduino pour récupérer la valeur lue par celle-ci. Ce nombre pouvant osciller entre 0 et 1023, il est envoyé en deux parties par l'Arduino, puis reconstitué dans la fonction askValues() de la Raspberry. Ce nombre est ensuite utilisé pour calculer la résistance de la photorésistance. On arrondit ensuite ce nombre, puis on le stocke dans la base de données préalablement créée, avec d'autres données comme la date et l'heure d'envoi. On attend ensuite 30 secondes, avant de recommencer le processus.

```
import smbus
import time
import requests
import datetime
bus = smbus.SMBus(1)

# This is the address we setup in the Arduino Program
address = 0x12
#Send value to arduino
def writeNumber(value):
    bus.write_byte(address, value)
    return -1
#Get value from Arduino
def readNumber():
    number = bus.read_byte(address)
    return number
#Get the value detected by the sensor
def askValues():
    writeNumber(1)
    number1 = readNumber()*(2**8)
    time.sleep(1)
    writeNumber(2)
    number2 = readNumber()
    return (number1+number2)

#add reocrded value to db every 30s
while True:
    data = askValues()
    res1= 9100.0
    todayTime=datetime.datetime.now()
    buffer= data * 3.3
    Vout= (buffer)/1024.0
    buffer= (3.3/Vout) -1
    R2= res1 * buffer
    v2=round(R2,2)
    print "resistance: ", R2
    value = {"day": todayTime.day, "month": todayTime.month, "year":todayTime.year, "hour": todayTime.hour, "minutes": todayTime.minute, "value":v2 }
    resp = requests.post('https://api.mlab.com/api/1/databases/raspberry/collections/resistance?apiKey=vQNIw4w4UJJ_VG-6_dt96XZc7ieICcAa', json=value)
    if resp.status_code != 200:
        print(resp.status_code)
    print('Created value. ID: {}'.format(resp.json()["_id"]))
    time.sleep(30)
```

Figure 13 : Code Python pour récupérer les informations et les envoyer dans la base de données

La figure 14 montre le code effectué sur l'Arduino afin de récupérer les informations de la photorésistance et de les transmettre à la Raspberry. La commande `Wire.begin()` permet de déclarer la transmission I2C entre la Raspberry et l'Arduino. Les commandes `Wire.onReceive()` et `Wire.onRequest()` permettent d'associer les fonctions de callback à exécuter lors de la réception d'un ordre de la Raspberry et lors de la demande d'une réponse de la part de la Raspberry. Ici, lorsqu'un ordre est reçu, la valeur de la photorésistance est relevée et séparée en deux parties afin de pouvoir la transmettre à la Raspberry. Comme expliqué précédemment, celle-ci enverra d'abord le code « 1 » pour recevoir la première partie du nombre, puis « 2 » pour la deuxième partie.

```
#include <Wire.h>

#define SLAVE_ADDRESS 0x12
int order = 0 ;
int answer = 0 ;
int beginning = 0 ;
int rest ;

// pour la mesure de la résistance
int analogPin= A0;
int raw= 0;

void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(9600); // start serial for output
  // initialize i2c as slave
  Wire.begin(SLAVE_ADDRESS);
  // define callbacks for i2c communication
  Wire.onReceive(receiveData);
  Wire.onRequest(sendData);
  Serial.println("ready");
}

void loop() {
  raw= analogRead(analogPin);
  Serial.println(raw);
  delay(1000);
}

// callback for received data
void receiveData(int byteCount) {

  while(Wire.available()) {
    order = Wire.read();
    Serial.print("order received : ");
    Serial.println(order);
    if (order == 1){
      raw= analogRead(analogPin);
      Serial.println(raw);
      rest = raw ;
      beginning = raw >> 8 ;
      answer = beginning ;
    }
    if (order == 2){
      answer = rest ;
    }
  }
}

// callback for sending data
void sendData() {
  Wire.write(answer);
}
```

