Département IMA/4A (S8)
Année scolaire 2011/2012

Rapport de projet de 4ème année

Sujet : Carte du maraudeur

CHEMIN Aymeric (Systèmes communicants)
WANG Quanquan (Systèmes autonomes)

Contenu

Introduction

Dans le cadre de notre quatrième année d'étude à l'école d'ingénieur Polytech'Lille, nous avons travaillé à la réalisation d'un projet intitulé «Carte du maraudeur ». L'objectif de ce projet était de fabriquer un système permettant de localiser un objet ou une personne dans l'école et d'afficher son déplacement à l'aide de la technologie RFID.

Dans ce rapport, nous allons revenir en détail sur le sujet du projet. Nous vous présenterons les périphériques utilisés. Nous détaillerons le fonctionnement du système et enfin nous évoquerons les améliorations que nous avons apporté ou auxquelles nous avons pensé.

Remerciements

Ce projet nous a permis d'aboutir à la réalisation concrète d'un système, améliorant ainsi nos connaissances et notre compréhension des choses. Nous tenons avant tout à remercier les encadrants pour l'aide et les conseils qu'ils nous ont apporté durant ce projet :

Mr. Nicolas Defrance, Maître de conférence en Electronique à Polytech'Lille, pour son aide tout au long du projet, en particulier pour extraire les données d'un transpondeur RFID.
Mr. Alexandre Boé, Maître de conférence en Electronique à Polytech'Lille, pour son aide tout au long du projet, en particulier sur les commandes AT.
Mr. Thomas Vantroys, Maître de conférence en Informatique à Polytech'Lille, pour son aide sur l'émulation d'un port série sur des broches classiques d'un Arduino.

Mais également

Mr. Xavier Redon, Maître de conférence en Informatique à Polytech'Lille, pour son aide sur les différentes techniques utilisées dans les communications sans fil.
Mr. Thierry Flamen, Responsable du pôle d'électronique à Polytech'Lille pour ses précieux conseils en matière de soudure, ainsi que pour les composants électroniques qu'il nous a fourni durant les phases de tests.
Mme. Emmanuelle Pichonat, Maître de conférence en Electronique à Polytech'Lille, pour l'idée de ce projet et sa bonne humeur.

1 - Présentation

A l'aide de lecteurs de tag RFID il est possible de localiser un objet muni d'une étiquette. Ce projet a pour but d'associer des lecteurs RFID (de faible portée) avec des modules Zigbee afin de transmettre les données reçues par les lecteurs RFID. La portée d'un module Zigbee étant elle aussi limitée à 30m théoriques avec obstacles, le système doit gérer le transfert des données entre plusieurs balises Zigbee jusqu'à arriver vers un PC. Le PC devra ensuite gérer une carte sur laquelle sera affichée la position de ces TAGS.

2 - Périphériques utilisés

2.1 Carte RFID

Les cartes transpondeur RFID que nous utilisons sont des cartes opérant sur la fréquence 125KHz conçu sur la base d'un circuit EM4102 d'EM MICROELECTRONIC. Elles disposent d'une mémoire de 5 octets (+2 octets de parités) faisant office de numéro de série unique pré configuré en usine. Elles ne peuvent être utilisées qu'en lecture.

Architecture de la mémoire :

Byte numéro	Transpondeur
1	ID1 (8bits)
2	ID2 (8bits)
3	ID3 (8bits)
4	ID4 (8bits)
5	ID5 (8bits)
6	ID6 (8bits)
7	Parité 1 (8bits)
8	Parité 2 (6bits)

Les 5 octets de mémoire seront donc utilisés pour représenter une personne ou un objet dans l'école.

2.2 Lecteur de transpondeur RFID

Le module qui nous permet la lecture sans contact de transpondeur RFID est le module « OEM RFID UM005 ». Une alimentation +5V et une antenne externe sont nécessaires pour son fonctionnement. Il dispose d'une sortie série au format RS-232 aux niveaux logiques TTL sur laquelle est envoyée une suite d'octets relatifs au « code usine » préconfiguré sur chaque transpondeur lorsque celui-ci est situé à portée de l'antenne.

Le module fonctionne sur le principe du « auto-read », à savoir que tant qu'un TAG RFID est présent devant l'antenne, le flot de données est envoyé 2 fois par seconde sur le port série.

Exemple d'une trame envoyée par le module :

Adresse du module	Longueur de la trame	Identifiant de l'opération	Identifiant du TAG	Code	CRC (2 octets)
01	0B	01	ID1 … ID5	FF	XX XX

2.3 Arduino uno

Les modules Arduino™ sont des plates-formes de prototypage micro contrôlées "open-source" architecturée autour d'un microcontrôleur Atmel™ ATmega328 et programmables via un langage proche du "C".

