Institut des Sciences et Techniques de l'Ingénieur d'Angers

4ème année, option Automatique et Génie Informatique
Année universitaire 2010 – 2011

Rapport de stage

Étude et développement d'un traqueur GSM/GPS

Soutenu par :
Simon Landrault

Maître de stage :
Michel Haigron

Entreprise :
TeckniSolar

Enseignant tuteur :
Laurent Hardouin

http ://www.tecknisolar.com/

4 quai du Val BP51
35403, St-Malo cedex
02 99 82 32 33

Acronymes

AOB "Application On Board", déclinaison du module fourni par Erco & Gener, ne possèdant pas de boîtier ou d'interface, mais seulement un logiciel embarqué.

API "Application Programming Interface", interface normalisée par l'intermédiaire de laquelle le programmeur a directement accès aux fonctions d'un système d'exploitation ou d'autres programmes.

AT Jeu de commandes développé pour les modems par le fabricant HAYES. Il est devenu par la suite un standard dans le monde des modems.

CMS "Composant Monté en Surface", composant électronique soudé en surface de la carte, non traversant.

GPS "Global Positionning System", technologie de positionnement par satellite.

GSM "Global System for Mobile communication", système de téléphonie sans fil utilisé par les téléphones portables.

Li-Ion Technologie de batterie utilisée, dont la réaction est fournie par un composé à base de lithium.

OEM "Original Equipment Manufacturer", dans notre cas le produit est fourni sans boîtier ni interface (connectique).

PCB "Printed Circuit Board", circuit imprimé sur lequel sont soudés les composants.

SMS "Short Message Service", mini message utilisé principalement sur les téléphones mobiles. Dans la majorité des cas, le SMS est limité à 160 caractères.

Remerciements

J'aimerais remercier l'ensemble du personnel de la société TeckniSolar, pour leur accueil et leur sympathie et plus particulièrement M. Pascal Barguidjian, *responsable de l'entreprise, pour m'avoir accepté en qualité de stagiaire, mais aussi M. Michel* Haigron, *mon maître de stage et ingénieur du bureau d'études.*

Merci aussi à Jérôme et Samuel, pour toute l'aide si précieuse qu'ils m'ont apportés tout au long de la période de stage.

Enfin, merci aussi à M. Laurent Hardouin, *enseignant à l'ISTIA et référent pour cette période de stage.*

Introduction

Pour ce stage de quatrième année, j'ai décidé de faire une activité proche de mon projet professionnel : développeur orienté système embarqué. J'ai donc décidé de le réaliser au sein de l'entreprise "TeckniSolar", à Saint-Malo.

L'entreprise TeckniSolar

L'entreprise qui m'accueille pour ces trois mois se nomme "TeckniSolar". Le bureau se situe à Saint-Malo. Son effectif est de six personnes :

- Le chef de l'entreprise M. Barguidjian
- Deux ingénieurs du bureau d'études (dont M. Haigron, mon maître de stage)
- Un technicien
- Une secrétaire
- Un étudiant en alternance

Son cœur d'activité est le développement et la réalisation de panneaux de signalisation lumineux autonomes. L'éclairage de ces panneaux est réalisé avec des diodes électroluminescentes (LEDs ou DELs) et l'apport d'énergie se fait avec des panneaux solaires ainsi qu'une batterie pour assurer l'alimentation lors des périodes avec moins de soleil.

Aujourd'hui, la société a diversifié son champ d'action. Sa principale activité reste la production de panneaux pour la sécurité routière, mais elle travaille aussi sur différents projets concernant aussi bien le civil que le militaire.

On retrouvera ainsi des projets concernant une veste de pompier "intelligente" (possédant différents capteurs) mais aussi des drones dédiés à la surveillance ou la reconnaissance de territoires.

Mon projet de stage

Le projet qu'il m'a été proposé de faire se situe au carrefour de différentes idées. En effet, je travaille sur l'étude et la conception d'un traqueur GPS. Le produit à concevoir possède une puce GPS et une carte SIM. Ce module peut ainsi savoir à tout moment sa position et la transmettre par SMS à un téléphone ou n'importe quel modem pouvant recevoir des messages.

Je ne suis pas parti de rien. En effet, à mon arrivé un logiciel avait déjà été développé (en Pascal) à partir d'un ancien traqueur "Tout-intégré" (produit acheté directement dans le commerce). La principale limite de cette application est quelle ne permet pas l'utilisation de plusieurs modules à la fois et ne peut pas évoluer, contrainte par ce caractère "tout-intégré". Ma tâche consiste donc à faire un nouveau logiciel pouvant utiliser et afficher à l'utilisateur plusieurs modules à la fois. Par exemple, si vous possédez une flotte de véhicules, vous pouvez à tout moment savoir quel véhicule se situe à quel endroit et ce partout dans le monde. Le module embarquant la puce GPS et la carte SIM possède déjà un logiciel embarqué. Cependant, le fabricant met à disposition tous les outils nécessaires pour effacer et réécrire un nouveau programme pour ce dernier. Ainsi, si la solution pré-intégré ne nous

convient pas (manque de fonction par exemple), il devient facile de modifier le code à l'intérieur du micro-contrôleur.

