Cuadros Alexandre
Taffin Valentin
IMA 5

Projet de Fin d'Études

«Réaliser deux trackers GPS permettant de suivre à distance le trajet d'un coureur »

Années 2016-2017

REMERCIEMENTS

Nous tenons à remercier nos tuteurs Mr. Boe et Mr. Vantroys de nous avoir fourni le matériel nécessaire à la réalisation de ce projet et Mr. Flamen pour son aide et ses conseils apportés lors de la réalisation des cartes électroniques.

SOMMAIRE:

I : Présentation du projet

1) Problématique

J'effectue des activités sportives en extérieur et je souhaiterais connaître ma position et mes performances à n'importe quel moment de la journée.

2) Contexte

Notre projet de fin d'études est destiné aux sportifs et/ou randonneurs qui souhaitent connaître les trajets qu'ils ont parcourus dans la journée et visualiser leurs performances ainsi que la difficulté qu'ils peuvent rencontrer lors de leurs circuits.

3) Objectif

Notre projet consiste à réaliser deux dispositifs :

- Le dispositif Smartphone : Le sportif est équipé d'un gant connecté donnant les informations sur son état de forme et ses coordonnées GPS sont récupérées à l'aide d'une application android.
- La dispositif LoRa : un matériel autonome, petit et basse consommation, permettant au sportif de connaître sa position en temps réel.

Les coordonnées GPS de chaque dispositif sont transmises sur un serveur afin d'être visualisées sur un site internet.

4) Contraintes

Pour la partie Smartphone :

- Le gant connecté doit posséder une batterie suffisante pour la durée de l'entraînement.
- Le gant connecté ne doit pas gêner le porteur durant son activité.
- L'application doit être réalisée sur smartphone Android.

Pour la partie LoRa :

- Le système doit être basse consommation.
- Facilement transportable dans une poche.
- Le dispositif doit être rapide, résister à des chocs ainsi qu'à l'humidité.

5) Technologies utilisées

a) Android Studio et Java:

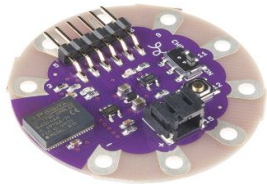
L'application mobile est faite à l'aide de l'environnement de développement "Android Studio" en langage de programmation Java.

Android Studio est un environnement de développement pour développer des applications Android.

Pour cela, la compilation du programme doit être à la même version Android que celui du téléphone (ici Android 5.1).

Nous avons utilisé la technologie Android puisque nous possédons des smartphones utilisant ce système d'exploitation.

b) Carte Lilypad Simblee et Bluetooth BLE:



Le Bluetooth Low Energy (BLE) permet un débit du même ordre de grandeur que le Bluetooth (1 Mb/s) pour une consommation d'énergie 10 fois moindre (environ 10mA). Le BLE possède une bande passante plus limitée et une très faible consommation alors que le Bluetooth a un niveau d'émission plus élevé et possède une portée plus grande. Dans notre cas, une plus grande portée d'émission n'est pas nécessaire (tels que le Bluetooth ou encore la Wifi) puisque nous souhaitons communiquer entre le composant Lilypad situé sur le bras de l'utilisateur et le Smartphone situé dans sa poche avec un débit correct.

La Lilypad Simblee est une carte de développement qui nous permet d'interagir avec des applications mobiles via Bluetooth Low Energy.

c) Arduino IDE:

Le logiciel Arduino IDE permet de se connecter au matériel Arduino (et Genuino) pour insérer les programmes et communiquer avec eux. Nous l'utilisons donc pour créer le programme de la Lilypad Simblee ainsi que des microprocesseurs Atmega328p.

d) Le réseau de l'internet des objets Lora, Sigfox et Qowisio:

Ci-dessous un tableau comparatif des réseaux de faible puissance :

	Qowisio	**SigFox**	**LoRa**
Coût	10 centimes/objet	abonnement de 14 euros / an et par objets. Diminue avec le nombre	3000 euros pour commercialiser
Distance d'émission	3 km en ville 60 km en campagne	1 km en ville 20 km en campagne	1 km en ville 20 km en campagne
Fréquence utilisée	868 MHz	868 MHz	433 MHz (USA) 868 MHz (EU)
Type de communication	Unidirectionnelle	Bidirectionnelle	Bidirectionnelle

L'Alliance LoRa est un réseau étendu de faible puissance (LPWAN) exploitable avec une batterie sans fil adaptée pour le réseau régional, national mais aussi mondial.

Le Réseau LoRa se base sur le protocole LoRaWAN (Long Range Wide-area ou réseau étendu de longue portée) qui est peu énergivore.

LoRaWan vise les exigences clés de l'internet des objets IOT tels que les services de communication bi-directionnelle, la mobilité et la localisation sécurisée.

La communication entre les appareils terminaux et les passerelles est répartie sur les différents canaux de fréquence et sur des débits de données (en France 863 à 870 MHz MHz, au US 902 à 928 MHz et Chine 779 à 787 MHz).

Le réseau LoRa a un débit compris entre 0,3 à 50 kbps. Le débit et la puissance d'émission s'adaptent automatiquement selon les besoins des objets, afin de limiter la bande passante et la consommation d'énergie. Son utilisation est idéale pour des capteurs émettant périodiquement une faible quantité de données telle que la géolocalisation.

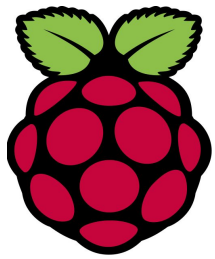
Contrairement à la wifi, la portée du réseau Lora est satisfaisante puisqu'elle peut être de 20 km en milieu rural et jusqu'à 3 km en zone urbaine (alors que la wifi possède une portée de 500 mètres maximum en extérieur).

Dans notre cas, nous souhaitons utiliser un protocole radio ayant la capacité d'être utilisé en tant que réseau privé, d'où le choix d'utiliser le protocole Lora plutôt que Lorawan ou encore Sigfox (utilisé uniquement en tant qu'opérateur).

e) La technologie SiRFatLas IV pour le GPS de la LoRa:

Le module GPS que nous utilisons pour le tracker LoRa utilise la technologie SiRFatLas IV. Le SiRFatLas IV est un processeur de système de localisation multifonction destiné aux dispositifs personnels de navigation d'entrée de gamme. Nous l'utilisons ici pour son prix intéressant tout en ayant de bonnes performances telles qu'une consommation de 41mA et une sensibilité de -163dBm (un GPS ultra-sensible est de -160dBm) permettant de capter les signaux des satellites même fortement atténués.

f) Raspberry Pi:



Le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM permettant l'exécution de plusieurs variantes du système d'exploitation GNU/Linux . Nous utilisons ce module pour obtenir une connexion internet au module LoRa récepteur afin d'envoyer les données GPS au serveur.

g) Serveur:

Pendant notre cursus à Polytech'Lille nous avons appris à installer et configurer des serveurs UNIX de façon sécurisée. Notre site web sera hébergé par un Ubuntu server maison grâce au logiciel apache HTTP server.

h) Serveur Node.js:

NodeJS permet d'exécuter du code javascript en dehors d'un navigateur web grâce à son moteur "**Chrome's V8 JavaScript**", maintenant très répandu dans le monde du développement côté serveur grâce à sa facilité de déploiement et d'implémentation.

Nous utilisons le javascript afin de faire le lien entre la base de données et les coordonnées GPS transmisent par nos trackers.

i) Site internet:

Afin de créer un site internet “responsive”, pour qu’il soit accessible et esthétique aussi bien sur standalone (ordinateur) que sur smartphone, nous utilisons bootstrap. Pour gérer les requêtes client-serveur, nous utilisons du PHP et AJAX. AJAX ayant pour but de récupérer les données de la requête php et de l’insérer dans notre page internet.

6) Matériels et coûts de productions

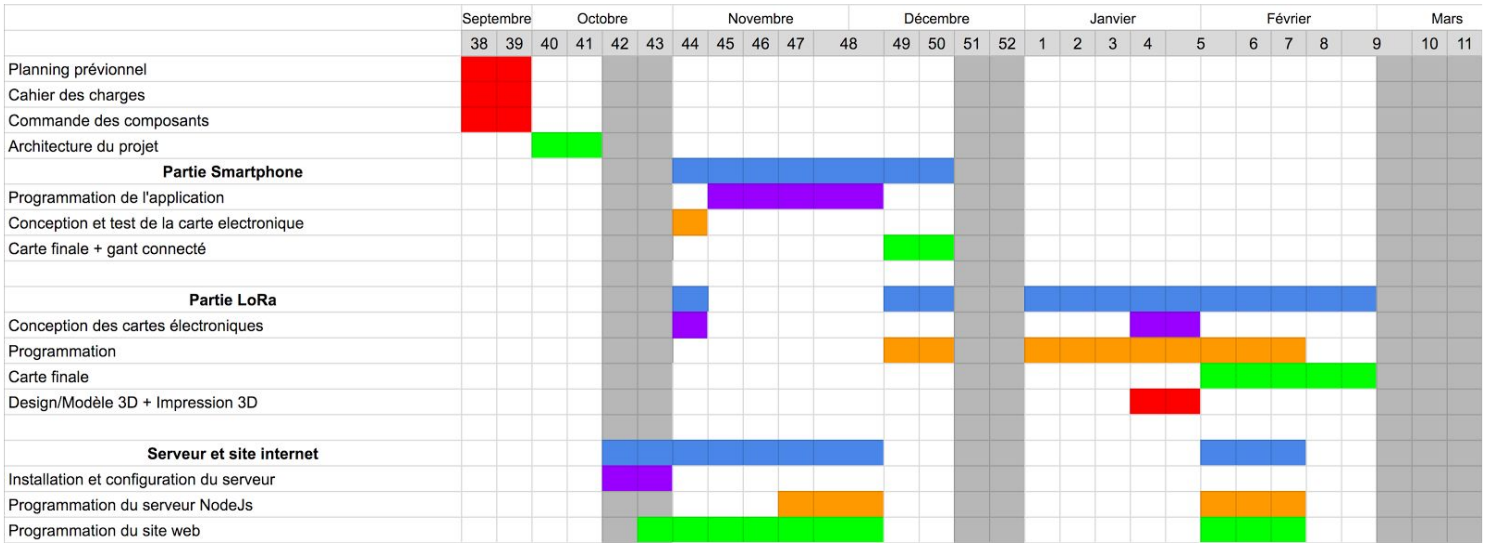
Matériels pour le Tracker Smartphone:

Matériels	**Coût (euro)**
1 Composant Lilypad Simblee	23.43
1 Smartphone possédant Android	X
1 Batterie Lithium 2200mah	15.99
Boutons poussoirs/Résistances/LED	2
TOTAL	41.42