L'Arduino sera utilisé pour

- Récupérer les identifiants des TAGS RFID
- Effectuer des vérifications (CRC)
- Transmettre les données au PC gérant la carte.

Il est le cœur de ce que nous appellerons plus tard une « base »

2.4 Modules Zigbee

Pour la communication sans fil , nous utilisons des module radio XBee®. Ils permettent de mettre en place rapidement une liaison radio 2.4Ghz de qualité avec un nombre de composants annexes très réduits. Ils assurent la transmission bidirectionnelle de signaux numériques séries (format RS-232 - niveau logique 0/3,3 V).

Afin de simplifier les connections, nous utilisons aussi 2 supports :

Xbee explorer : conçu pour raccorder rapidement et simplement un module XBee™ sur le port USB d'un PC.
Xbee shield pour Arduino : conçu pour être utilisé sur la base d'un Arduino Uno afin de connecter les broches Rx/Tx de l'Arduino aux broches Tx/Rx de la puce Xbee tout en adaptant les niveaux de tension.

2.5 Module complet

A l'aide de tout le composants précèdent, nous réalisons plusieurs « bases » qui permettront de récupérer les TAG des transpondeur RFID puis de les communiquer à un « ordinateur de contrôle ».

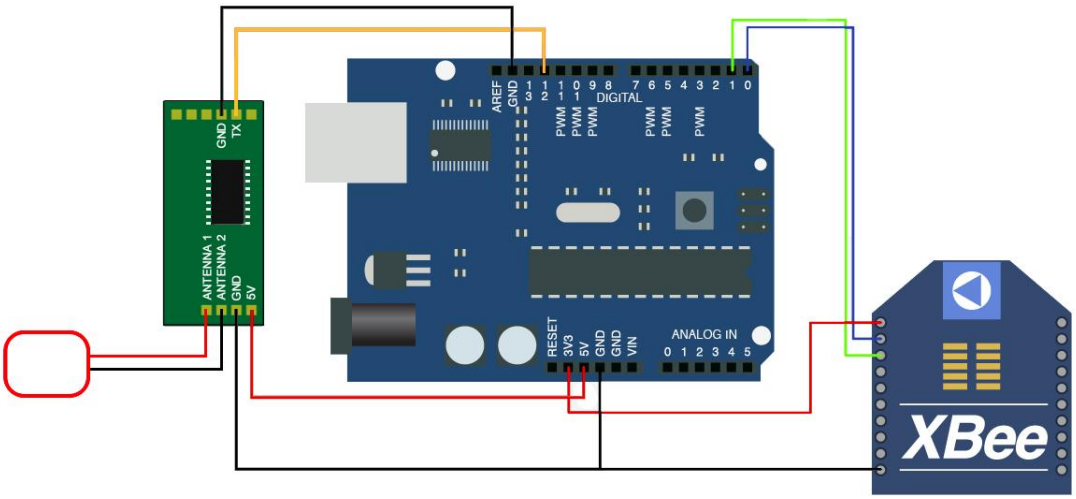
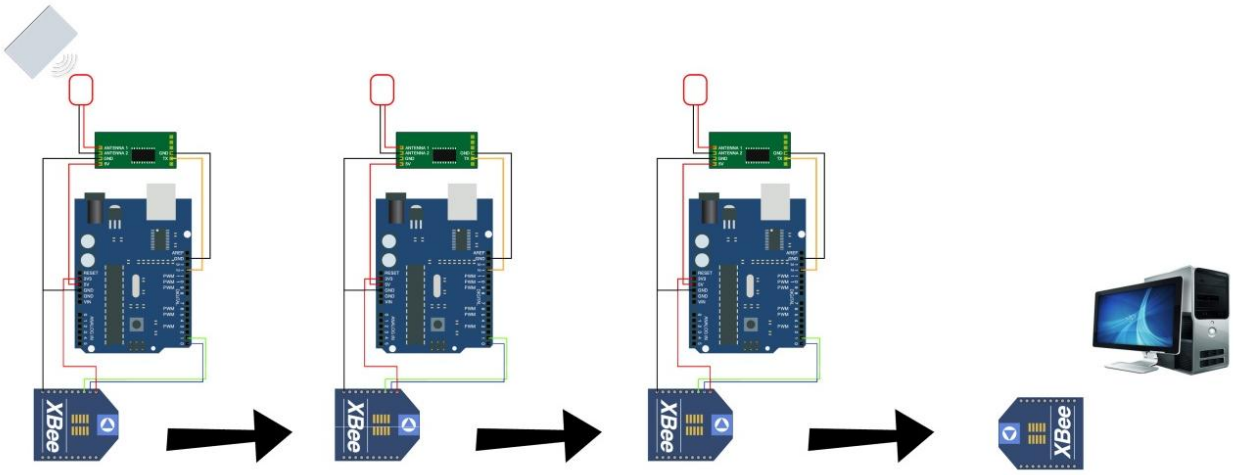