Le produit final doit donc regrouper les fonctions suivantes :
- Logiciel de visualisation "multi-traqueur"
- Module avec gestion de l'énergie intelligente
- Réveil et envoi de "Message d'alerte" lors d'un appui sur un bouton (SOS)
- Réveil et envoi d'un message lors de la détection d'un mouvement (vibration)
- Reproduction des fonctions existantes précédemment (envoi régulier, taille réduite...)

Suite à différents échanges tout au long du stage, certaines fonctions ont été rajoutées au fur et à mesure, telles que la sauvegarde/rappel d'historique, l'export de coordonnées...

À l'image de tous ces aspects, ce rapport se proposera de répondre à la question suivante :
"En quoi le développement de systèmes embarqués est-il pluridisciplinaires ?"

Réalisation

Le plan de ce rapport va donc suivre la progression logique du travail effectué tout en suivant les différentes spécialités rencontrés.
En arrivant dans l'entreprise, j'ai commencé par prendre connaissance du logiciel qui avait été fait précédemment. Ensuite, en attendant que le module sur lequel j'ai travaillé arrive, j'ai commencé à développer une interface en C++ avec le framework Qt. Ce développement fera l'objet de la première partie du rapport.

Une fois le module arrivé, j'ai commencé à faire quelques tests pour prouver le fonctionnement de quelques fonctions de base (telles que la requête de positions). J'ai aussi agrémenté le logiciel mentionné ci-dessus avec de nouvelles fonctions. Enfin, j'ai pu réaliser des prototypes de cartes électroniques pour pouvoir tester de manière "embarqué" les différentes fonctions réalisées. La deuxième partie du rapport sera consacré à ce développement électronique.

Lorsque le développement électronique a été achevé, j'ai pu continué le développement en modifiant le programme initial du module. Ce développement avait pour but d'ajouter des fonctions de veille afin d'optimiser l'utilisation de la batterie, point essentiel dans un système embarqué. La mise en place et la modification de ce programme sera expliqué dans une troisième partie du rapport.

Suite à tout cela, une conclusion fera un bilan sur le vécu de stage, ainsi que sur les travaux réalisés tout au long de celui-ci.

FIGURE 1 – Logo de l'entreprise "TeckniSolar SENI"

Sommaire

Acronymes **2**

Remerciements **3**

Introduction **4**

I Interface utilisateur **8**
0.1 Contexte . . . 9
0.2 Outils . . . 9

1 L'interface graphique **10**
1.1 Affichage des traqueurs . . . 11
1.2 Commandes des traqueurs . . . 11
1.3 Menu et sous-menu . . . 12

2 Le cœur du système **13**
2.1 La voie série et les SMS . . . 13
2.2 Les coordonnées . . . 14
2.3 Les cartes et l'API "Google Maps" . . . 17

3 Autres classes importantes **20**
3.1 SMS . . . 20
3.2 Coordonnées . . . 20
3.3 Tracker . . . 20

II Électronique **21**

4 Électronique générale **24**
4.1 Alimentation . . . 24
4.2 Antennes . . . 25
4.3 Prototypages . . . 27

5 Capteur de vibration **28**
5.1 Composant utilisé . . . 28
5.2 Circuit de filtrage . . . 28

6 Chargeur de batterie USB **30**
6.1 Principe . . . 30
6.2 Composant utilisé et contraintes . . . 31

III Informatique embarquée **32**

7 Chargement de la configuration **34**
7.1 Lecture/Écriture en mémoire flash 34
7.2 Initialisation 36

8 Cycles de fonctionnement **37**
8.1 Principe et organigramme du fonctionnement 37
8.2 Cycle court 39
8.3 Cycle long 39

Bilan **41**
8.4 Travail effectué 41
8.5 Méthodes de travail 42
8.6 Expression personnelle 42

IV Annexes **43**

A Carte électronique de support du module **44**
A.1 Vue schématique de la carte 44
A.2 Vue du typon 45
A.3 Réalisation finale 45

B Outils utilisés **46**
B.1 Réalisation de l'interface 46
B.2 Électronique 46
B.3 Informatique embarqué 46

Bibliographie **47**

Table des figures **48**

Première partie

Interface utilisateur

Afin de pouvoir surveiller les déplacements des différents traqueurs, une interface sur PC est réalisée. Cette interface permet de visualiser plusieurs traqueurs à la fois et aussi d'autres fonctions qui vont être présentées dans cette partie.

Afin de ne pas alourdir cette partie, seul les parties pertinentes en rapport avec le projet seront présentées. La création d'une interface ainsi que l'utilisation d'objet en C++ ne sera donc pas évoqué.

0.1 Contexte

Comme expliqué dans l'introduction, un logiciel avait déjà été réalisé précédemment. Ce dernier avait été fait en Pascal, un langage similaire au C++ mais dont je ne connais pas les subtilités. Son principal inconvénient est qu'il ne peut gérer qu'un seul traqueur à la fois. Sur cette partie du projet, la mission était donc de rendre le logiciel de supervision "multi-traqueur" (ayant la possibilité d'afficher le suivi de plusieurs modules en même temps).