Figure 14 : Code pour Arduino du prélèvement des données

```

Arduino receiving : 581
resistance: 6938.55421687
Created value, ID: {'_id': 'u'5c4890301f6e4f5f90cb22ad'}
```

```

Arduino receiving : 580
resistance: 6966.20689655
Created value, ID: {'_id': 'u'5c4890505d0e651a0230877c'}
```

```

Arduino receiving : 574
resistance: 7134.14634146
Created value, ID: {'_id': 'u'5c4890705d0e651a023087d5'}
```

```

Arduino receiving : 591
resistance: 6667.17428088
Created value, ID: {'_id': 'u'5c48908f5d0e651a0230884f'}
```

```

Arduino receiving : 575
resistance: 7105.91304348
Created value, ID: {'_id': 'u'5c4890af1f6e4f5f90cb241e'}
```

```

Arduino receiving : 577
resistance: 7049.74003466
Created value, ID: {'_id': 'u'5c4890cf5d0e651a023089ec'}
```

```

Arduino receiving : 586
resistance: 6801.70648464
Created value, ID: {'_id': 'u'5c4890ef5d0e651a02308b1a'}
```

```

Arduino receiving : 603
resistance: 6353.39966833
Created value, ID: {'_id': 'u'5c48910e1f6e4f5f90cb2485'}
```

Figure 15: Résultat du code lancé sur Raspberry

Lorsque la Raspberry crée une nouvelle entrée dans la base de données, un id est créé automatiquement pour cette entrée. La figure 15 montre le résultat affiché lorsque le code est lancé sur la carte Raspberry. Afin de suivre le fonctionnement du programme, nous avons décidé d'afficher la valeur mesurée toutes les trente secondes, ainsi que l'ID de l'entité ajoutée à la base de données. Afin de vérifier la justesse des données reçues, nous avons vérifié le comportement que devrait avoir la photorésistance en fonction de la lumière (voir annexe 1).

3) Mise en place de la base de données

Afin de pouvoir stocker les données et y accéder, il faut tout d'abord créer une API. Pour cela, il faut activer une option dans les paramètres de compte mLab. Ainsi, nous pouvons utiliser la clé de l'API pour accéder à la base de données et y ajouter des entrées à partir du code python (voir figure 16).

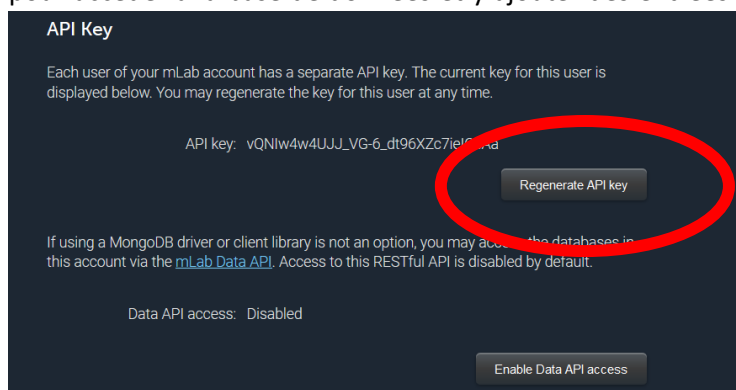


Figure 16: Compte utilisateur de mLab

Ensuite, il faut créer une base de données et une collection dans la base de données, où seront stockées les données relevées. Ainsi, le chemin pour soumettre la requête sera de la forme suivante : `https://api.mlab.com/api/1/databases/{dbName}/collections/{collectionName}?apiKey= {key}`

En lançant le programme sur la Raspberry, des données sont ajoutées toutes les 30 secondes. Voici le rendu :