Schéma d'une « base »

3 – Fonctionnement

3.1 Schéma de principe

Le prince est le suivant , lorsqu'un transpondeur RFID est placé à proximité d'une antenne d'un lecteur, son identifant doit être lu , puis envoyé à un ordinateur de contrôle. Toutefois , la portée d'un module Xbee n'etant que de 100m théorique en champ libre et 30m théorique avec obsactles, nous réalisons un reseau de « bases ».



Chaque base communique à la suivante les informations qu'elle détient jusqu'à que ces informations arrivent en bout de chaine à un ordinateur. Cette technique permet d'augmenter la portée globale du système, toutefois si un des maillons venait à ne plus fonctionné, une partie de la chaine serait hors service.

3.2 Lecture des identifiants des cartes

La première étape consiste à lire les données contenue dans la carte RFID lorsque celle-ci est présente à proximité de l'antenne reliée au transpondeur RFID. Comme l'UM005 fonctionne en auto-read, il nous suffit d'intercepter les données envoyées par le module sur la broche TX. L'Arduino UNO ne possédant qu'un seul port série qui sera utilisé plus tard par le module Xbee, il nous a fallu trouver un autre moyen pour lire ces données. Grace à la libraire Software Serial, il est possible d'émuler un port série sur des broches « classiques » d'entrée/sortie d'un Arduino. Nous avons donc intégré cette libraire dans notre programme Arduino. Cette librairie fournie des primitives de lecture haut niveau semblables à celles utilisées par le port série « officiel » (Serial.read() , Serial.print() …)

Le code suivant permet de récupérer les 11 octets envoyés par le module UM005 :

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(12,13);
// Définie les broches 12 et 13 comme un port série (12=RX 13=TX)

int  val;
int bytesread;
unsigned char code[11];
//Tableau servant à stocker la trame de 11 octets

void setup()
{
      mySerial.begin(9600);  //Initialise le port à 9600 bauds
}


void loop()
{

      if(mySerial.available())
      //Si des données sont présentes sur le port série
      {

            bytesread = 0;
            while(bytesread<11)
            {

                  if( mySerial.available() > 0)
                  {
                        val = mySerial.read();
                        code[bytesread] = val;
                        bytesread++;
                  }
            }
            mySerial.flush();  //Vide le buffer série
      }

}
```

3.3 Vérification de lecture

Sur les 11 octets récupérés, seulement 5 représentent l'identifiant de la carte, toutefois les 2 derniers octets sont utiles à la vérification de l'intégrité des données puisqu'il s'agit du code CRC de la trame. Afin de rendre plus fiable notre système et d'utiliser un peu plus la puissance des Arduino, nous effectuons une vérification du code CRC dans notre programme. La valeur du code est calculée à partir du polynôme $x^{16} + x^{12} + x^5 + 1$ et de la valeur initiale 0x0000. Le code du programme de calcul du CRC est fourni par le constructeur en langage C, toutefois ce dernier n'est pas très clair. De plus la datasheet nous indique que le calcul du CRC ce fait sur 5 bits alors qu'il se fait en réalité sur 9 bits ! Nous utilisons donc le code suivant :

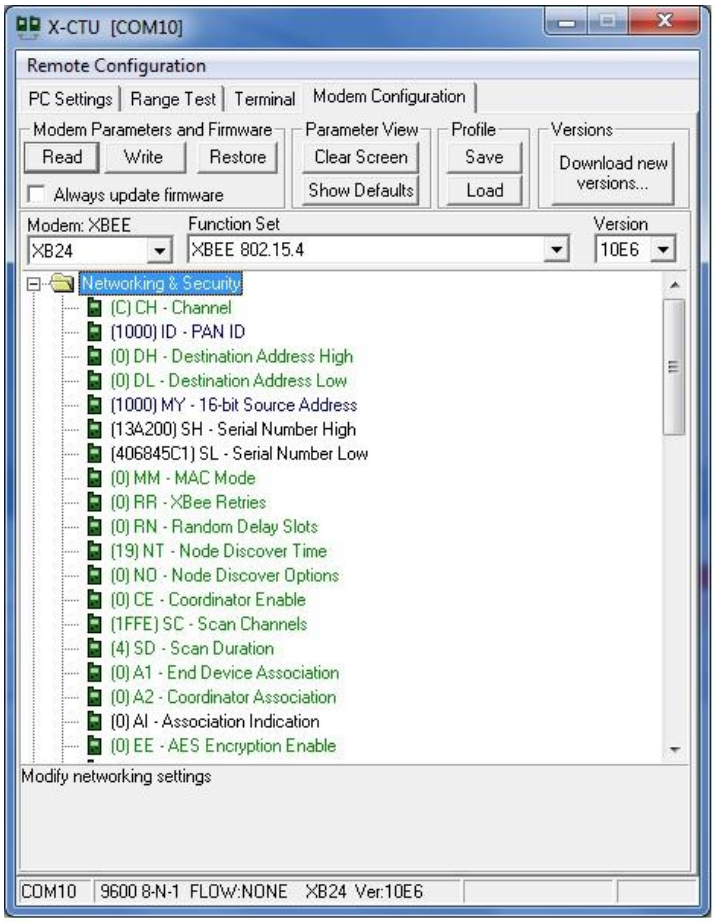
Apelle de la fonction :