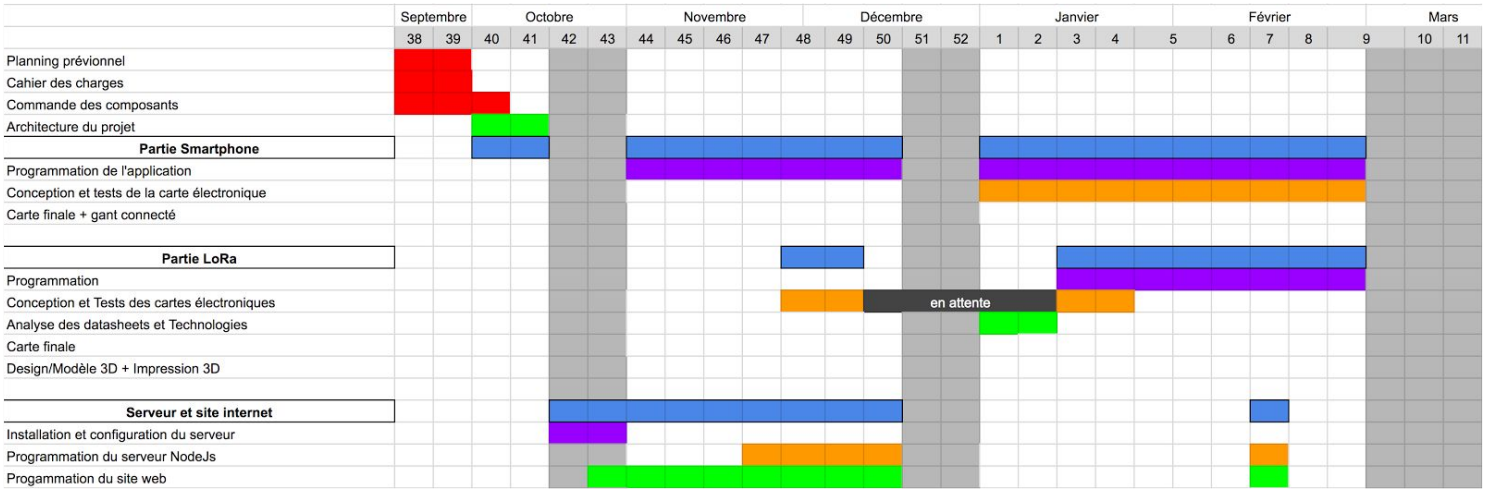
Matériels pour le Tracker LoRa:

Matériels	**Coût à l'unité (euro)**	**Nombre**	**Total**
module LoRa RN2483 (CMS)	12.1	2	24.2
module GPS A2200-A (CMS)	15.63	1	15.63
Batterie Lithium 2200mah	15.99	1	15.99
microcontrôleur Atmega 328p (CMS)	3.40	2	6.80
Connectiques USB, programmation ISP/Résistances/LED (CMS et traversant)	2	X	2
Raspberry pi 3	32	1	32
TOTAL			96.62

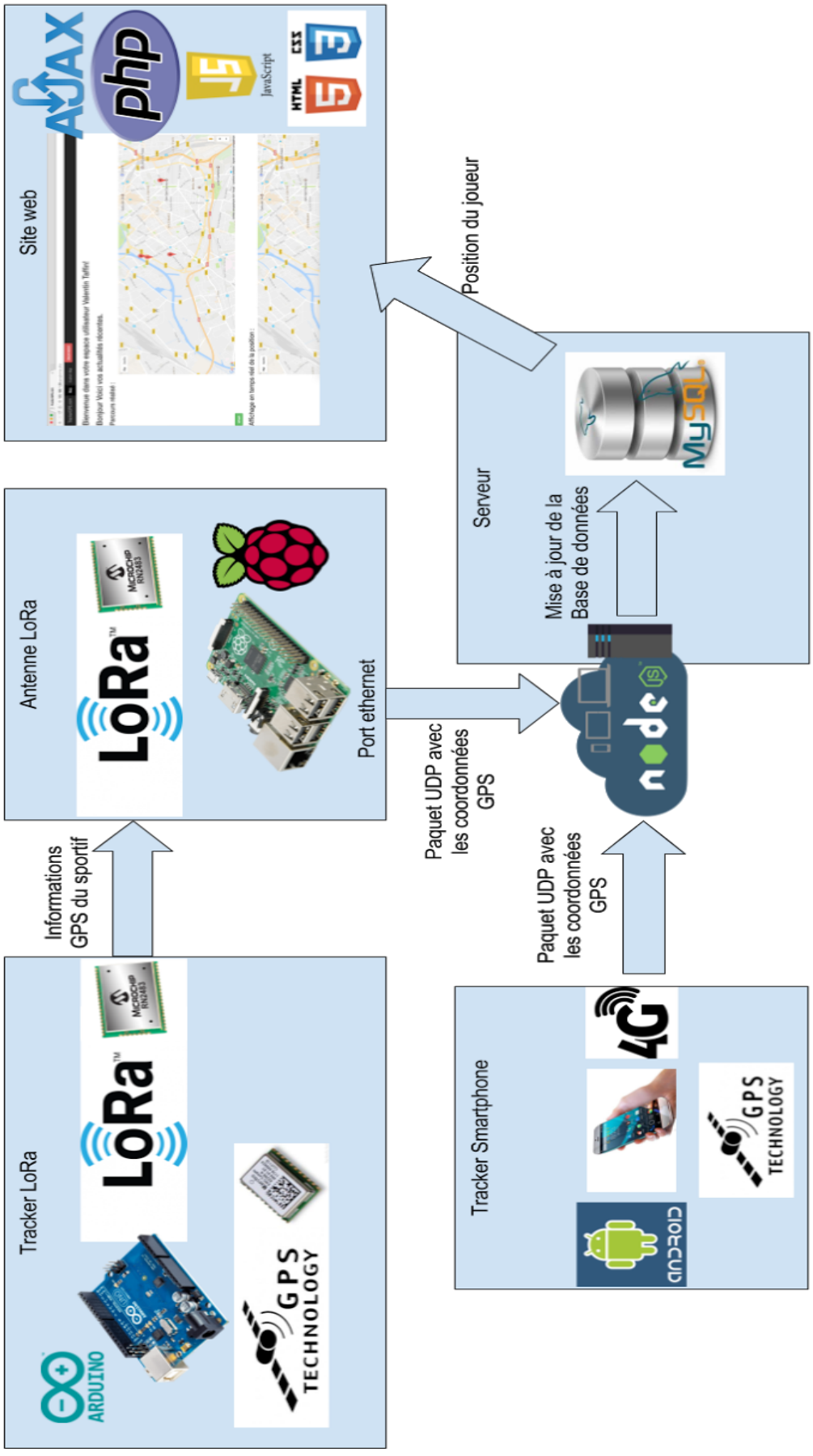
7) Planning prévisionnel



8) Planning officiel

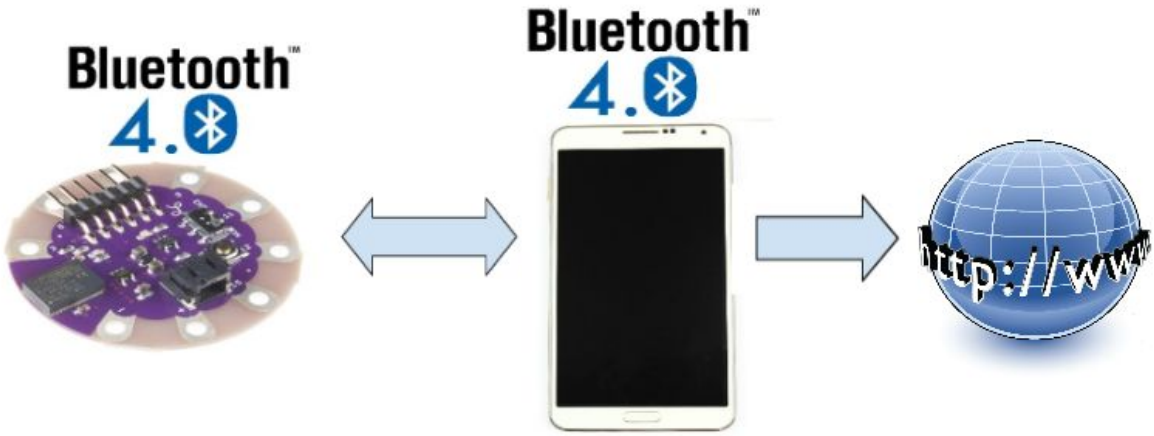


9) Vue d'ensemble du projet



II : Tracker Smartphone

1) Présentation



Le tracker Smartphone est composé de deux parties:

-un bracelet connecté
-une application android sur Smartphone

Le coureur sera équipé d'une carte électronique permettant d'envoyer des informations au Smartphone. L'application android permettra ensuite de récupérer les coordonnées GPS et de les envoyer à un serveur Web.

2) Objectif et Contraintes

Objectif:

La partie tracker Smartphone a besoin d'une application mobile pouvant recevoir et transmettre toutes les informations nécessaires.

Pour cette partie, nous avons utilisé :

-Le Bluetooth Low Energy entre la Lilypad Simblee et le Smartphone
-Le GPS interne du Smartphone
-La 4G du Smartphone

Le coureur sera équipé :

- d'une carte électronique contenant les boutons poussoirs où chacun d'eux correspond à un état du coureur (fatigue ou blessure) ou à un problème technique.
- d'une autre carte (Lilypad Simblee) permettant d'envoyer ces informations par Bluetooth Low Energy au Smartphone.

Contraintes:

L'alimentation du Smartphone peut difficilement être améliorée, malgré que le Bluetooth Low Energy consomme très peu de ressources, le GPS interne du téléphone ainsi que la 4G nécessitent une grande consommation d'énergie.

3) Descriptions des tâches réalisées

a) Partie Smartphone :

L'application mobile est composée :

-D'une classe permettant la récupération des données GPS

-D'une classe permettant d'envoyer les données GPS au Serveur

-D'une classe regroupant la partie BluetoothLE

Pour utiliser convenablement le logiciel, nous avons téléchargé le JDK au lien suivant :

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Le smartphone utilisé possède la version 5.1 d'Android (Lolipop), le projet a donc été configuré en fonction de ce SDK.

Configuration du téléphone :

Pour se configurer en mode développeur, nous devons aller dans:

- « Paramètre » du téléphone,
- aller dans sécurité => activer « source inconnue ».
- Dans « à propos de l'appareil », nous avons besoin de cliquer 7 fois sur le numéro de version du téléphone pour pouvoir passer en mode « développeur » .
- Pour finir, nous allons dans « option de développement » pour activer le débogage USB.

La classe Gps:

Nous avons utilisé plusieurs méthodes pour s'abonner et se désabonner du GPS pour que le programme ne puisse rencontrer aucun problème si celui-ci n'est pas disponible.

AbonnementGPS() - Méthode pour s'abonner à la localisation par gps.

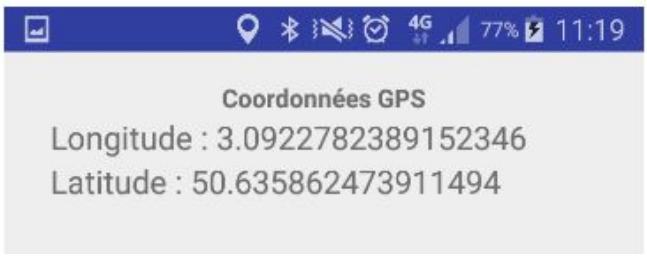
DesabonnementGPS() - méthode permettant de se désabonner de la localisation par GPS.

OnProviderEnabled() - si le gps est activé , on s'abonne.

OnProviderDisabled() - si le gps est désactivé, on se désabonne.

OnLocationChanged() - on affiche dans un message la nouvelle localisation (longitude et latitude).

Le programme demande au démarrage si le GPS est disponible, ensuite on s'y abonne et nous pouvons récupérer les données.