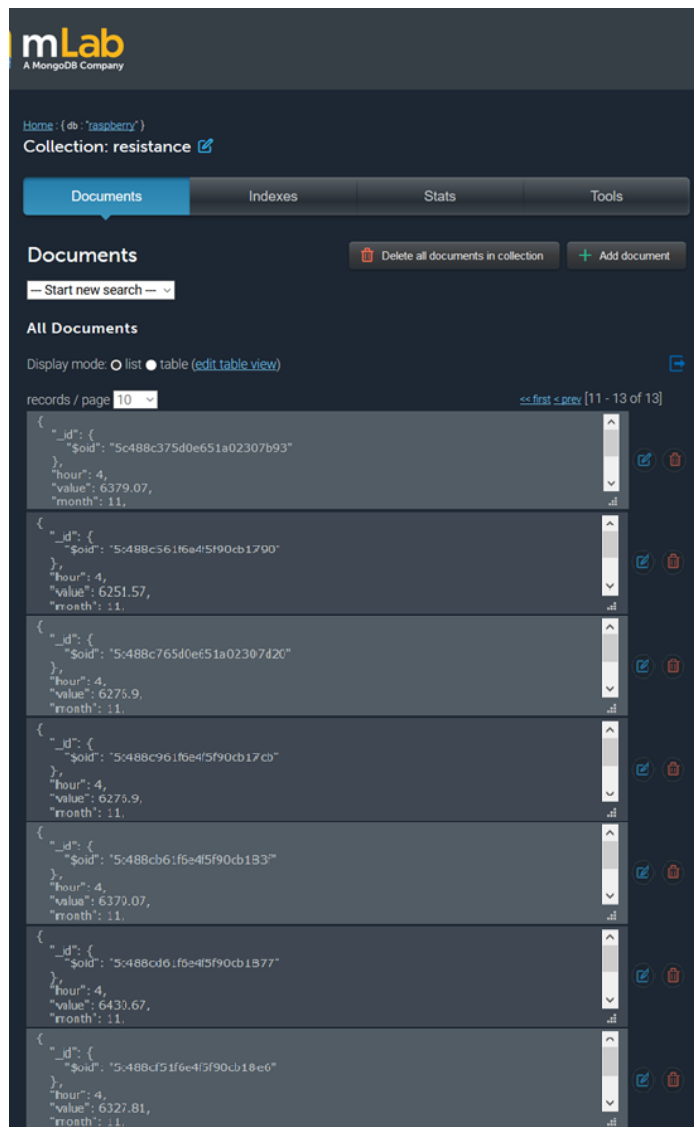


Figure 17: Contenu de la base de données créée avec mLab

4) Restitution des données sur Android

L'application Android est constituée d'une zone de texte indiquant la dernière valeur relevée, ainsi que d'une zone où un graphique va être tracé au fur et à mesure de la réception des nouvelles données.

```
public void sendRequest() {
    RequestQueue queue = Volley.newRequestQueue(context);
    String url = "https://api.mlab.com/api/1/databases/raspberry/collections/resistance?apiKey=vQNIw4w0JJ_VG-6_dt96XZc7ieICoAa";
    JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.GET, url, null, new Response.Listener<JsonObject>() {
        @Override
        public void onResponse(JSONObject response) {
            Log.v("TAG", "VALUE", response.toString());
            lastResponse = response;
            try {
                changeText();
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.v("TAG", "VALUE", error.toString());
        }
    });
    queue.add(jsonObjectRequest);
}
```

Figure 18: Extrait du code pour Android : l'envoi d'une requête

Une requête est envoyée pour récupérer les données à partir de la base de données, en utilisant le même chemin que celui indiqué précédemment pour la requête python. Le code de la fonction permettant d'envoyer la requête et de récupérer les données dans un Json est visible sur la figure 18.

Le graphique est mis à jour en temps réel. Lorsqu'une nouvelle donnée est entrée dans la base de données par la Raspberry, l'application récupère les informations de cette donnée. Ainsi, lorsqu'une nouvelle mesure est prise, sa date d'ajout et sa valeur sont affichées au-dessus du graphique, puis la valeur est ajoutée sur le graphique. La figure 19 correspond au code réalisant ceci.

```
public void updateGraph(){
    JSONArray json = lastResponse;

    if(json != null){
        LineChart chart = (LineChart) findViewById(R.id.chart);
        List<Entry> entries = new ArrayList<Entry>();
        for (int i=0; i<json.length(); i++) {

            // turn your data into Entry objects
            try {
                int time = json.getJSONObject(i).getInt("hour") + json.getJSONObject(i).getInt("minutes");
                entries.add(new Entry(time, json.getJSONObject(i).getInt("value")));
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }

        LineDataSet dataSet = new LineDataSet(entries, "Label"); // add entries to database
        LineData lineData = new LineData(dataSet);
        chart.setData(lineData);
        chart.invalidate(); // refresh
    }
}
```