```
unsigned short calculated_crc;
//Variable dans laquelle sera stockée le CRC calculé

unsigned char code[11];
//Tableau contenant la trame de 11 octets

CRC16(code,&calculated_crc,9);

void CRC16(unsigned char * Data, unsigned short * CRC, unsigned char
Bytes){
  int i, byte;
  unsigned short C;
  *CRC = 0x0000;
  for (byte = 1; byte <= Bytes; byte ++, Data ++)
  {
    C = ((*CRC >> 8) ^ *Data) << 8;
    for (i = 0; i < 8; i ++)
    {
      if (C & 0x8000)
        C = (C << 1) ^ 0x1021;
      else
        C = C << 1;
    }
    *CRC = C ^ (*CRC << 8);
  }
}
```

3.4 Envoi / réception

Une fois que les données ont correctement été lues et stockées en mémoire, celle-ci doivent être acheminées jusqu'à « l'ordinateur de contrôle ». Pour cela les modules Xbee doivent être configurés préalablement pour pouvoir communiquer en chaine (multi-sauts).

Nous avons donc configuré les modules Xbee à l'aide du programme X-CTU. Nous utilisons l'adressage 64bits : chaque module Xbee possède une adresse unique garantie par le constructeur, seules les données à destination de cette adresse seront lues. Il suffit alors de renseigner dans les champs « Destination Adresse » (HIGH et LOW) l'adresse du module suivant à contacter.



Afin de clarifier la compréhension des informations sur l'ordinateur de contrôle qui reçoit toutes les données en provenance de tous les modules, nous avons mis au point une syntaxe très simple :

Lors d'un envoi d'identifiant de carte, chaque module envoie la phrase suivante :

Base X Carte YYYYY

Où X est le numéro du module et YYYYY l'identifiant de la carte. Ainsi il est facile de localisé la carte YYYYY sachant que la position du module X est connue à l'avance.

En Arduino, le code est le suivant :