Après avoir étudié le logiciel pendant les premiers jours du stage et suite à une discussion avec le dirigeant de la société, j'ai décidé de repartir sur un projet vierge, dans un langage que je maîtrise et avec des outils que je connais.

0.2 Outils

Pour réaliser cette interface, j'ai utilisé un framework [1] nommé "Qt". Il est produit par la société Nokia et distribué sous plusieurs types de licences. Dans notre cas, nous utilisons une licence de type LGPL [2] :

- Elle est gratuite
- Le logiciel produit peut-être distribué sans obligation de distribuer le code source...
- ...à la condition de distribuer les sources des modules Qt modifiés (s'il y en a)

Ce framework a été choisi car il permet de réaliser très facilement des interfaces graphiques en fournissant de nombreux composants standards. De plus, sa documentation est très claire et complète, il bénéficie d'une très grande communauté d'utilisateurs permettant la résolution de problème très rapidement. Enfin, il est personnalisable simplement. Si un composant graphique ou une bibliothèque est nécessaire mais n'existe pas de base, il est facile de trouver des librairies tiers partagés par d'autres personnes.

L'environnement de développement "Qt Creator" a aussi été utilisé afin d'améliorer la productivité. En effet, tous les composants sont ainsi intégrés dans un seul logiciel (développeur d'interface, documentation interactive, compilateur, débuggueur, gestionnaire de version...) et augmente donc l'intégration et l'amélioration de l'utilisation de l'ensemble.

FIGURE 2 – Le logo du framework Qt

1. Un framework (littéralement "cadre de travail") est un ensemble de bibliothèques, d'outils et de conventions permettant le développement d'applications.
2. Lesser General Public License

Chapitre 1

L'interface graphique

L'interface graphique a pour but de présenter à l'utilisateur les différents traqueurs qu'il utilise mais, aussi, de les paramètrer. Elle est divisé en trois parties visibles sur la figure 1.1 :

- La zone d'affichage des traqueurs (vert)
- La zone de réglage du traqueur sélectionné (rouge)
- Une barre de menu permettant différents règlages (bleu)

Ce chapitre va maintenant présenter chacune de ces différentes parties, leurs rôles ainsi que quelques détails fonctionnels.