Figure 1919: Extrait du code pour Android : la mise à jour du graphique

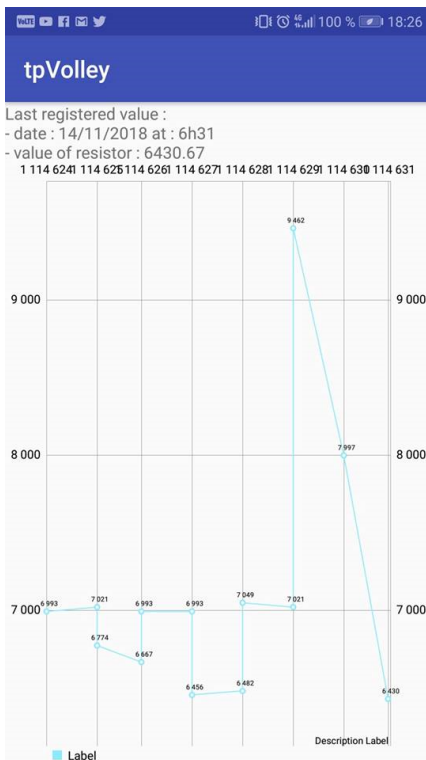
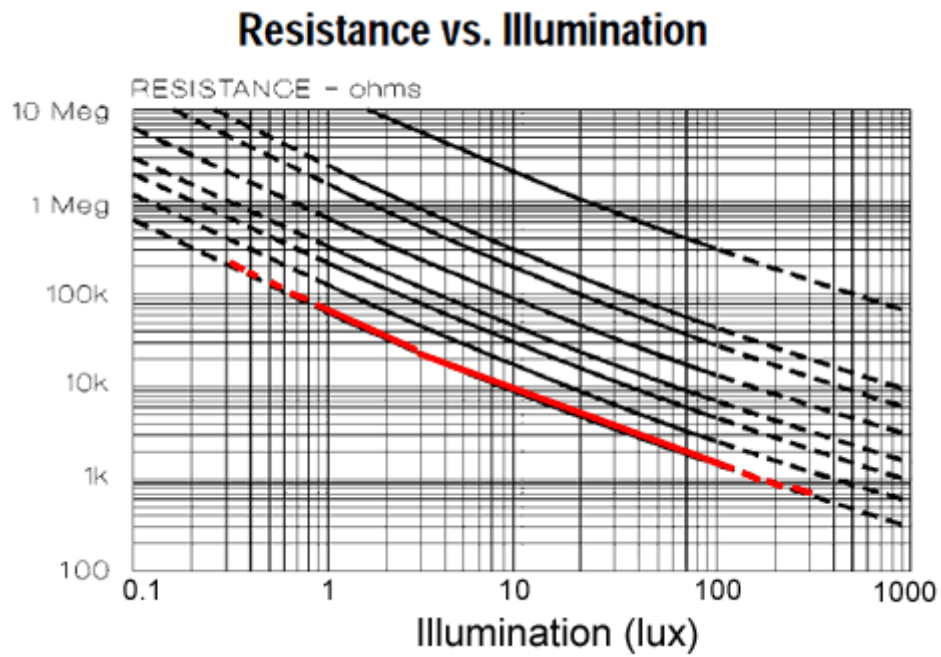


Figure 20: Capture de l'application mobile



Annexe 1 : Courbe d'évolution de la résistance d'une photorésistance en fonction de l'illumination

Source : <https://www.instructables.com/id/Photocell-tutorial/>