```
// On vérifie que le CRC calculé correspond à celui reçu (octets 10 et 11)
if( (calculated_crc>>8)==code[9] && (calculated_crc&0x00FF) ==
code[10]){
            Serial.print("Base ");
            Serial.print(base);
            Serial.print(" Carte ");
            for(int i=3;i<8;i++)
            {
              Serial.print(code[i],HEX);
            }
            Serial.println("");
            bytesread = 0;
}
```

Lorsqu'un Arduino reçoit des données de la part d'un autre Arduino il lui suffit de les retransmettre au module suivant, on réalise donc un simple « echo »

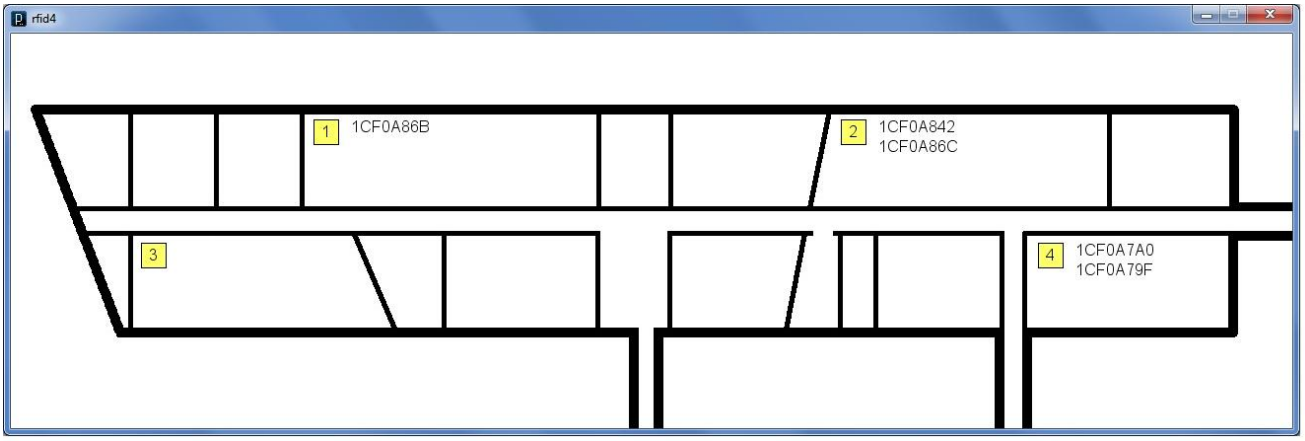
```
if (Serial.available()){
      data=Serial.read();
      Serial.print(data);
}
```

Expérimentalement, un simple « echo » n'est pas suffisant et provoque beaucoup d'erreurs, nous avons donc intégré une vérification de syntaxe entre les modules (voir partie amélioration) afin de limiter la propagation de messages erronés.

3.5 Traitement et affichage

En bout de chaine, le module Xbee n'est pas connecté à un Arduino mais à un ordinateur qui permet le traitement et l'affichage des données. Nous avons décidé d'utiliser le programme Processing pour la programmation de notre système d'affichage car sa prise en main est assez simple du fait de sa ressemblance avec le code Arduino et des nombreux exemples présents sur internet.

L'objectif principal du programme est de fournir un affichage simple et clair de la position des tags RFID sur un plan. Nous avons donc positioné des « bases » à coté desquelles viennent s'inscire les tags RFID passés à portée :



Rendu final de notre application

Le langage utilisé sur Processing est une variante du Java et propose ainsi tous les avantages de la programmation objet, mais également des outils d'affichage graphiques simple et une gestion des interruptions série. Notre programme comporte alors 3 éléments importants :

- Une fenêtre graphique avec un fond représentant une carte d'un bâtiment de l'école (par exemple)
- Un objet « base » qui comporte :
  - Un numéro
  - Une position (x,y) sur la fenêtre d'affichage
  - Une liste des cartes attachées à cette base
  - Des fonction d'ajout/suppression/impression des cartes attachées à cette base
- Une fonction d'interruption série qui permet de recevoir les données en provenance de la puce Xbee

Ces éléments permettent facilement de charger une carte quelconque et de positionné des base sur cette carte en entrant leurs coordonnées (x,y). L'affichage des identifiants est entièrement paramétré sur la position de la base et ne nécessite donc pas d'être modifié.

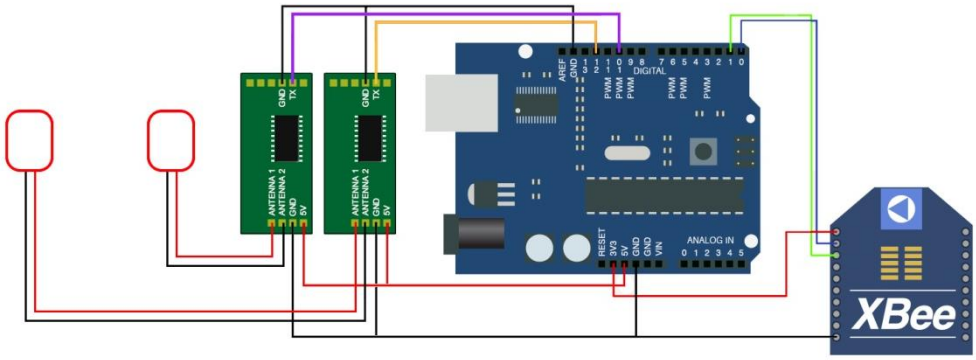
4 – Améliorations

4.1 Vérification de la syntaxe entre les modules

Apres les premiers essais, nous avons constaté que de nombreux messages incohérents arrivaient au PC, et que des identifiants incorrects s'affichaient à l'écran. Afin de limiter la propagation de ces messages, nous avons effectué une vérification de notre syntaxe Base XX Carte YYYYY entre les modules au lieu de se contenté d'un simple écho. Les tests mené après l'ajout de cette fonctionnalité ne comportaient quasiment plus d'erreurs

4.2 Module « double »

Afin d'utiliser un peu plus la puissance des Arduino, nous avons tenté de connecté 2 lecteurs de transpondeur RFID sur 2 port émulés via la librairie « SoftwareSerial »



Pour que cela fonctionne, il faut écouter tour à tour les ports séries :

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(12,13);
SoftwareSerial mySerial2(10,11);

void setup()
{
      mySerial.begin(9600);
      mySerial2.begin(9600);
}
void loop() {
      mySerial.listen();
      while(mySerial.available()>0)
      {
            …
      }
      mySerial2.listen();
      while(mySerial2.available()>0)
      {
            …
      }
}
```