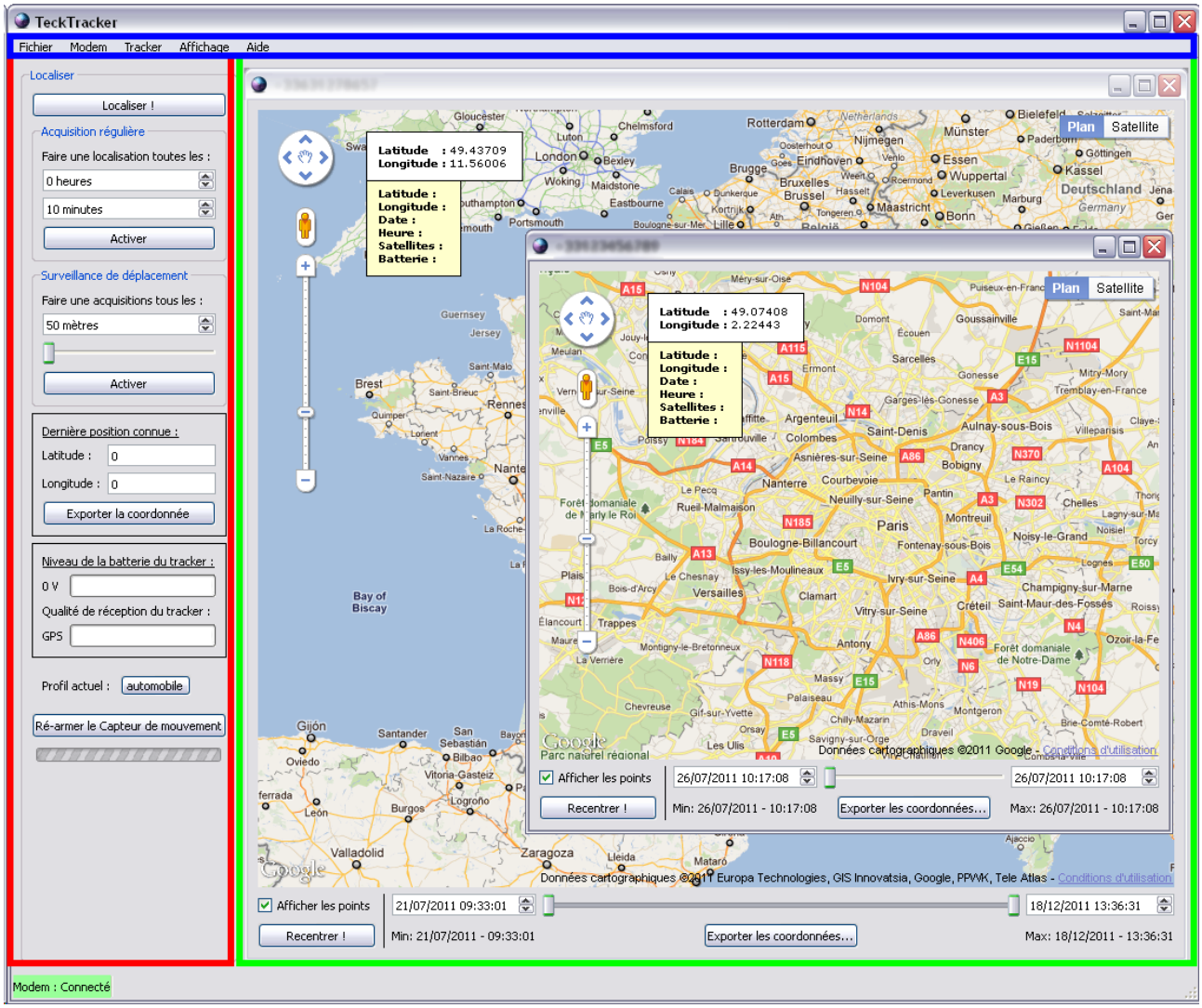

FIGURE 1.1 – L'interface utilisateur

1.1 Affichage des traqueurs

La partie occupant le plus d'espace est naturellement celle affichant les cartes des trajets des différents traqueurs. En effet, c'est cette section qui représente le rôle principal du projet, il est donc normal qu'elle occupe un espace majeur.
Elle possède plusieurs rôles :

- Afficher la position courante et le trajet effectué par le traqueur
- Filtrer l'historique des points pour n'afficher qu'une portion du trajet
- Donner des informations sur la nature du terrain (nom de rue, relief...) grâce aux différentes couches proposés par Google Maps

Le composant qui héberge cette zone est un QMdiArea. Ce composant particulier est en fait une zone d'accueil permettant de recevoir plusieurs occurences d'un composant enfant. Ici, le composant enfant sera représenté par la carte avec ses outils de réglages.
L'avantage de ce composant est qu'il permet très facilement d'arranger les fenêtres comme l'utilisateur le souhaite. Elles peuvent se chevaucher ou être disposées en onglet (comme le font les navigateurs web). Si l'utilisateur préfère, il peut aussi les afficher en plein écran et réduire les autres en arrière plan.

Afin de limiter les erreurs de manipulation, des messages d'avertissements apparaissent si l'utilisateur essaie de fermer le logiciel alors que des traqueurs sont actifs. Enfin, si l'utilisateur quitte tout de même, les fenêtres sont ré-ouvertes au démarrage suivant de l'application, dans l'état et à la position où elles étaient restées.

1.2 Commandes des traqueurs

Le second espace visible à l'écran est la zone de commande, comme vu sur la figure 1.2 ci-contre. Comme son nom l'indique, ce panneau sert à piloter les traqueurs. En effet, à chaque modification sur cette interface, le traqueur actif (dans la zone d'affichage) se verra envoyer un message pour changer d'état. Grâce à cet espace, on peut effectuer différentes actions :

- Demander une acquisition ponctuelle de position
- Demander une acquisition de position régulière (selon un temps ou une distance)
- Changer le profil dynamique du récepteur GPS (stationnaire, piéton, voiture...)
- Ré-armer le capteur de mouvement

Cette zone possède aussi un affichage des différents détails concernant le traqueur, tels que sa dernière position connue ou encore l'état de sa batterie. Bien entendu, lorsque l'utilisateur sélectionne une autre sous-fenêtre (et donc un autre traqueur) dans la zone d'affichage, les données sont mises à jour sur ce panneau.

Dans un souci de sécurité et pour éviter les erreurs de manipulations par un utilisateur non-habitué, un mode de supervision "simplifié" (aussi appelé "mode enfant") a été mis en place. Lorsque l'utilisateur a fini de régler tous ces traqueurs et ne souhaite plus faire de modifications pour un long moment, il peut décider d'enclencher ce mode. Le logiciel lui demande alors de saisir un mot de passe. Ensuite, l'interface devient plus simple. Seul les éléments "d'états" (dernière coordonnée, état de la batterie...) seront visibles.

FIGURE 1.2 – La zone de commande des traqueurs

Certaines actions du menu deviennent aussi inaccessible. Cela évite que quelqu'un ne fasse des erreurs en venant toucher au logiciel.
Pour revenir avec une interface complète, la personne doit de nouveau rentrer le mot de passe initial. Si jamais le logiciel est quitté, le mot de passe est sauvegardé (crypté avec un codage MD5) dans un fichier *.ini*. Ainsi, lorsque le logiciel est ouvert de nouveau, il retourne à son état "limité".

1.3 Menu et sous-menu

La dernière zone visible sur l'interface est la barre de menu, comme vu sur la figure 1.3. Cette barre est divisée en plusieurs sous-menus.