La classe ClientUDP:

Nous avons combiné la partie récupération des données GPS avec l'envoi des données au serveur. Pour cela nous avons créer un paquet contenant :

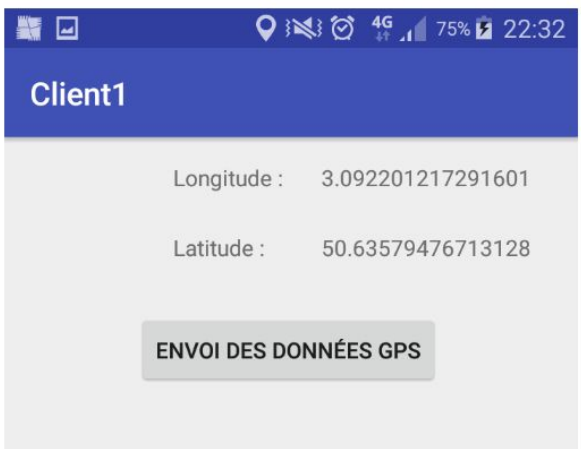
-le message que le serveur souhaite recevoir

-la taille du message

-l'adresse

-le port

Celui-ci sera ensuite envoyé au serveur. A chaque appui du bouton, on actualise puis on envoie les données GPS au serveur.



La classe BluetoothLE:

Pour concevoir cette partie, nous nous sommes d'abord intéressés à la partie Bluetooth classique, nous avons fait un programme permettant d'activer le Bluetooth, de repérer les dispositifs environnant disponibles afin d'avoir la possibilité de se connecter.

La Lilypad Simblee est utilisée en Bluetooth Low Energy (Bluetooth 4.0) qui produit très peu d'énergie pour fonctionner. Nous avons donc fait un programme permettant de :

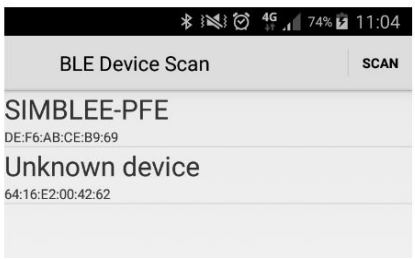
- Scanner les dispositifs environnant en vérifiant que ceux-ci supportent le BluetoothLE.
- Une fois la Lilypad repérée, nous nous connectons à celle-ci.
- Ensuite nous faisons une lecture des données envoyées par la Lilypad

Pour effectuer la lecture, nous avons construit un GATT qui permet à deux dispositifs BLE de transférer des données en appelant des services et des caractéristiques en utilisant des ID de 16 bits (Universal Unique Identifier).

Afin de lire correctement la Lilypad, nous nous sommes servis des UUID suivants :

-Service : 0000fe84-0000-1000-8000-00805f9b34fb

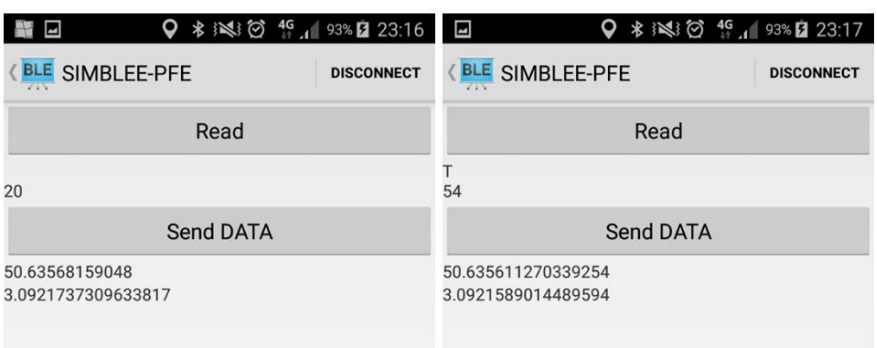
-Caractéristiques : 2d30c082-f39f-4ce6-923f-3484ea480596



Nous pouvons voir ci-dessus que notre Simblee a bien été trouvée, une fois connectés, nous pouvons utiliser les fonctions "Read" et "Send Data".

Le programme récupère bien les informations de la Lilypad tout en permettant d'envoyer les données sur le serveur.

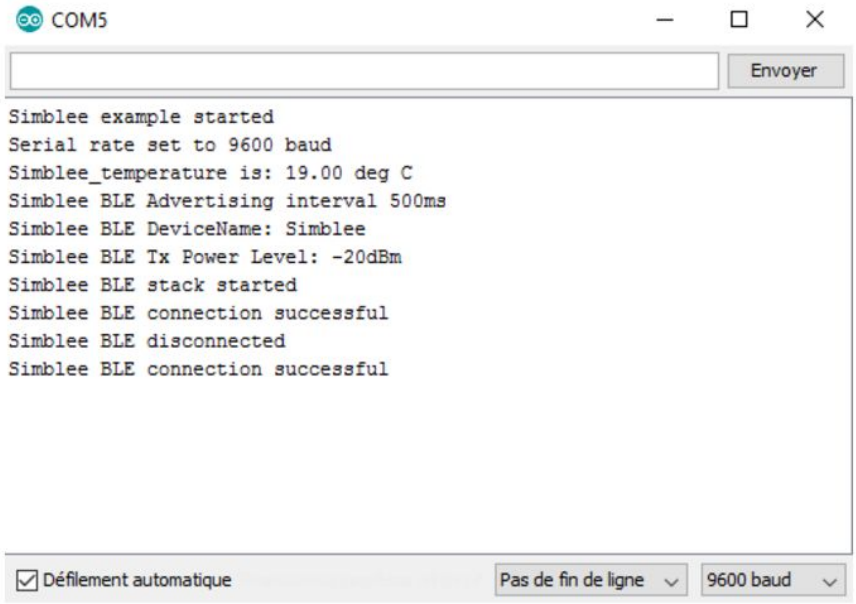
Nous pouvons voir ci-dessous les coordonnées du GPS ainsi que les données envoyées par la Lilypad. Ici, un « espace » ainsi que la lettre « T » ont bien été envoyés.



b) <u>Partie Lilypad Simblee :</u>

<u>Conception du programme de la Lilypad:</u>

Cette partie permet d'utiliser la Lilypad Simblee Ble pour envoyer les données souhaitées au smartphone. Pour cela, nous utilisons le logiciel Arduino IDE pour pouvoir récupérer le signal du bouton poussoir et l'envoyer par bluetooth avec la fonction "Simbleeble.send". Une fois que le bouton poussoir est enclenché, le programme envoie la donnée au mobile. Nous utilisons ici la librairie Simbleeble.h.
Ci-dessous nous pouvons vérifier si la connexion entre le smartphone et la Lilypad s'est bien effectuée en affichant dans le terminal de la Arduino si une connexion et une déconnexion est faite.



Une fois la Lilypad connectée au Smartphone, nous pouvons envoyer différents types de données à l'aide de boutons poussoirs. Le coureur aura par exemple le choix entre :

-ne rien envoyer

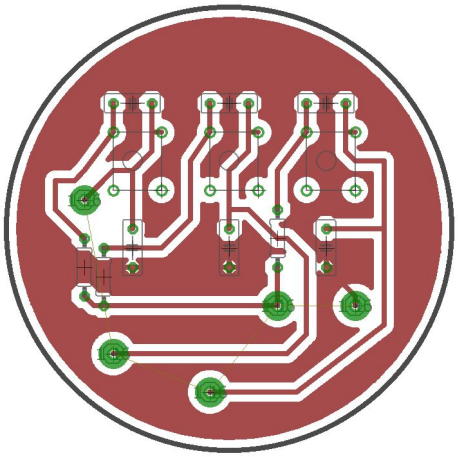
-envoyer un état de fatigue (caractère « F » envoyé)

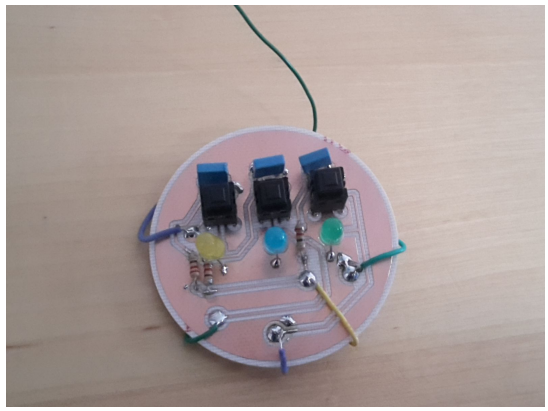
-envoyer un problème technique (caractère « T » envoyé)

-prévenir que le coureur est blessé (caractère "B" envoyé)

Conception de la carte électronique de la Lilypad:

Une fois le programme effectué, nous avons décidé de créer une carte électronique pour que le coureur puisse avoir accès sans encombrement aux trois boutons poussoirs avec les résistances et condensateurs nécessaires, ainsi que la batterie pour que le tout puisse être porté au bras.





Le circuit est donc assemblé avec la Lilypad et fonctionne en toute autonomie grâce à la batterie externe. La lilypad a une consommation en moyenne de 10 mA (réception,transmission et appui des boutons poussoirs) et donc avec une batterie d'une capacité de 2200mah, nous pouvons obtenir une autonomie de 220 heures.

L'idée était de faire tenir la carte électronique dans la main de l'utilisateur pour plus de confort. Malheureusement, la carte électronique prenait trop de place. Nous avons alors pensé à faire une carte plus accessible.

Le schématique et PCB en **annexe 1 page 30** est la version finale de la carte électronique des boutons poussoirs avec des dimensions plus petites (20mmx30mm au lieu de 60mm de diamètre):

Finalisation du Tracker Smartphone :

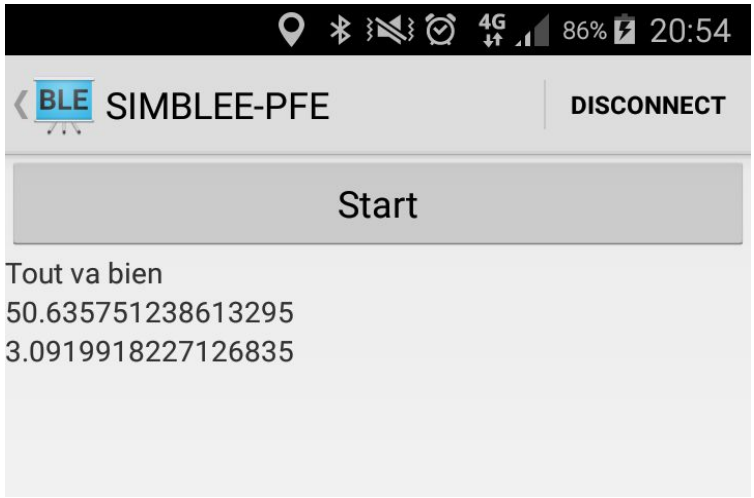
Pour conclure la partie Tracker Smartphone, le coureur peut utiliser le programme sur son smartphone pour :

-faire une recherche et se connecter à la Lilypad.

-activer la récupération de données de la Lilypad pour que celle-ci puisse envoyer les informations souhaitées à l'aide des boutons poussoirs.

-le programme du Smartphone envoie ensuite les coordonnées GPS ainsi que les informations de la Lilypad sur le serveur toutes les secondes avec le bouton “Start”.

Ci-dessous le test de l’application finale :



4) Caractéristiques et performance

Avantages:

- faible consommation de la Lilypad en envoyant l’état du coureur toutes les secondes au serveur.
- L’envoi des informations grâce à la Lilypad est simplifié en évitant d’utiliser son smartphone à la main.

Inconvénients:

- Les technologies du Smartphone (GPS et 4G) consomment le plus.
- L’envoi des données GPS au serveur fonctionne seulement si la Lilypad est connectée au Smartphone.

5) Améliorations possibles

- Amélioration du code et du design de l’application du Smartphone.
- Créer deux modes de fonctionnement avec et sans le gant connecté.
- Ajout d’un écran LCD sur la Lilypad pour vérifier la connexion entre le serveur et le Smartphone.
- Utiliser une batterie plus petite et réduire la taille du composant électronique installé sur le bras.
- Faire un boitier de petite taille pour pouvoir le transporter et l’utiliser facilement.
- Créer notre propre PCB du module Bluetooth Low Energy pour réduire la taille.