Expérimentalement on constate que le temps de lecture est un peu plus long que pour un module simple. En effet il faut l'Arduino écoute le bon port série. Le temps n'est toutefois pas si énorme puisqu'il est de l'ordre de grandeur de la seconde.

4.1 Bidirectionnalité du flux de données

Un des inconvénients de notre système est que le flux de données est unidirectionnel, de la base la plus éloignée jusqu'au PC. Nous avons donc voulu rendre ce flux bidirectionnel, or les modules Xbee ne peuvent être configurés qu'avec une seule adresse de destination ou en « broadcast ». Pour pallier ce problème nous avons voulu modifier l'adresse de destination « à la volée » en utilisant l'Arduino et les commandes AT.

+++ : Démarre le mode de commande.
ATDH + paramètre (0xFFFFFFFF) : Modifie ou lit les 32 bits MSB de l'adressage destinataire.
ATDL + paramètre (0xFFFFFFFF) : Modifie ou lit les 32 bits LSB de l'adressage destinataire.
ATWR : Sauve les paramètres dans la mémoire non volatile
ATCN : Pour quitter le mode commande.

Le programme suivant devrait donc théoriquement changer l'adresse destinataire :

```
Serial.print("+++");
delay(1500);
Serial.print("ATDHxxxx\r");
delay(50);
Serial.print("ATWR\r");
delay(50);
Serial.print("ATDLxxxx\r");
delay(50);
Serial.print("ATWR\r");
delay(50);
Serial.print("ATCN\r");
```

Or nous avons constaté que l'adresse n'est pas changée et l'Arduino est rebooté en boucle. Nous n'avons donc **PAS** réussi à rendre notre flux de donné bidirectionnel.

Conclusion

Ce projet nous aura permis d'élargir nos champs de connaissance en manipulant de nouvelles technologie comme la technologie RFID ou encore la communication sans fil basée sur le protocole Zigbee.