FIGURE 1.3 – La barre de menu

1.3.1 Modem

Ce sous-menu permet de gérer le modem. L'utilisateur peut ainsi le connecter ou le déconnecter. Lorsqu'il clique sur "connecter", une première fenêtre permettant de paramétrer la voie série s'affiche. Ensuite, si la voie série s'ouvre correctement, la procédure de configuration du modem se déroule (pour plus de détails, voir la section 2.1 "La voie série et les SMS").

1.3.2 Tracker

Ce sous-menu est le plus important. En effet, il gère le coeur de métier de l'application : les traqueurs. Á partir de ce menu, on peut ajouter des traqueurs à suivre dans la zone d'affichage, mais aussi juste charger l'historique d'un traqueur déjà utilisé. Une action permet aussi de régler certains paramètres avancés du traqueur courant (celui sélectionné sur l'affichage) :

- Régler les filtres logiciels du bouton SOS et du capteur de vibration
- Paramètrer la durée d'un cycle du traqueur
- Ajuster le comportement du ré-armement du capteur de vibration

Pour plus de détail concernant ces différents paramètres, se référer au chapitre 8 sur la programmation embarquée et les cycles de fonctionnement.

1.3.3 Affichage

Cette option ne propose que peu d'option.
En premier, on aura un bouton à bascule permettant de mettre l'affichage en mode "onglet" ou "ensemble de sous-fenêtres". Ensuite, une deuxième action permet de passer en mode de supervision uniquement ("mode enfant") évoqué plus haut.

1.3.4 Fichier et Affichage

Ces sous-menus n'ont pas un rôle très important. Le menu Fichier propose juste un bouton pour quitter l'application et le bouton d'aide affiche seulement une page "Á propos" concernant le framework Qt. En effet, par manque de temps je n'ai pas pu rédiger un guide d'utilisation complet. Cependant, des tooltips (encadré jaune présent s'affichant lorsque la souris reste longtemps sur un composant) sont présents sur de nombreux points pour aider l'utilisateur.

Chapitre 2

Le cœur du système

Cette partie va être consacrée au "cœur" du système. Il y sera présenté les classes qui se passent de l'interface graphique - ou qui n'en ont pas fondamentalement besoin - telles que celles interagissant avec le modem ou encore celles gérant la conversion et l'export des coordonnées. Enfin, il y sera expliqué le fonctionnement de l'interaction C++ $\leftrightarrow$ JavaScript utilisé pour faire de l'affichage dynamique grâce à l'API Google Maps.

2.1 La voie série et les SMS

2.1.1 Le modem

Afin de pouvoir communiquer avec le traqueur et pouvoir échanger des SMS de configuration ou de positions, on utilise un modem USB. La communication avec cet appareil se fait via une voie série émulée par le driver du modem. Comme le framework Qt ne possède pas de composant natif pour gérer les communications séries, on utilise une librairie tiers nommée "QextSerialPort".

FIGURE 2.1 – Exemple de Modem USB

Afin d'utiliser le modem, certaines précautions sont à prendre. Pour pouvoir communiquer, la voie série doit être correctement paramétrée (vitesse, bit de start/stop...etc). Ensuite, pour obtenir des informations, récupérer des SMS, il faut utiliser des commandes AT. Ces commandes normalisées ([4][9][10]) permettent le paramétrage du matériel mais aussi l'envoi de SMS, la lecture de SMS et d'autres fonctions.

2.1.2 Héritage

Comme expliqué précédemment, la programmation est faite en C++. Ce langage possède de nombreux avantages et l'on retiendra notamment celui de l'héritage. C'est dans ce cadre que l'utilisation d'une classe particulière prend tout son sens. En effet, pour pouvoir manipuler aisément le modem, sa classe est obtenue à partir d'héritage successif. Á la base, on part d'un objet de type "QIODevice". Cet objet est un "méta-objet" encadrant tout ce qui est relatif aux entrées/sorties matériels (gestion des flux de fichiers...etc). De cet objet dérive l'objet "QextSerialPort", provenant d'une librairie tiers et servant à communiquer avec une voie série. Enfin, la classe modem créée va dériver de QextSerialPort pour aboutir à la classe Modem.

On utilise une classe dérivée pour pouvoir lui ajouter des membres (variables) ou des méthodes (fonctions) que la classe parente ne possède pas. Dans notre cas, voici certains membres rajoutés notables :

- *db* : La base de données "archives" gérant les messages reçus mais non traités par les traqueurs
- *etat* : Variable indiquant si le modem est connecté et bien configuré

Et voici quelques méthodes supplémentaires :
- verifierSMS() : Méthode de lecture des SMS dans la mémoire du modem
- enregistrerSMS() : Méthode enregistrant un SMS dans la base de données d'archives
- envoyerSMS() : Comme son nom l'indique, elle permet d'envoyer des SMS

2.1.3 Fonctionnement

On hérite de la classe "QextSerialPort" afin de pouvoir rajouter des fonctions comme vu ci-dessus. Ce point est important car il permet de manière puissante d'améliorer le comportement de la classe de base pour s'adapter au cas particulier qu'est celui du modem. En effet, la classe parente toute seule ne permet pas de faire de l'enregistrement de SMS dans une base de données, ou encore ne permet pas de faire un envoi de SMS de manière simple. Ensuite, l'utilisation du modem se fera en plusieurs étapes.