III : Tracker LoRa

1) Présentation

La partie tracker LoRa consiste à réaliser un tracker GPS contenu dans un petit boitier afin que le coureur puisse le transporter facilement durant ses activités sportives ou autres.

2) Objectif et contraintes

Objectif :
Le but de cette partie est de réaliser un tracker GPS utilisant les technologies de l'internet des objets :

- Pour la transmission des données on utilisera le réseau bas débit LoRa.
- La technologie GPS (Global Positioning System).

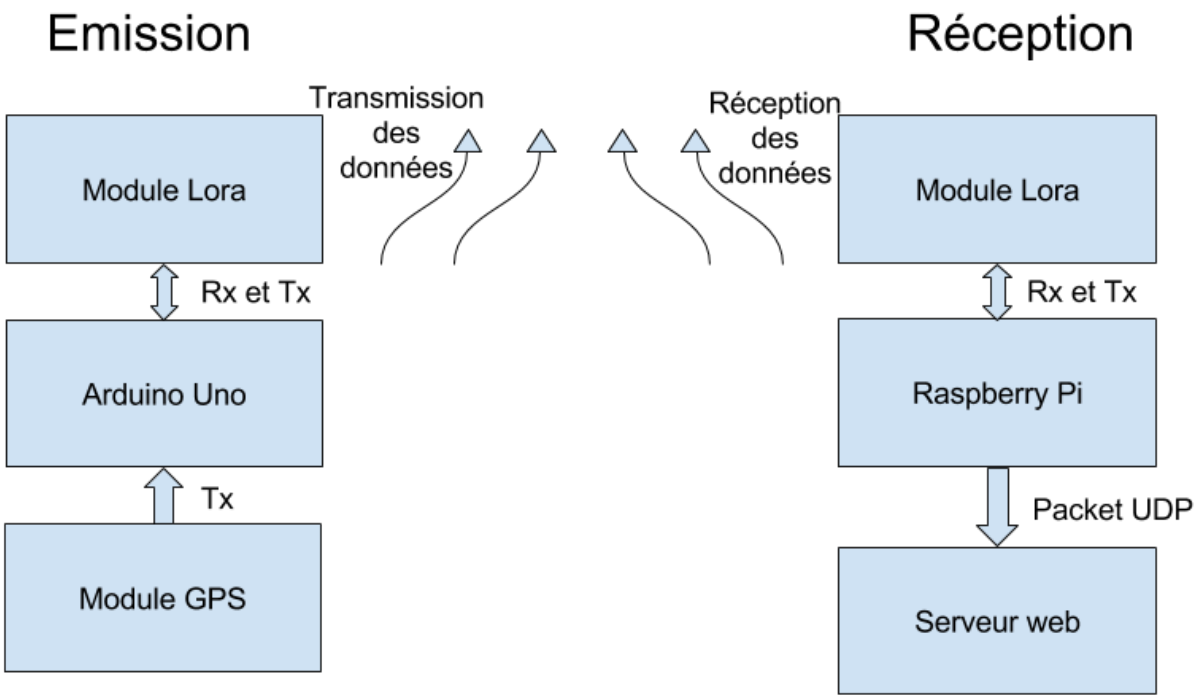
Contraintes :

La course est un sport extérieur, ce qui signifie que nous devons étudier et vérifier que le matériel utilisé est capable de résister à la température, l'humidité, les intempéries et choc.

Comme le Tracker Smartphone, l'autonomie est une caractéristique importante de notre système, il faudra que celui-ci ait une autonomie de plusieurs semaines afin que le coureur puisse réaliser ses courses en toute tranquillité.

3) Description

Le tracker LoRa correspond à la partie émission du schéma ci-dessous et la partie réception permet de relayer les données transmises sur le serveur web afin d'être visualisées sur le site internet.
Dans cette section, nous expliquerons la conception des cartes électroniques, l'implémentation de chacun des modules LoRa et GPS sur la Arduino Uno. Puis de la partie réception composée d'une Raspberry Pi et d'un module LoRa.



Conception des cartes électroniques de test :

Nous avons décidé de concevoir 3 cartes électroniques afin de tester chaque module. Au lieu de rooter une carte globale nous avons décidé de créer un PCB pour chacun des modules LoRa et une pour le module GPS. La raison de ce choix fut très simple, au début du projet nous ne possédions pas les compétences suffisantes pour réaliser une carte PCB qui serait alors, composée d'un module LoRa et d'un module GPS communiquant tous deux à une Atmega328p programmable par ISP et d'une batterie Lithium à ion rechargeable.

Chaque carte électronique est alors une carte de prototypage, c'est pour cela que nous utilisons une Arduino Uno afin de profiter d'un environnement de développement facile d'utilisation. Nous avons fait ressortir toutes les pins importantes, VCC , GND, Tx, Rx ainsi que le Reset de chacun des composants.

Les schématiques et PCBs de nos cartes sont disponibles à l'**annexe 2 page 31 et à l'annexe 3 page 32**.

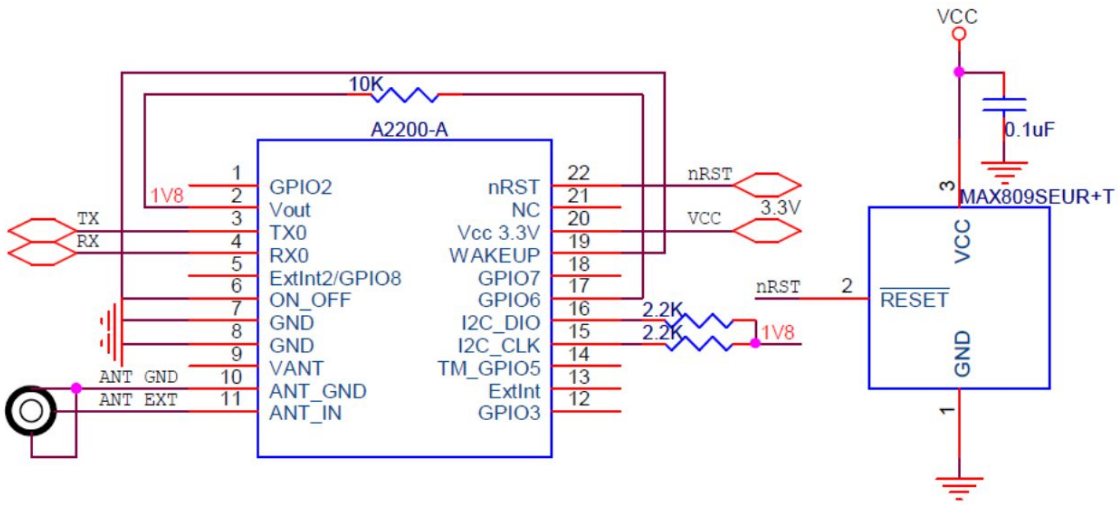
Implémentation du module GPS A2200-A avec la Arduino Uno *(Voir annexe 4)* :

Le composant A2200-A est un GPS bas de gamme développé par Maestro.

Voici les caractéristiques du module GPS:

- Dimension : 10.2 mm x 14 mm
- Fréquence : 1,575 MHz
- Sensibilité : -163dBm
- Résistance à des températures de -40°C à +85°C
- Bauds rate : 4800 / 9600 / 38400
- Protocole supporté NMEA et OSP
- Précision GPS : 2.5 mètres

Ci-dessous le montage utilisé pour réaliser les tests **(ne pas prendre en compte le MAX809)** :



Une fois la carte réalisée, il nous faut trouver la longueur des antennes, le GPS possède deux ports antennes ANT_GND et ANT_IN, elles doivent être de la même longueur et être positionnées sur le même plan tout en étant parallèle.

D'après le calcul de la longueur de la longueur d'onde :

$\lambda = \frac{v}{f}$ avec :
λ : longueur d'onde en mètres,
v : vitesse de déplacement en m/s
f : fréquence en hertz.

Nous obtenons une longueur d'onde de 19.0476 cm on choisit une longueur d'antenne **λ/2** soit 9.5238 cm.

Le montage ci dessus permet un démarrage automatique du GPS grâce à la liaison entre les pins ON_OFF et WAKEUP. On peut choisir différents modes pour le gps en envoyant des impulsions de 200 ms sur la pin ON_OFF, il possède 3 modes :

- Hibernate Mode est un mode basse consommation de 20 uA.
- Push-to-Fix est un mode développé pour les applications ayant pour but de transmettre des positions non fréquentes. Le récepteur reste généralement en basse consommation pendant 2 heures mais se réveille périodiquement pour mettre à jour sa position, le temps, les données éphémères et le RTC calibration.
- SiRFaware recherche à maintenir un niveau d'incertitude en position, temps et fréquence et permet de maintenir le courant valide Ephemeris utiliser pour la réception des données transmise par les satellites en vue.

Après s'être synchronisé aux satellites (au minimum 3 pour la triangularisation afin de connaître la position du module GPS sur la Terre), le module code ses informations transmises via protocole NMEA : The **N**ational **M**arine **E**lectronics **A**ssociation a développé une norme qui est une interface pour les équipements électroniques de la marine. Ce standard permet d'envoyer des informations aux ordinateurs et à d'autres équipements de la marine, on le nomme le protocole NMEA.

Le protocole NMEA possède plusieurs trames possibles, notre GPS utilise les trames suivantes :

- GGA : Donne notre position dans l'espace 3D.
- GSV : Montre les satellites en vue et affiche l'information sur le satellite capable de trouver notre position en utilisant son masque de vue et la donnée almanac. Il peut également affiché l'habileté à suivre la donnée.
- GSA : La précision du GPS **DOP** (**D**ilution **O**f **P**recision) et les satellites actifs.
- RMC : NMEA a sa propre version sur la position, la vélocité et le temps, le sigle RMC signifie "The **R**ecommended **M**inimum sentence **C**"

voici les exemples de trames que nous transmet le GPS :

```
$GPGSA,A,3,24,25,32,19,14,06,17,15,,,,,1.8,0.9,1.6*3C
$GPRMC,1623,E,1,09,0.9,36.1,M,47.2,M,,0000*6C
$GPGSA,A,3,12,24,25,32,19,14,06,17,15,,,,1.5,0.9,1.3*37
$GPGSV,3,1,12,12,86,293,08,24,62,122,15,25,41,251,22,32,38,298,23*7E
$GPGSV,3,2,12,19,29,048,30,14,20,317,11,06,15,081,32,17,14,037,21*7B
$GPGSV,3,3,12,15,09,174,10,02,09,117,18,10,05,261,,29,02,195,*74
$GPRMC,165008.000,A,5036.4574,N,00308.1623,E,0.00,9.69,220217,,,A*69
$GPGGA,165009.000,5036.4574,N,00308.1623,E,1,09,0.9,36.0,M,47.2,M,,0000*6C
```

Nous utilisons la trame GGA afin de récupérer les coordonnées GPS et autres informations relatives :

```
trame GGA
heure 104614
latitude Nord 5036.4406
longitude Est 00308.160
Fix quality 1
nombre de satellites 04
Horizontal dilution of position 1.9
altitude 1.4M
envoi message lora
```

Pour récupérer les données satellites du GPS nous devons lire la pin **“Rx”** de la Arduino.

Implémentation du module LoRa RN2483 avec la Arduino Uno *(Voir annexe 4)* :

Afin de prototyper notre antenne Lora nous utilisons une carte arduino munie d'une Atmega328p, ce qui nous permet d'utiliser directement l'IDE de développement Arduino et de profiter de leurs librairies.