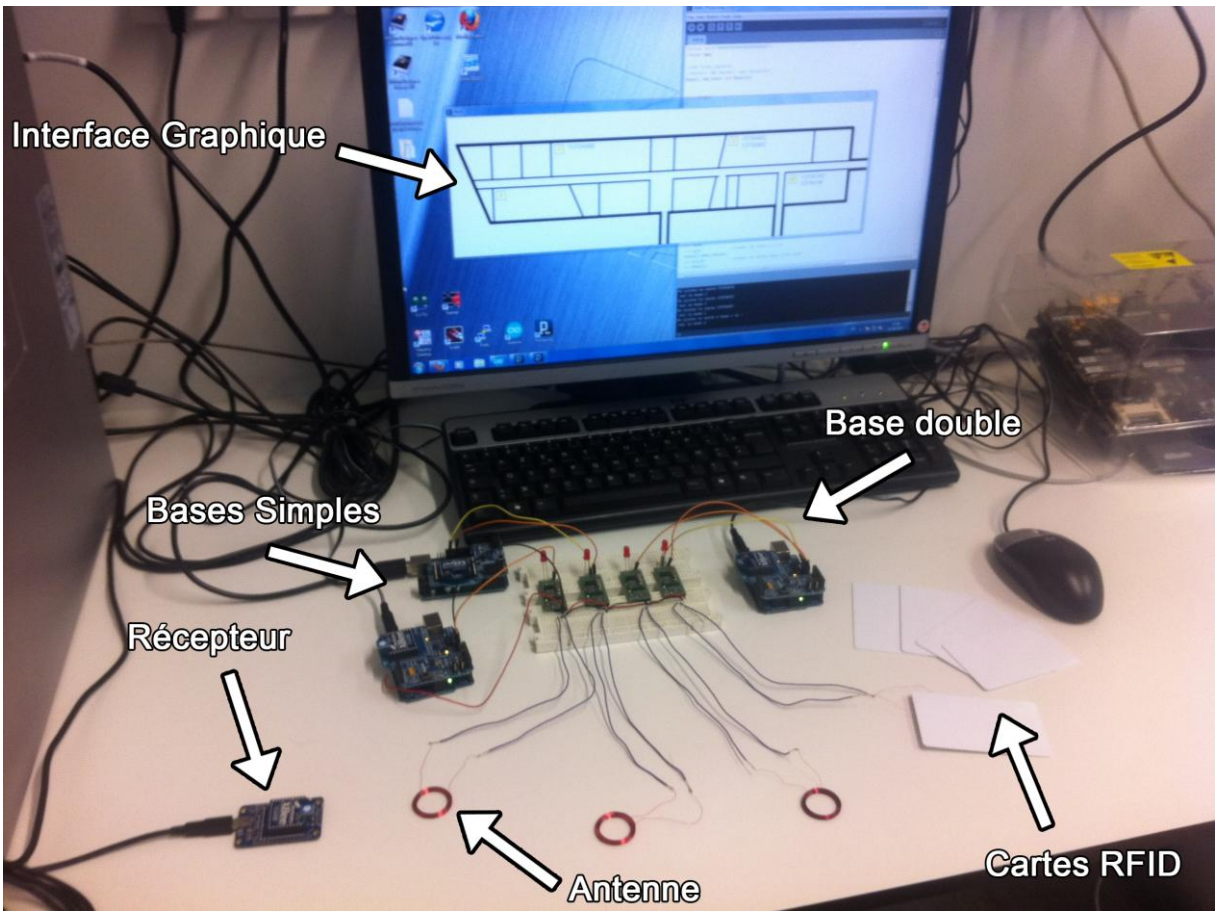
Les différents langages de programmation utilisés tout au long de ce projet nous ont permis d'améliorer nos connaissances et notre adaptation à ces langages.

La réalisation d'un affichage graphique simple est une satisfaction car n'importe quel utilisateur est capable de le comprendre contrairement à un affichage dans un terminal.

Enfin nos capacités à travailler en équipe et à respecter des délais se sont agrandies, chose importante pour notre future vie professionnelle.

Annexes

Photo du système :



Code de l'interface sous Processing :

```
// Variables globales
import processing.serial.*;
Serial port;
PFont f;
String text="000000000000000000";
PImage img;

Base[] tab_base= new Base[4];

// Objet Carte
class Carte {
        //Variables d'instance
        String id;
        String base;

        //Constructeur
        Carte(String text)
        {
                id = text.substring(13);
                base =  text.substring(5,6);
        }
}

//Objet Base
class Base {
        //Variables d'instance
        int n_b;
        Carte[] base_cartes;
        int n_c_b;
        int base_x;
        int base_y;

        //Constructeur
        Base(int n,int x,int y) {
                n_b=n;
                base_cartes = new Carte[20];
                n_c_b=0;
                base_x = x;
                base_y = y;
        }
```

```
	//Fonction d'ajout d'une carte à une base
	void ajouter(Carte c) {
		if (n_c_b<19) {
			base_cartes[n_c_b]=new Carte("00000000000000000");
			base_cartes[n_c_b].id=c.id;
			base_cartes[n_c_b].base=c.base;
			n_c_b=n_c_b+1;
		}
		else
			println("Nombre max de carte atteint !");
	}

	//Test de présence d'une carte sur la base
	int test_supprimer(Carte c) {
		int i;
		int s=0;
		for (i=0;i<n_c_b;i++) {
			if (base_cartes[i].id.equals(c.id))     //si elle deja exist,on la suppremer
			{
				if (base_cartes[i].base.equals(c.base)) s=1;
				n_c_b=n_c_b-1;
				base_cartes[i]=base_cartes[n_c_b];
			}
		}
		return s;
	}

	//Fonction d'affichage des cartes
	void display() {
		int i;
		for (i=0;i<n_c_b;i++)
		{
			fill(0);
			if (n_b==1)
				text(base_cartes[i].id,base_x+25,base_y+i*20);
			else if (n_b==2)
				text(base_cartes[i].id,base_x+25,base_y+i*20);
			else if (n_b==3)
				text(base_cartes[i].id,base_x+25,base_y+i*20);
			else if (n_b==4)
				text(base_cartes[i].id,base_x+25,base_y+i*20);
		}
		stroke(0);
		fill(255,255,102);
		rectMode(CENTER);
		rect(base_x,base_y,25,25);
		fill(0);
		textFont(f);
		text(n_b,base_x-5,base_y+5);
	}
}
```

```
//Fonction d'initialisation du programme
void setup() {
	size(1300,400);
	f = createFont("Arial",16,true);
	port = new Serial(this, "COM3", 9600);
	port.bufferUntil('\n');
	int i=0;
	int n;
	tab_base[0]=new Base(1,320,100);
	tab_base[1]=new Base(2,855,100);
	tab_base[2]=new Base(3,145,225);
	tab_base[3]=new Base(4,1055,225);
	img = loadImage("photo.jpg");
}

//Fonction d'affichage
void draw() {
	background(0);
	image(img,0,0);
	for(int i=0;i<4;i++)
		tab_base[i].display();
}

//Fonction d'interruption série
void serialEvent (Serial port) {
	int ajout=0;
	text = port.readStringUntil('\n');
	Carte c=new Carte(text);
	for(int i=0;i<4;i++)
	{
		if (tab_base[i].test_supprimer(c)!=1) ajout=1;
	}
	if(ajout==1) {
		println(c.id+" "+c.base);
		if (c.base.equals("1"))
			tab_base[0].ajouter(c);
		else if (c.base.equals("2"))
			tab_base[1].ajouter(c);
		else if (c.base.equals("3"))
			tab_base[2].ajouter(c);
		else if (c.base.equals("4"))
			tab_base[3].ajouter(c);
	}
}
```