En premier, on paramètre le modem dans l'ordre suivant :

1. On supprime l'echo
2. On vérifie que le code PIN est OK
3. On met le modem en mode "SMS Text"
4. On règle les mémoires de stockage par défaut (carte sim)
5. Enfin, on vérifie que le réseau est "accroché"

Une fois le modem réglé, on peut commencer à l'utiliser. Le modem sera utilisé pour deux actions principales, l'envoi de SMS et leurs réceptions.

Envoi de message

L'envoi de message consiste à écrire sur la voie série d'une manière ordonnée. Tout d'abord, on envoie la commande "AT+CMGS=" puis une chaîne de caractère contenant le numéro de téléphone du destinataire (le traqueur à paramétrer). Ensuite, il faut écrire le caractère "Entrée" (code ASCII 13). Enfin, on peut écrire le corps du message. Pour valider l'envoi, il faut écrire le caractère "ctrl+z" (code ASCII 26).

Lecture des messages reçus

La lecture des messages s'effectue en deux étapes. Tout d'abord, on va lire dans la mémoire de la carte SIM. Cette mémoire contient les messages qui arrivent lorsque le modem est sous tension. Dans un second temps, on va lire dans la mémoire nommée "ME". Cette mémoire stocke les messages qui arrivent lorsque le modem est hors tension (et qui donc arrivent après démarrage du modem, avant qu'il ne soit paramétré par l'application). La commande permettant de lire les messages est "AT+CMGL="ALL"". Naturellement, lorsqu'un message est lu, il faut le supprimer de la mémoire pour éviter de la saturer et ainsi être sur de ne rater aucun message (à l'aide de la commande "AT+CMGD=n" n étant le numéro du message à supprimer).

2.2 Les coordonnées

Lorsque le GPS envoie ses coordonnées au sein d'un SMS, elles sont dans un format "degrés/minutes". Cependant, pour qu'elles puissent être utilisée par l'API Google Maps, elles doivent être dans un format contenant uniquement des degrés. Une fois la conversion effectuée, la position peut être affichée voire exportée pour être exploitée par un autre support.
Cette partie va présenter ces différentes étapes.

2.2.1 Conversion

Comme expliqué ci-dessus, les coordonnées sont exprimées au format "degrés/minutes". Notre but est donc d'obtenir des degrés. La méthode est similaire pour la latitude et la longitude, la seule différence étant les valeurs maximales :

- La latitude va de -90 à +90 degrés
- La longitude va de -180 à +180 degrés

Pour la suite, nous utiliserons l'extrait de trame suivant :

$$[...], 4807.038, S, 01131.324, E, [...]$$

Latitude

Dans la trame contenant les coordonnées, la latitude est représentée par le premier couple "nombre/lettre". La valeur contenant la coordonnée sera de la forme "*ddmm.mmm*" (*dd* représentant les degrés et *mm.mmm* étant les minutes). Il faut donc parser la chaine pour isoler les degrés des minutes. Dans l'exemple, on a donc 48 degrés et 7,038 minutes.
Une fois les nombres isolés, de simples opérations permettent de passer les minutes (et secondes) en degrés.

1. On isole les minutes (ici : 7)
2. On déduit les secondes ($0.038 * 60 = 2.28 secondes$)
3. Enfin, on les convertit en degrés en divisant les minutes par 60 et les secondes par 3600 (donc on obtient ici $7/60 + 2.28/3600 = 0.1173$ degrés)
4. Pour finir, on additionne ces résultats aux degrés initiaux et on obtient $48 + 0.1173 = 48.1173$ degrés.

La lettre suivante le nombre indique si l'on se trouve au nord (N) ou au sud (S). Dans ce dernier cas, le nombre final est négatif, sinon il est positif.
Ici, le nombre final sera -48.1173 *degrés.*

Longitude

La conversion de la longitude fonctionne de la même manière à l'exception que la forme est "*dddmm.mmm*" (*ddd* représentant les degrés et *mm.mmm* étant les minutes). Comme on peut le voir, la seule différence est que les degrés initiaux de la trame possède un chiffre de plus.
En faisant le calcul précédent on a :

- Degrés initiaux : 11
- Minutes : 31
- Secondes : $0.324 * 60 = 19.44 secondes$

La longitude obtenue au final sera donc : 11.522 degrés. La encore, la lettre indique la direction (Est (E) ou Ouest (W)). Si la direction est d'ouest, le nombre doit être négatif.

2.2.2 Export

Une des fonctions qui a été demandée suite à une rencontre avec un client est l'export d'une coordonnée afin de l'utiliser avec un autre support. Par exemple, on peut utiliser le fichier dans le GPS d'une voiture pour ensuite être guidé vers le traqueur ou la position choisie. En exportant plusieurs coordonnées, on peut aussi refaire le tracé sur un autre logiciel tel que Google Earth.