Voici les caractéristiques du modules LoRa :

- Dimension : 17.8 x 26.7 x 3 mm
- Bande de fréquence : 863 MHz à 870 MHz ou 433.050MHz à 434.790 MHz
- sensibilité : -148 dBm
- Vitesse de transmission de donnée dans les airs : 300kbps par FSK ou 10937bps avec la technologie de modulation LoRa
- Il possède une interface UART
- Portée de 15 Km en campagne et 5 Km dans les zones urbaines.
- Résistance à des températures de -40°C à +85°C
- Tension d'alimentation 3.3 Volts

A la manière du module GPS, il faut calculer la longueur de l'antenne pour la LoRa. Nous obtenons une longueur d'antenne **λ/4** de 8.6906 cm.

Tout d'abord le module LoRa RN2483 fonctionne par envoi de commandes spécifiques car il nous faut synchroniser la partie émission et réception.

Voici la commande pour connaître la version du module LoRa R2483 : **"sys get ver"**.

On doit synchroniser les 2 modules ensemble en choisissant la bonne fréquence parmi celles imposées **"radio set freq 868100000"** exprimée en hertz puis choisir la puissance du signal d'émission **"radio set pwr 14"**, elle peut être comprise entre -3 et 15, réglée à 14 cela revient à une puissance 13.5 dBm et 38.0 mA.

A chaque commande le module renverra une réponse comme **"ok"**, **"busy"** ou **"invalid_command"**.

Une fois la LoRa initialisée il suffit d'utiliser la commande **"radio tx message"** pour envoyer nos données, mais il ne faut pas oublier que le message doit être transmis en hexadécimal car la LoRa ne peut envoyer les caractères de 0 à 9 et A à F.

La trame type que nous envoyons : 0004A30B001BE76,162344,5036.4552,00308.165

- 0004A30B001BE76 : correspond à l'identifiant constructeur de la LoRa
- 162344 : l'heure, ici 16 heures 23 minutes et 44 seconds
- 5036.4552 : la latitude soit 50°Nord '36.4552
- 00308.165 : la longitude soit 3°Est '8.165

Voici à quoi ressemble la trame plus haut une fois codée en hexadécimal :

"*30303034413330423030314245373 62C3136323334342C353033362E343535322C30303330382E313635*"

Une fois la trame convertie en hexadécimal celle-ci est envoyée au module LoRa pour être transmis au module LoRa de la raspberry Pi. Grâce à la commande **"radio rx 0"** qui signifie que le module écoute sur le channel 0 continuellement, si celui-ci reçoit le message plus haut la réponse sera : **"radio_rx message_reçu"**.

Le schématique et PCB utilisés pour les tests sont à l'**annexe 6 page 34 et 35**.

Implémentation du module LoRa RN2483 avec la Raspberry pi *(Voir annexe 5)*:

Le but de cette partie est de recevoir les données codées en hexadécimal correspondant aux informations sur les coordonnées GPS transmises par le module LoRa de la Arduino.

La partie programmation de la raspberry pi est réalisée en Python pour des raisons de simplicité.

La raspberry pi doit-être accessible depuis internet afin de rediriger les informations du tracker sur le serveur web, ce qui implique que l'on doit faire les modifications nécessaires afin de la sécuriser comme changer le nom d'utilisateur et les mots de passe. Ce que nous n'avons pas fait la première fois, elle s'est donc fait hacker mais sans gravité car nous avons pu supprimer le programme que l'intrus nous a laissé qui consistait à se connecter sur des sites "malveillants" afin d'augmenter leurs vues et gagner de l'argent. Une fois reconnecté il a fallu retirer son adresse ip sur les serveurs de spam, on appelle cela les listes de blacklistage.

Anecdote faites, on utilise le langage python pour synchroniser le module LoRa en réception sur la raspberry pi avec celui de la Arduino configuré en transmission. Quand nous recevons les données du tracker LoRa, codé alors en degré Nord et minutes on les décode puis on convertit les coordonnées gps en décimal. Une fois les données décodées et mises au format voulu, un paquet est envoyé sur le serveur. Cependant, par la suite, pour optimiser la conversion des positions GPS en décimal elles devront être faites sur le serveur.

La donnée reçue codée en hexadécimal est :

"*3030303441333042303031424537362C3136323334342C353033362E343535322C30303330382E313635*"

Puis décodée :

idLora	heure	latitude	longitude

```
['0004A30B001BE76', '180015', '5036.4586', '00308.157']
UpdateGpsPositionLoRa 0004A30B001BE76 50.607643 3.13595
radio rx 0
ok
```

4) Caractéristiques et performances

En utilisant une batterie de 2ah nous obtenons sans optimisation particulière une autonomie d'environ 40 heures sachant que contrairement aux appareils sur le marché, nous envoyons la position du coureur constamment car grâce au réseau LoRa, nous ne sommes pas limités par le nombre de paquets que l'on peut envoyer dans la journée (140 fois par jours pour Sigfox).
Notre module restera petit et facilement transportable dans une poche ou sur un porte-clef (60 x 40 x 2 mm).
Il possède les caractéristiques comme la résistance à la chaleur et au froid (-20°C / + 60°C).
Le module transmet sa position GPS entre 1 à 5 fois par seconde, ramener à 1 fois par seconde en moyenne afin de gagner en autonomie.

5) Améliorations possibles

- Taille réduite du tracker.
- Optimiser la consommation du tracker en jouant sur l'hibernate mode du module LoRa et GPS en complexifiant le schématique et la programmation.
- Créer une antenne PCB pour chaque module afin d'optimiser la place.
- Ajouter des boutons sur le tracker LoRa afin de connaître l'état de fatigue du sportif ou transmettre des informations diverses.
- Connaître l'état de la batterie.
- Ne plus utiliser une raspberry pi mais se connecter à internet et utiliser le réseau LoRaWan.
- Créer le PCB final avec recharge d'une batterie Lithium à ion et regroupant tous les composants nécessaires (GPS A2200-A, RN2483,Atmega328p etc).
- Création d'un boitier permettant de transporter facilement le tracker, résistant à des chocs mineurs.
- Coder essentiellement en C et assembleur.
- Trouver un microcontrôleur de plus faible consommation et plus petit.

IV : Serveur et site Internet

1) Présentation

Le but est de créer un espace utilisateur pour le sportif afin qu'il puisse visualiser son parcours et connaître sa position en temps réel via un site web.

Cette partie décrit comment le serveur fonctionne en récupérant les données à l'aide d'un serveur NodeJS transmises par les différents trackers.

2) Objectifs et contraintes

Objectifs :

- Créer un site internet composé d'un espace utilisateur pour le sportif
- Visualiser ses positions GPS en temps réel sur une carte

Contraintes :

- Accessibilité et navigation intuitive depuis un smartphone
- Gestion de plusieurs requêtes simultanément
- Sécurité du serveur
- Synchronisation du projet à l'aide de l'utilitaire GIT

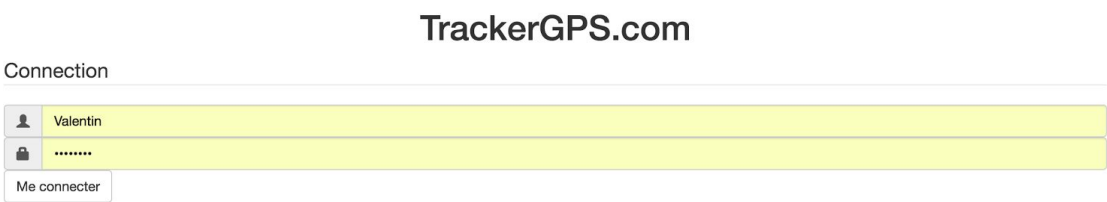
3) Description

Dans cette partie sera décrit comment fonctionne le site web et les scripts appelés durant la navigation (sous forme de state machine).

a) Le Site Web:

Le site web est réalisé en PHP afin de récupérer les données contenues dans la base de données. Les technologies telles que le Javascript, AJAX, Json, Google API sont utilisées afin de visualiser la position de notre coureur en temps réel.

Ci-dessous la page permettant de se connecter afin d'accéder à son compte utilisateur.



Une fois connecté, il y a deux cartes :

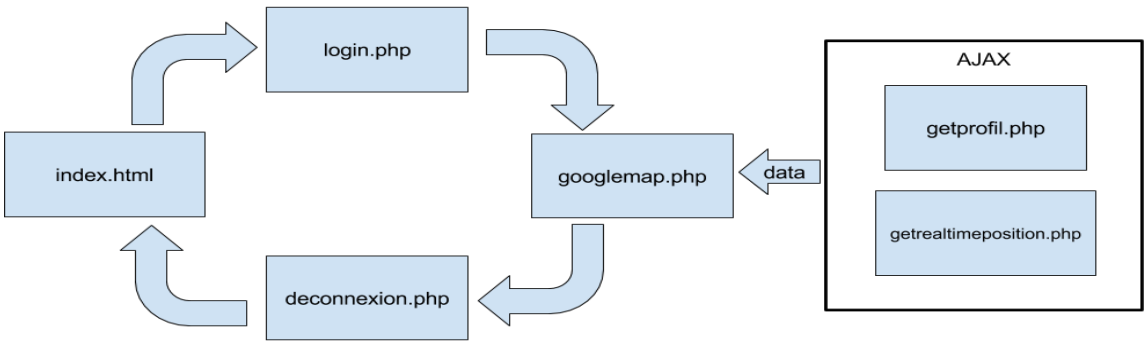
- Celle destinée à l'affichage des itinéraires

- Et celle destinée à l’affichage de la position du coureur en temps réel

Voici le déroulement d’une session client. Tout d’abord le client accède à la page d'accueil du site web “index.html” pour se connecter en entrant ses identifiants utilisateur, le script login.php vérifie que l’utilisateur existe dans la base de données. Puis nous accédons au coeur du site “googlemap.php”, qui gère l’affichage du parcours réalisé de notre sportif et une carte affichant sa position courante.

Les scripts ajax getprofil.php et getrealtimeposition.php, permettent de réaliser les requêtes nécessaires sur la base de données.

Une fois terminé, l’utilisateur peut se déconnecter à l’aide d’un bouton déconnexion.



b) La Base de Données:

La base de données est réalisée avec mySQL, Le schéma ci-dessous n'est qu'une ébauche de notre base de données et sera mis à jour suivant l'avancement de notre application. Celle-ci est découpée en plusieurs parties :

- "profil" contient toutes les informations relatives au compte client.
- "point" qui correspond aux coordonnées de géolocalisation GPS.
- "itinéraire" composé d'un ensemble de points afin d'enregistrer des parcours prédéfinis.
- “profil_itineraire” permettant d’ajouter des itinéraires à nos profils.
- "positiongps", la position GPS courante de notre sportif.

c) La récupération des coordonnées GPS des différents trackers:

Le serveur d’écoute des paquets UDP est réalisé en nodeJS, son but est de se connecter à notre base de données afin de l’actualiser suivant les paquets UDP transmis par nos trackers.
Chacun des trackers possèdent une norme à respecter afin d’être traité normalement.
Voici la donnée que nous devons recevoir pour le tracker smartphone :

- “UpdateGpsPositionLoRa lastname firstename latitude longitude state”

Pour le tracker LoRa :

- “UpdateGpsPositionLoRa idLora latitude longitude”

Chaque donnée contenue dans le paquet UDP est séparée par un espace afin de faciliter le traitement.

Pourquoi réaliser les trames UDP ?