Code Arduino

```
#include <SoftwareSerial.h>

//Variables globales
SoftwareSerial mySerial(12,13);

int base = 1;
byte buf;
int  val = 0;
unsigned char code[11];
int bytesread = 0;
int mark=0;
String data="";

//Fonction d'initialisation
void setup() {
        Serial.begin(9600);
        mySerial.begin(9600);
        Serial.print("Init base ");
        Serial.print(base);
        Serial.println(" ok ");
}

//Fonction principale
void loop() {
        // Si des données sont présentes sur le port série, on les propages au module suivant
        if (Serial.available()>0) {
                while((Serial.available()>0)&&(data.length()<23)) {
                        data += char(Serial.read());
                        delay(2);
                        mark=1;
                }
                if(mark==1) {
                        //Vérification de la syntaxe "Base X Carte YYYYY"
                        if ( (data.length()<=23 && data[0]=='B' && data[1]=='a'&& data[2]=='s'&&
                        data[3]=='e' && data[7]=='C' && data[8]=='a'&& data[9]=='r'&&
                        data[10]=='t'&& data[11]=='e'&& data[15]!='0' && data[17]!='0' &&
                        data[19]!='0') || (data.length()<=40 && data[0]=='I' && data[1]=='n'&&
                        data[2]=='i'&& data[3]=='t')) {
                                for(int i=0;i<data.length();i++)
                                        Serial.write(data[i]);
                        }
                        else
                                Serial.print("Erreur de lecture");
                        mark==0;
                }
        }
```

```
	else {
		//Si des données sont envoyées par le lecteur RFID
		if(mySerial.available()) {
			bytesread = 0;
			while(bytesread<11) {
				if( mySerial.available() > 0) {
					val = mySerial.read();
					code[bytesread] = val;
					bytesread++;
				}
			}
			mySerial.flush();
		}
		//Si les données lues sont correctes on les transmet au module suivant
		if(bytesread == 11) {
			unsigned short crc,calculated_crc;
			CRC16(code,&calculated_crc,9);
			if( (calculated_crc>>8)==code[9] && (calculated_crc&0x00FF) == code[10]) {
				Serial.print("Base ");
				Serial.print(base);
				Serial.print(" Carte ");
				for(int i=3;i<8;i++)
					Serial.print(code[i],HEX);
				Serial.println("");
				bytesread = 0;
			}
			else
				Serial.println("WRONG CRC !!;
			delay(250);
		}
	}
}


//Fonction de calcul du CRC
void CRC16(unsigned char * Data, unsigned short * CRC, unsigned char Bytes)
{
	int i, byte;
	unsigned short C;
	*CRC = 0x0000;
	for (byte = 1; byte <= Bytes; byte ++, Data ++) {
		C = ((*CRC >> 8) ^ *Data) << 8;
		for (i = 0; i < 8; i ++) {
			if (C & 0x8000)
				C = (C << 1) ^ 0x1021;
			else
				C = C << 1;
		}
		*CRC = C ^ (*CRC << 8);
	}
}
```