Format GPX

Ce premier format signifie "GPS eXchange" [11]. C'est un format de fichier normalisé et ouvert, basé sur une structure XML [1]. Il est utilisé par certains constructeurs de GPS du commerce.

1. eXtensible Markup Language (langage de balisage évolué), langage d'expression des données représenté sous forme de balises en arbre

Structure

La structure de ce type de fichier est faite à partir d'un format XML, où les données sont organisées dans un système d'arbres à balises. La balise centrale, décrivant une position est la balise "Waypoint" : <wpt>.
Elle possède deux propriétés importantes, *lon* et *lat*, représentant respectivement la longitude et la latitude exprimées en degrés. D'autres propriétés peuvent servir pour donner plus de détails sur le Waypoint tels que la vitesse, l'altitude ou la date.

Exemple

Voici un exemple de fichier que l'on peut obtenir avec l'exemple de la partie précédente :

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx
   version="1.1"
   creator="TekniSolar"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://www.topografix.com/GPX/1/1"
   xsi:schemaLocation="http://www.topografix.com/gpx/1/1/gpx.xsd">
     <wpt lat="-48.1173" lon="11.522"></wpt>
</gpx>
```

Pour décrire un trajet, il suffit de mettre plusieurs balises <wpt> à la suite.

Format KML

Le format KML (Keyhole Markup Language) est lui aussi basé sur le formalisme XML [12]. Il est utilisé principalement au sein des applications google Maps et Google Earth, mais aussi par de nombreuses autres applications qui l'ont rendu populaires.

Structure

Son fonctionnement est similaire à celui du format GPX. La différence se situe au niveau de la balise utilisée. Ici, elle se nomme <Placemark>. Le Placemark contient lui-même une balise <Point> qui contiendra la balise <coordinates> qui va renfermer les données géographiques. Dans cette balise <coordinates>, on place la longitude, puis la latitude puis, si disponible, l'altitude. Les coordonnées sont là encore exprimées en degrés.

Exemple

Voici un exemple de fichier que l'on peut obtenir avec l'exemple de la partie précédente :

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Placemark>
      <Point>
        <coordinates>11.522,-48.1173</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

Là encore, pour décrire un trajet, il suffit de mettre plusieurs balises <*Placemark*> <*Point*> <*coordinates*> à la suite au sein de l'entité <Document>.

2.3 Les cartes et l'API "Google Maps"

Afin de pouvoir faire un affichage interactif avec l'utilisateur, le rendu des cartes se fait à partir de l'API Google Maps fourni par Google. Cette API permet d'afficher des cartes de types différents types (géographiques, géopolitiques...etc) et à différents niveaux de détails et de zoom. De plus, il est possible de dessiner différentes informations supplémentaires telles que des marqueurs référencés géographiquement, des polygones....

2.3.1 Initialisation de la carte

Le contenu principal de la carte est contenu dans un fichier HTML, fichier standard d'affichage par un navigateur internet. Dans l'en-tête du fichier, on peut trouver la ligne suivante :

```
<script type="text/javascript"
  src="http://maps.google.com/maps/api/js?libraries=geometry&sensor=false">
</script>
```

Son rôle est de préciser au moteur web, lorsqu'il charge la page, qu'il doit aussi charger l'api google maps à l'adresse précisée. Ce chargement permet ensuite d'utiliser toutes les fonctions fournies par Google.

Ensuite, une fonction nommée "load()" sera exécutée lors du chargement de la page. Voici son contenu :