- La simplicité d’implémentation, en TCP il aurait fallu gérer l’accusé de réception
- Nous envoyons les coordonnées GPS régulièrement (toutes les 1 à 10 sec cela dépend des satellites), si l’on perd 10% des paquets transmis pour des raisons inconnues, le sportif ne verrait pas la différence étant donné qu’il se déplace à une vitesse comprise entre 10 et 50 km/h.
- Ce n’est pas le cas du réseau LoRa mais le réseau Qowisio est unidirectionnel, si l’on veut implémenter notre système sur Qowisio on le pourrait aisément. Quant au réseau SIGFOX on ne peut envoyer que 140 messages par jour soit un envoi des coordonnées GPS toutes les 10 minutes, si l’on commence à implémenter les accusés de réception on perdrait des messages avec la synchronisation du serveur

et de l’appareil IOT. De plus l’idée de l’IOT est que le dispositif connecté n’envoie que ses données capteurs sans se préoccuper du serveur (server-side).
- Gain en consommation d’énergie.

Problème :

- On ne peut pas vérifier que le tracker est bien connecté au serveur.

4) Caractéristiques et performances

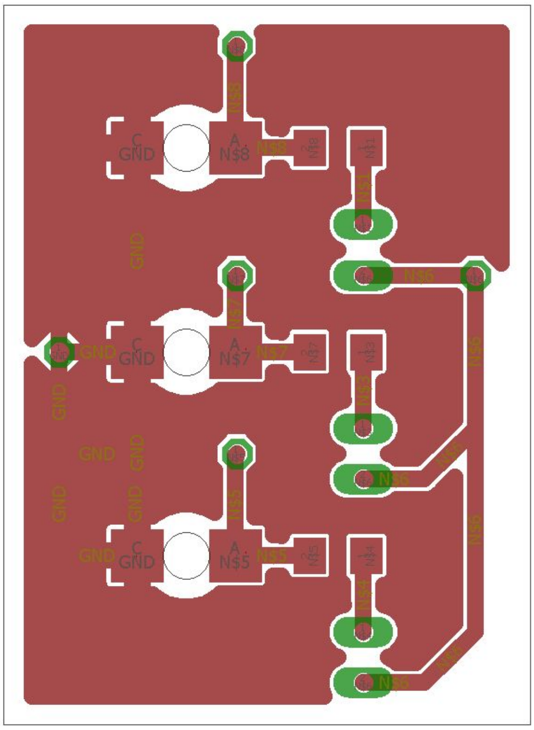
- Le site web peut gérer plusieurs utilisateurs en même temps.
- Il est “responsive” c’est à dire qu’il est accessible aussi bien depuis un appareil mobile que sur un ordinateur fixe.

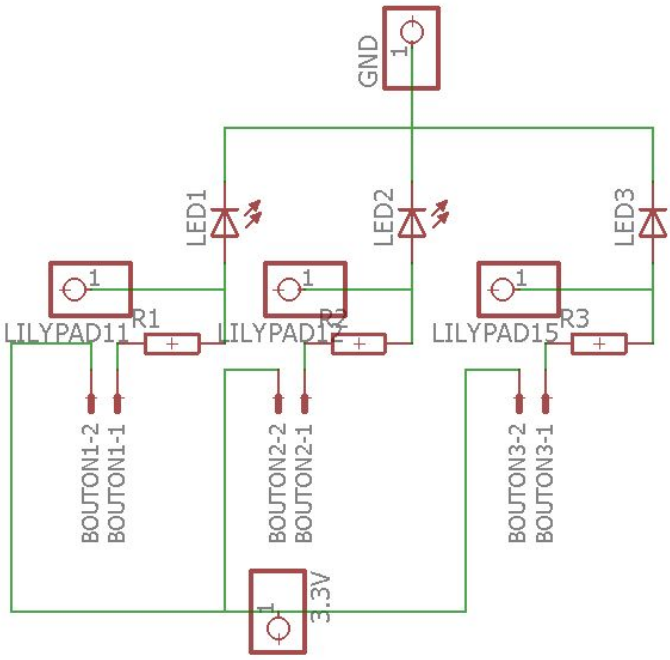
5) Améliorations possibles

- Ajout d’un formulaire pour s’inscrire avec protection anti-robot.
- Utiliser une protection à l’aide de captcha lors de l’authentification.
- Ajouter un espace de customisation des parcours, afin d’éditer et de créer ses propres itinéraires.
- Calculer les performances réalisées par le sportif, les comparer à ses sessions précédentes.
- Ajouter la création d’événements où plusieurs sportifs participeront .
- Ajouter un espace “Amis” afin d’être connecté et discuter à plusieurs.
- Ajouter un espace Workout permettant de planifier son entrainement durant la semaine.
- Sécuriser les trames UDP.

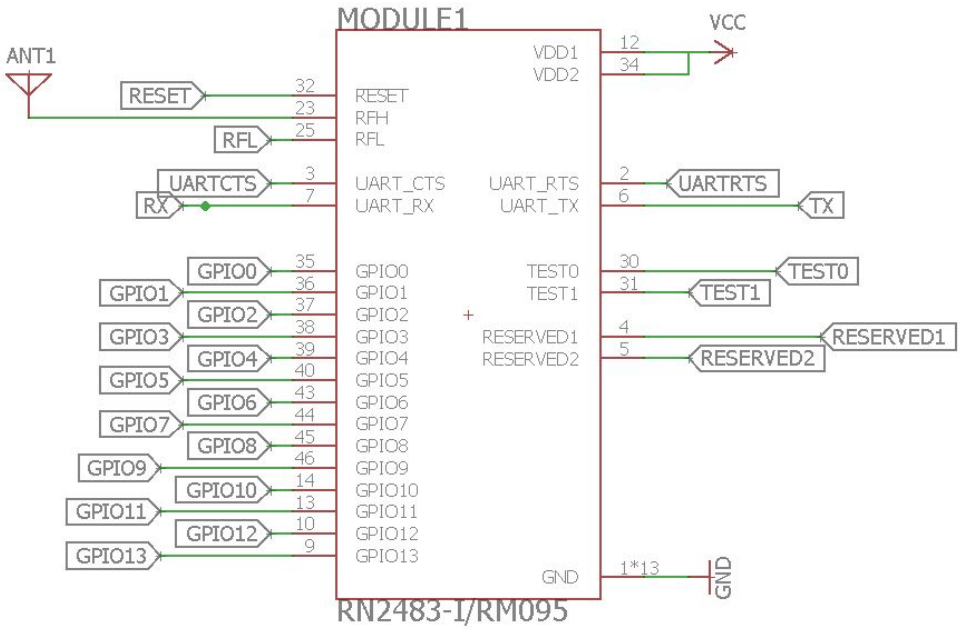
V : Annexes

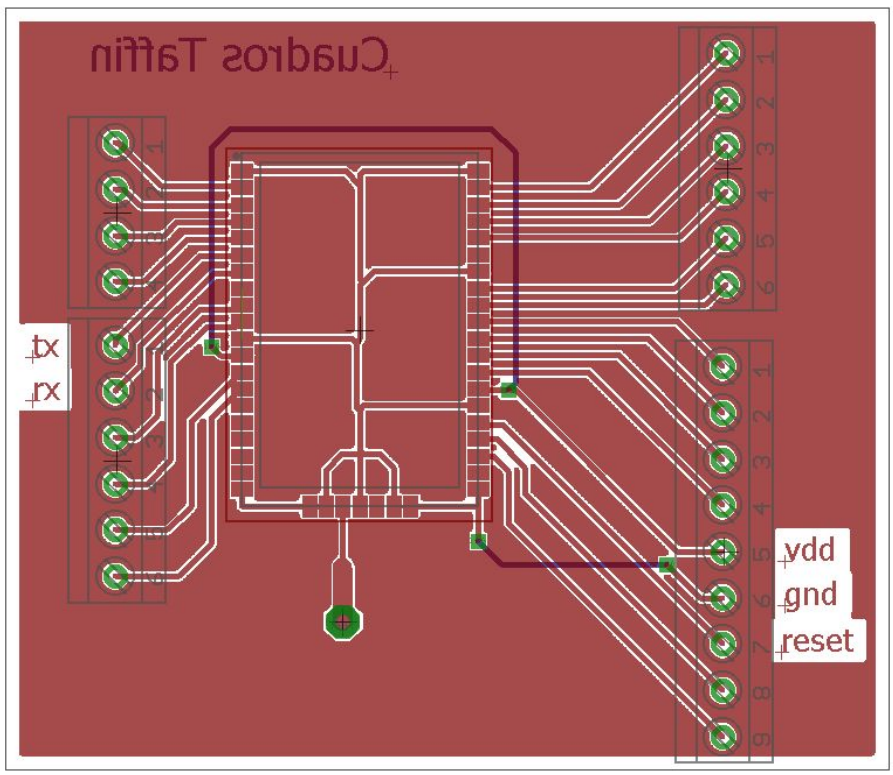
Annexe 1 : Version finale de la carte électronique pour la Lilypad :



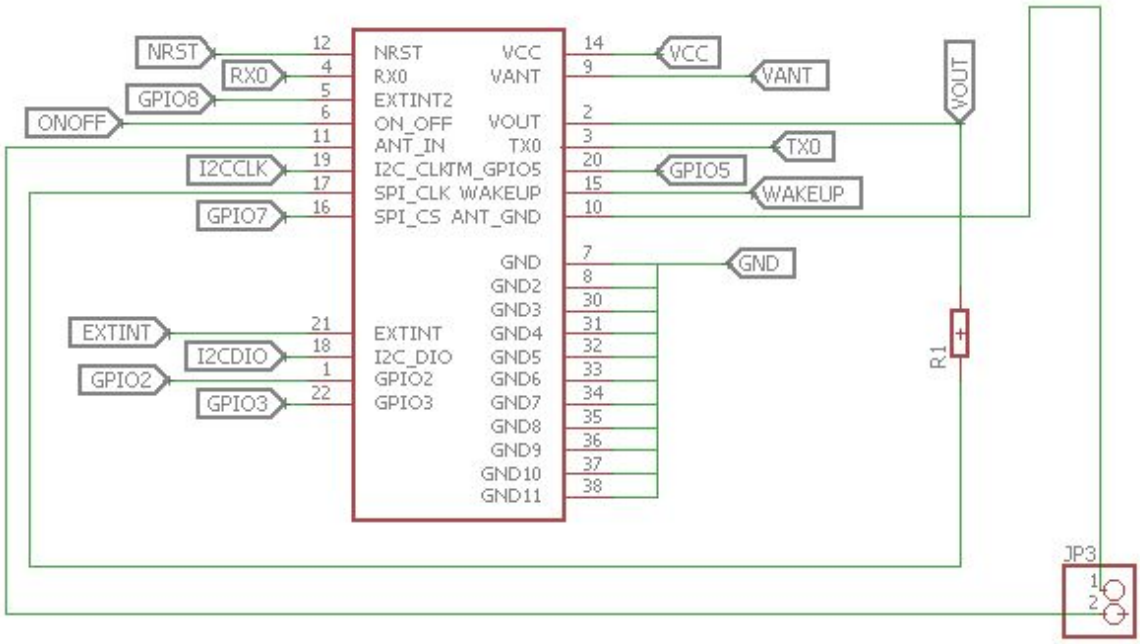


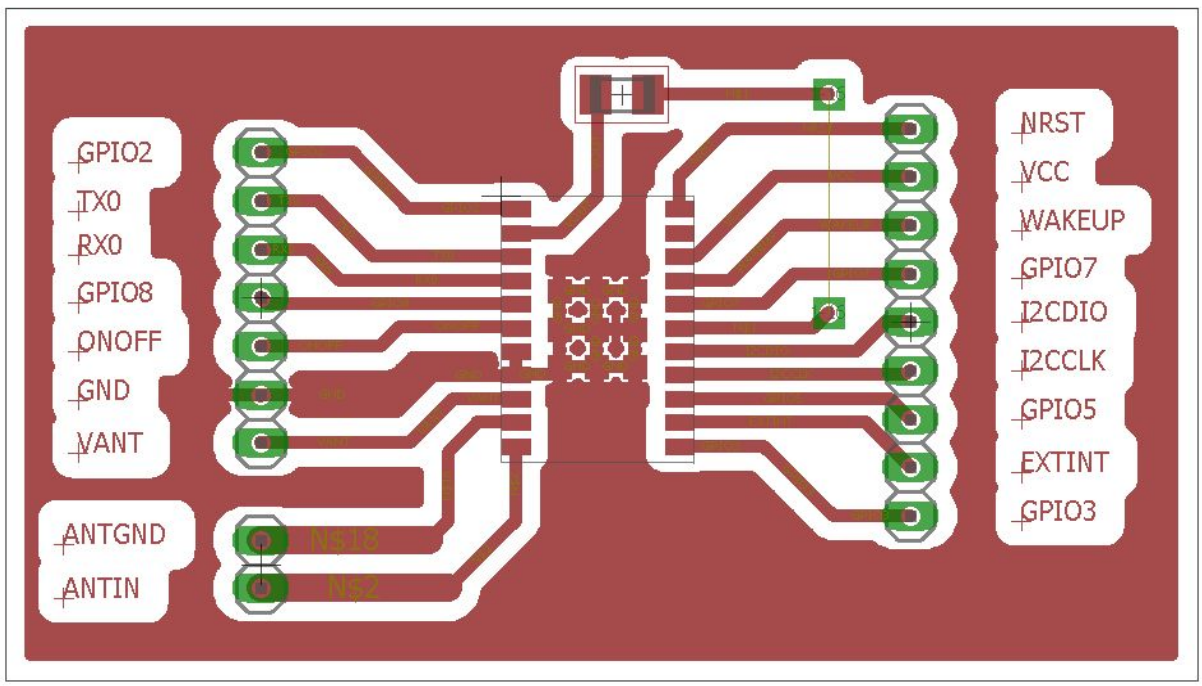
Annexe 2 : Schématique et PCB test du module Lora



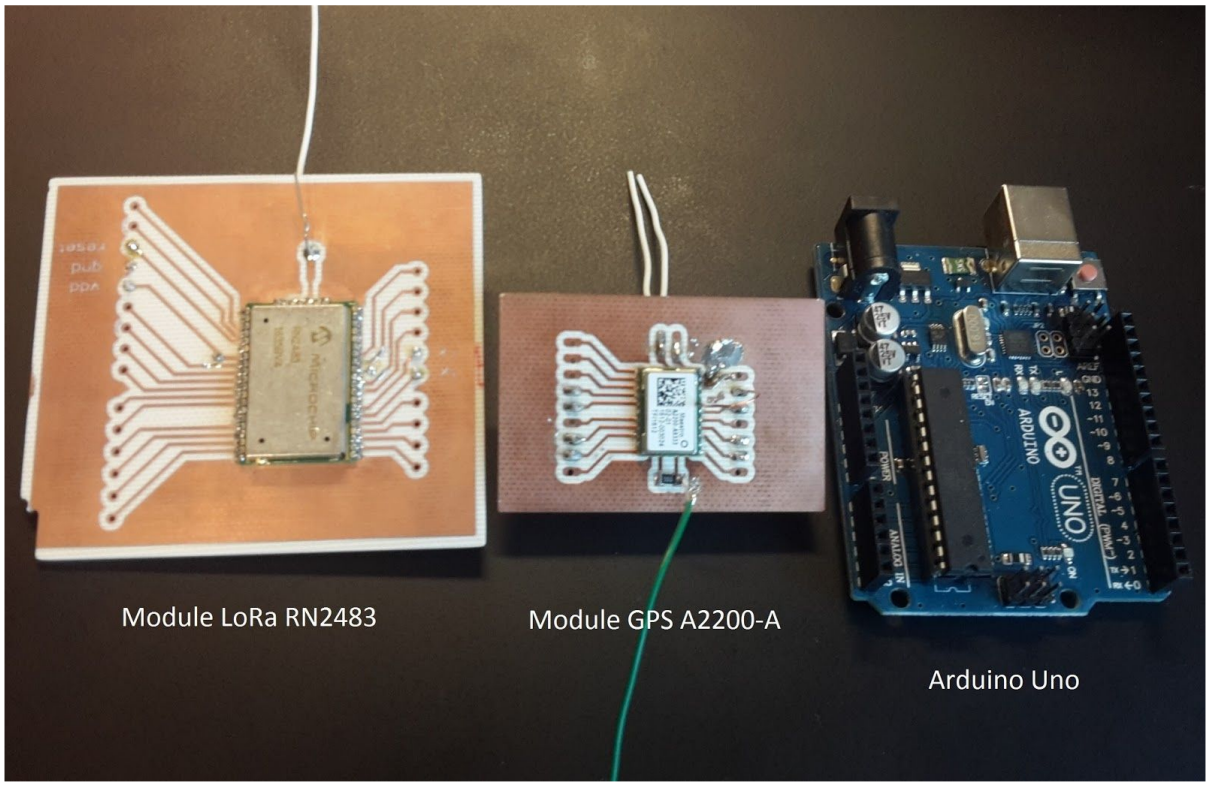


Annexe 3 : Schématique et PCB test du module GPS

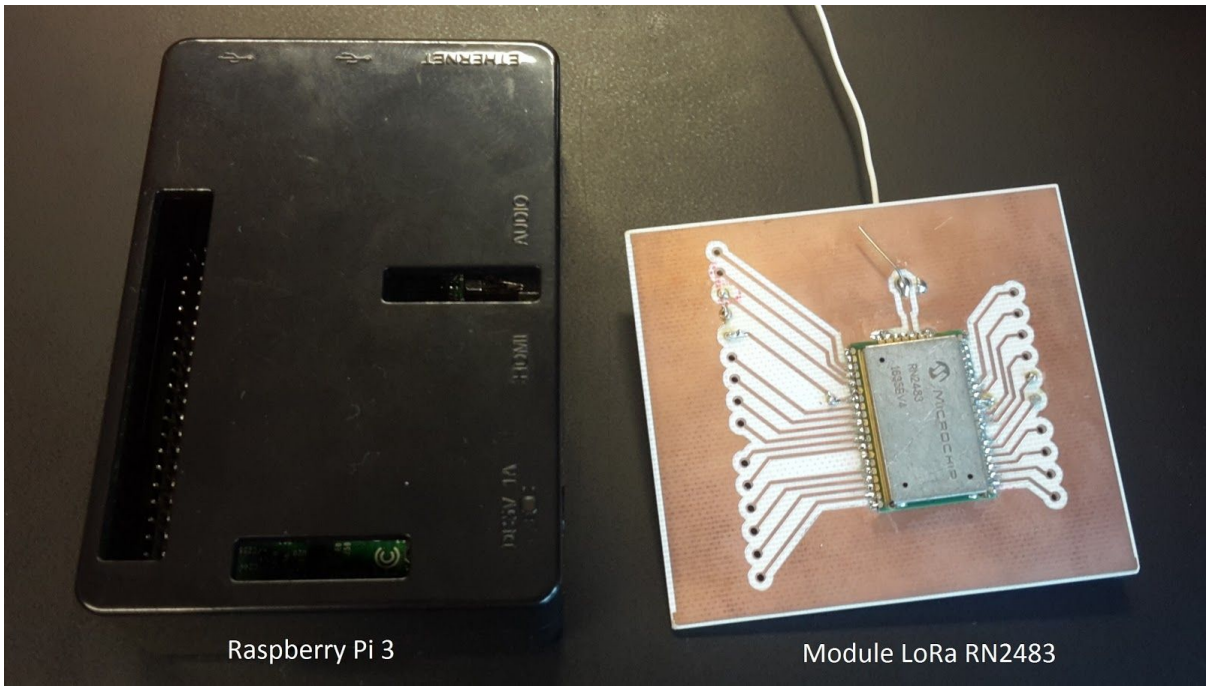




Annexe 4 : Matériels Utilisés (Module Lora,Module GPS et Arduino Uno)

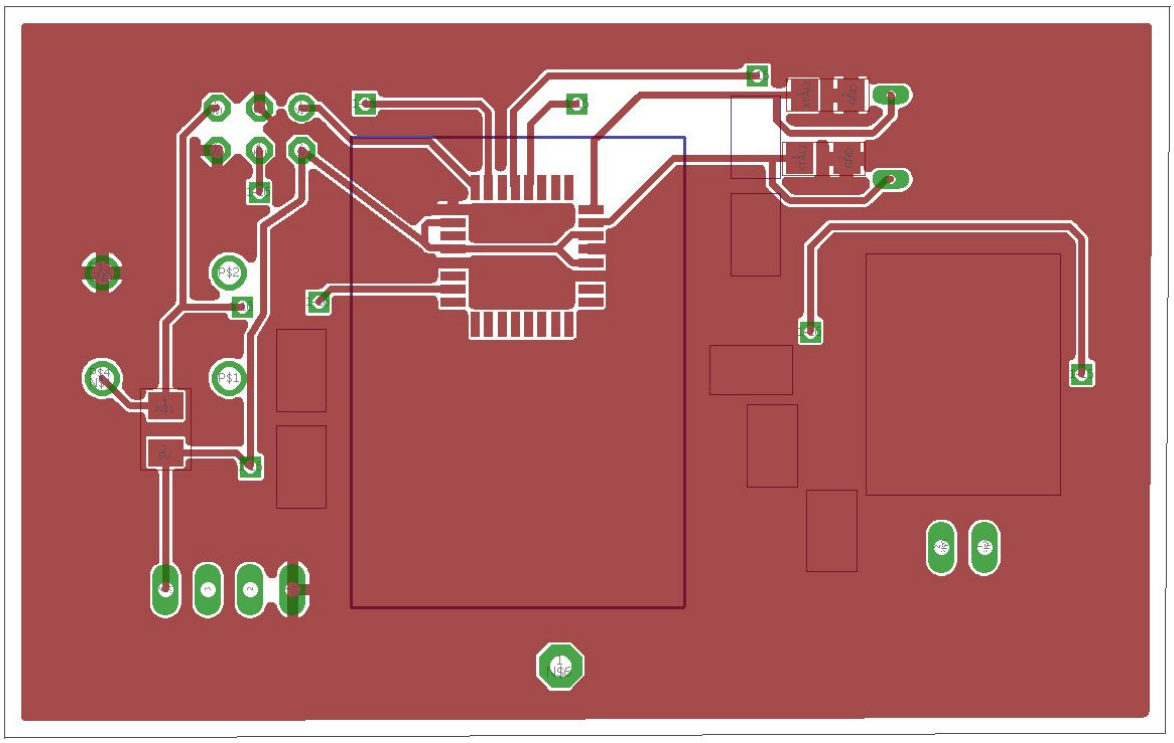


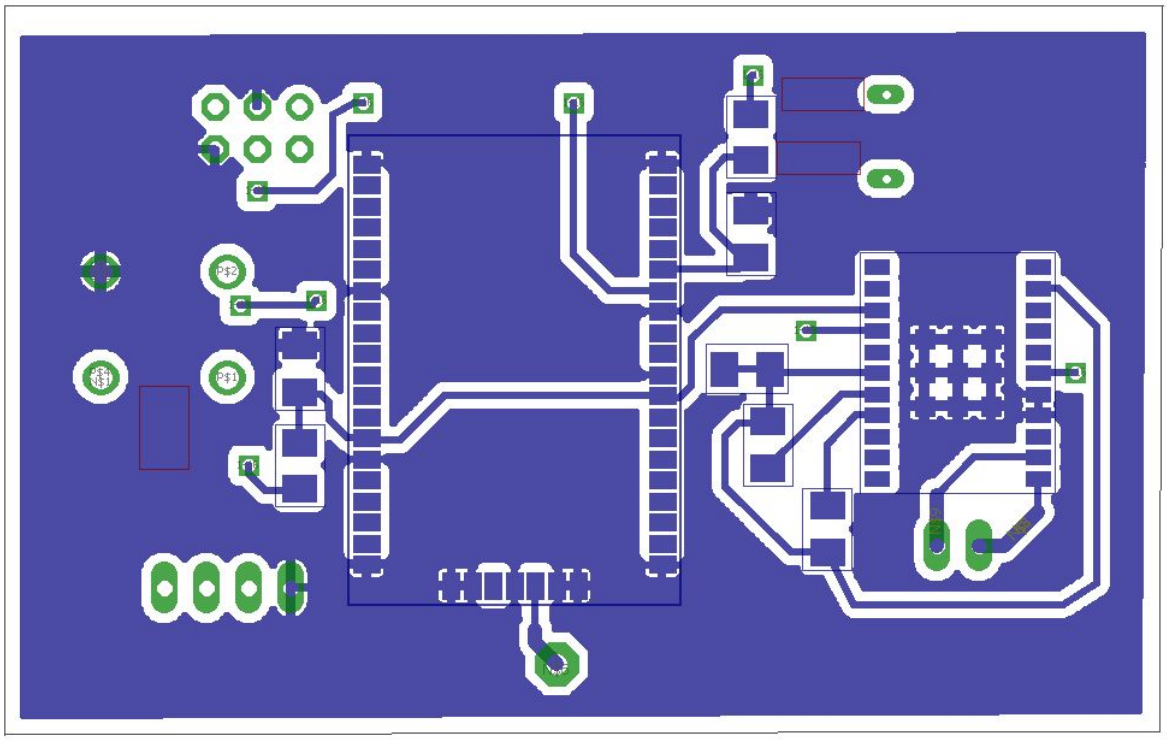
Annexe 5 : Matériels utilisés (Raspberry Pi et Module Lora)

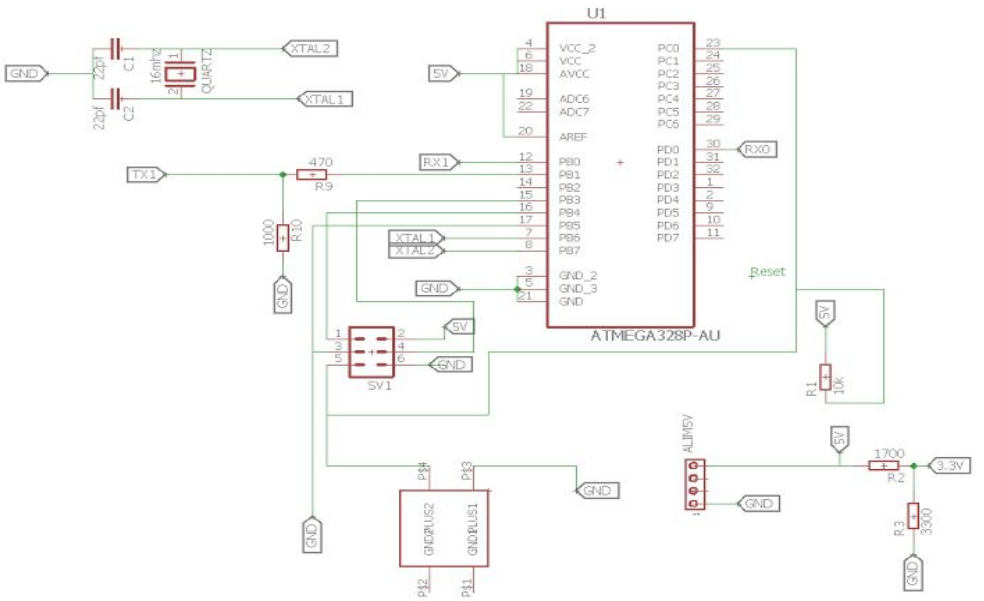


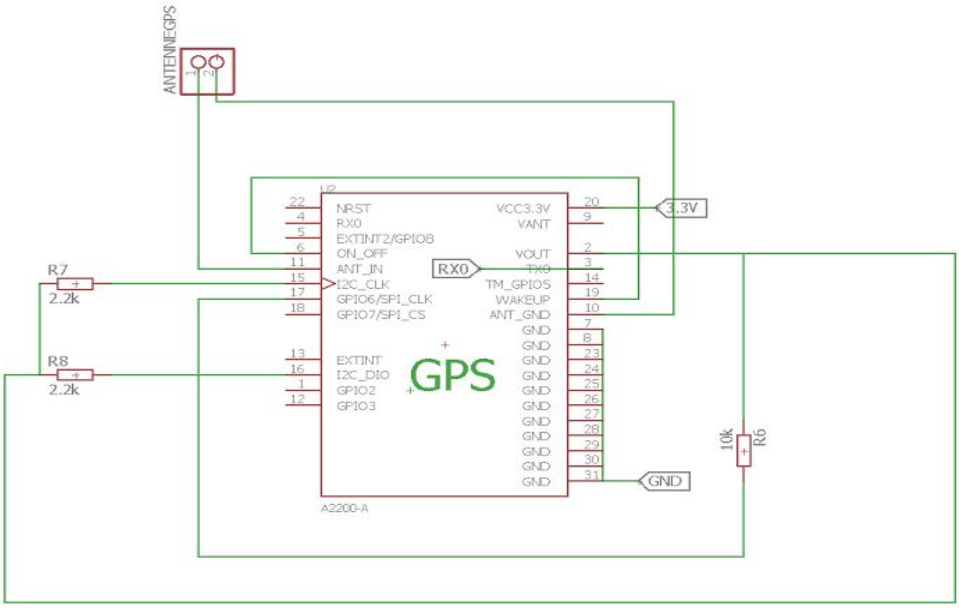
Annexe 6 : Schématique et PCB de la carte finale

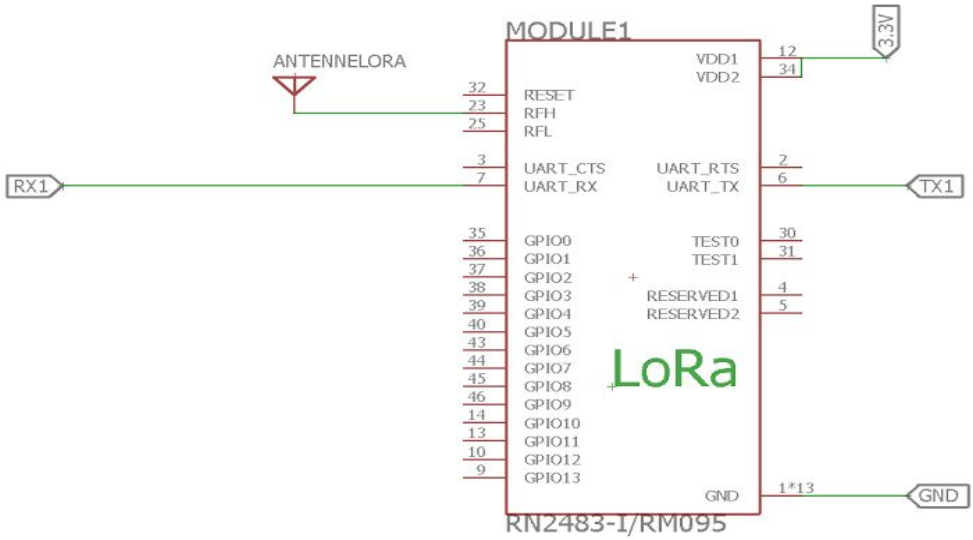
-






U1
ATMEGA328P-AU
XTAL2
XTAL1
QUARTZ
16mhz
22pf
C1
C2
GND
5V
TX1
RX1
RX0
470
R9
1000
R10
Reset
SV1
ALIMSV
1700
R2
3300
R3
R1
10k
3.3V


ANTENNEGPS
U2
GPS
A2200-A
NRST
RX0
EXTINT2/GPIO8
ON_OFF
ANT_IN
I2C_CLK
GPIO6/SPI_CLK
GPIO7/SPI_CS
EXTINT
I2C_DIO
GPIO2
GPIO3
VCC3.3V
VANT
VOUT
TX0
TM_GPIO5
WAKEUP
ANT_GND
GND
3.3V
RX0
R7
2.2k
R8
2.2k
R6
10k


MODULE1
RN2483-I/RM095
ANTENNELORA
LoRa
RESET
RFH
RFL
UART_CTS
UART_RX
UART_RTS
UART_TX
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
GPIO8
GPIO9
GPIO10
GPIO11
GPIO12
GPIO13
TEST0
TEST1
RESERVED1
RESERVED2
VDD1
VDD2
GND
3.3V
RX1
TX1
1*13

VI : Conclusion

Notre projet de fin d'études nous a permis de mettre en pratique toutes les notions acquises durant notre cycle ingénieur en Informatique Microélectronique et Automatique tels que la programmation (réseau, android, système embarqué), l'électronique(conception de carte et utilisation de composants électroniques) et l'étude des antennes.
Ce fut une très belle expérience et un réel plaisir d'en apprendre davantage et de concevoir deux trackers avec des technologies nouvelles et différentes.

VI : Bibliographie

- Datasheet du GPS A2200-A :
  - http://www.mouser.com/catalog/additional/Maestro_GPS_Receiver_A2200_A_User_Manual_V1.5.pdf

- Datasheet de la LoRa RN2483 :
  - http://www.farnell.com/datasheets/1947946.pdf?_ga=1.86893208.1208047654.1486462507

- Documentation pour le module LoRa :
  - https://www.thethingsnetwork.org/docs/

- Forums pour le module LoRa :
  - https://www.thethingsnetwork.org/forum/t/how-to-build-your-first-ttn-node-arduino-rn2483/1574
  - https://thinginnovations.uk/getting-started-with-microchip-rn2483-lorawan-modules
  - http://blog.smartcityzen.it/tag/rn2483/?lang=en
  - https://www.thethingsnetwork.org/forum/t/rn2483-breakout/729/45

- Forums pour l'application Smartphone :
  - http://www.allaboutcircuits.com/projects/how-to-communicate-with-a-custom-ble-using-an-android-app/
  - https://learn.sparkfun.com/tutorials/simblee-concepts/library-reference
  - http://ftp.icpdas.com/pub/beta_version/VHM/wince600/at91sam9g45m10ek_armv4i/cesysgen/sdk/inc/bt_sdp.h
  - https://code.tutsplus.com/tutorials/create-a-bluetooth-scanner-with-androids-bluetooth-api--cms-24084
  - https://developer.android.com/guide/topics/connectivity/bluetooth.html#ManagingAConnection
  - https://www.simblee.com/Getting%20Started%20with%20SimbleeForMobile%20v1.02.pdf

- Documentation Atmega :
  - http://www.mouser.com/ds/2/268/atmel-8271-8-bit-avr-microcontroller-atmega48a-48p-1065900.pdf
  - http://www.fiz-ix.com/2012/11/schematic-of-a-breadboard-arduino/

- Documentation Bluetooth Low Energy :
  - https://blog.groupe-sii.com/le-ble-bluetooth-low-energy/