```
function load() {
  var france = new google.maps.LatLng(46.75984, 1.738281);
  var myOptions = {
    zoom: 6,
    center: france,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  };
  map=new google.maps.Map(document.getElementById("map_canvas"),myOptions);
}
```

Cette fonction a pour but de créer un objet de type "Map" de l'API Google maps. Des options précisent le niveau de zoom ainsi que le type de fond de carte et la région sur laquelle la carte doit-être centrée. Le résultat de ce travail peut-être vu sur la capture d'écran (2.2) suivante :

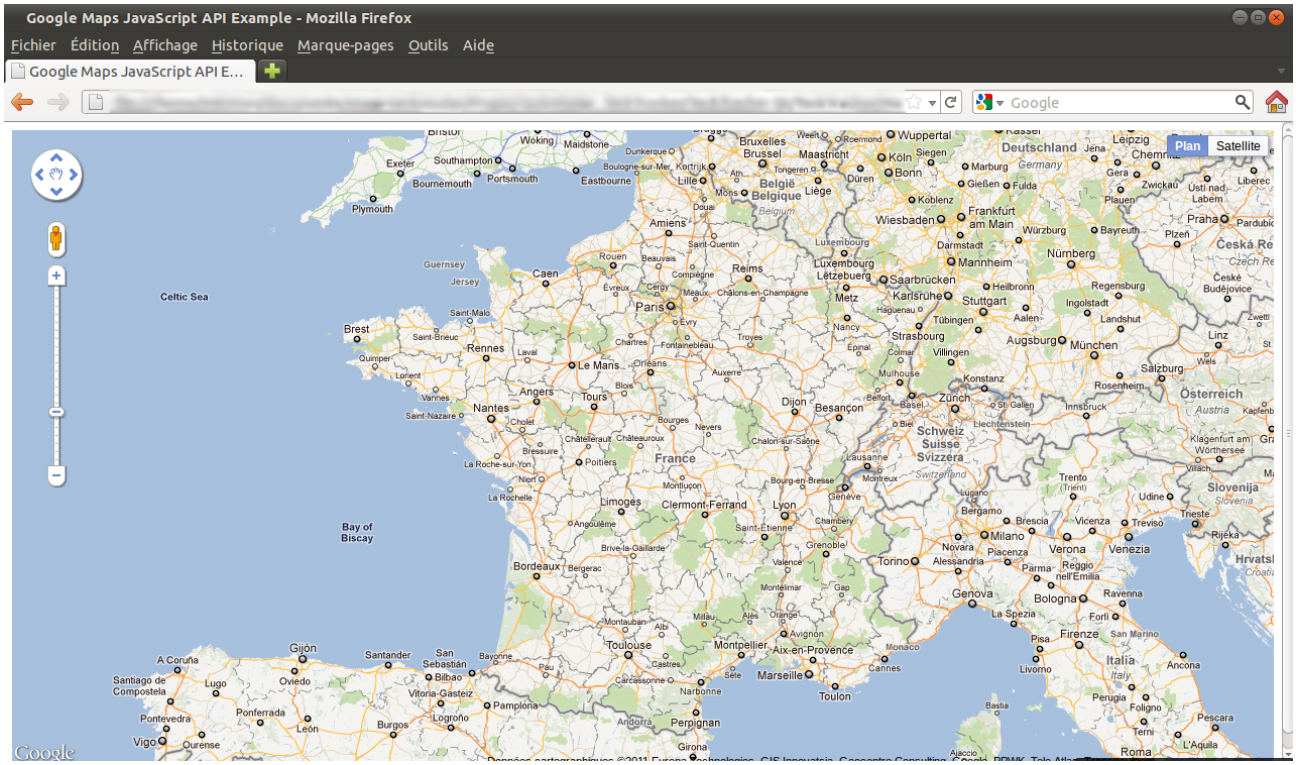

FIGURE 2.2 – La carte initialement chargée

2.3.2 Fonctionnement de l'interaction C++ ↔ JavaScript

Dans notre cas, la carte n'est pas chargée directement par un navigateur web traditionnel, mais par un objet du framework Qt servant à afficher du contenu web (QWebView). Lors de la création d'une carte, cet objet charge le contenu du *carte.html* qui va ensuite charger la carte et l'API. Ensuite, pour pouvoir interagir avec la carte chargée, on fait appel à des fonctions JavaScript que l'on "injecte" dans le code html chargé dans l'objet QWebView. La fonction permettant ce travail est la méthode "evaluateJavaScript()".

Par exemple, si l'on voulait afficher une boîte de dialogue d'alerte on pourrait faire :

```
evaluateJavaScript("alert(\"Hello World\")");
```

2.3.3 les marqueurs de positions

A chaque fois que le traqueur envoie une position, on la représente sur la carte sous la forme d'un cercle. Le cercle à pour but de montrer à l'utilisateur que le GPS n'a pas une précision absolue. Par rapport au fond de carte, le cercle dessiné mesure 3m de diamètre au sol.

La classe qui gère la carte possède une variable membre qui représente l'ensemble des positions faites par le traqueur. Cette liste est parcourue et affichée à chaque fois qu'une position est ajoutée ou supprimée. Lorsqu'on modifie un marqueur, on appelle une fonction JavaScript contenue dans le fichier *carte.html*. Cette fonction ajoute alors le point sur la carte par rapport aux coordonnées passées en paramètre.

2.3.4 Dessin du trajet

Lorsque l'on a plusieurs positions, on peut dessiner un trajet. Ce trajet sert à estimer la trajectoire empruntée par le traqueur. Il sera plus ou moins précis selon l'intervalle entre deux mesures. En effet, il représente la trajectoire à vol d'oiseau entre deux points. Une fois de plus, c'est une fonction JavaScript qui va se charger de faire le dessin du polygone représentant le trajet.

2.3.5 Affichage d'informations sur évènement

Pour donner un maximum d'informations à l'utilisateur, des données sont affichées en fonction des mouvements de la souris. Lorsque la souris se déplace sur le fond de carte, on donne dynamiquement la coordonnée du curseur par rapport à sa place sur la carte. Si le pointeur survol un marqueur, on pourra voir dans un cadre des informations sur ce dernier :

- La coordonnée du centre du marqueur
- La date et l'heure d'acquisition de la coordonnée
- Le nombre de satellites visibles lors de l'établissement de la position (estimation de la qualité de l'information)
- La tension de la batterie au moment de l'acquisition (estimation de l'autonomie restante, statistique)

Toutes ces fonctions peuvent être retrouvées sur la capture d'écran (2.3) suivante :

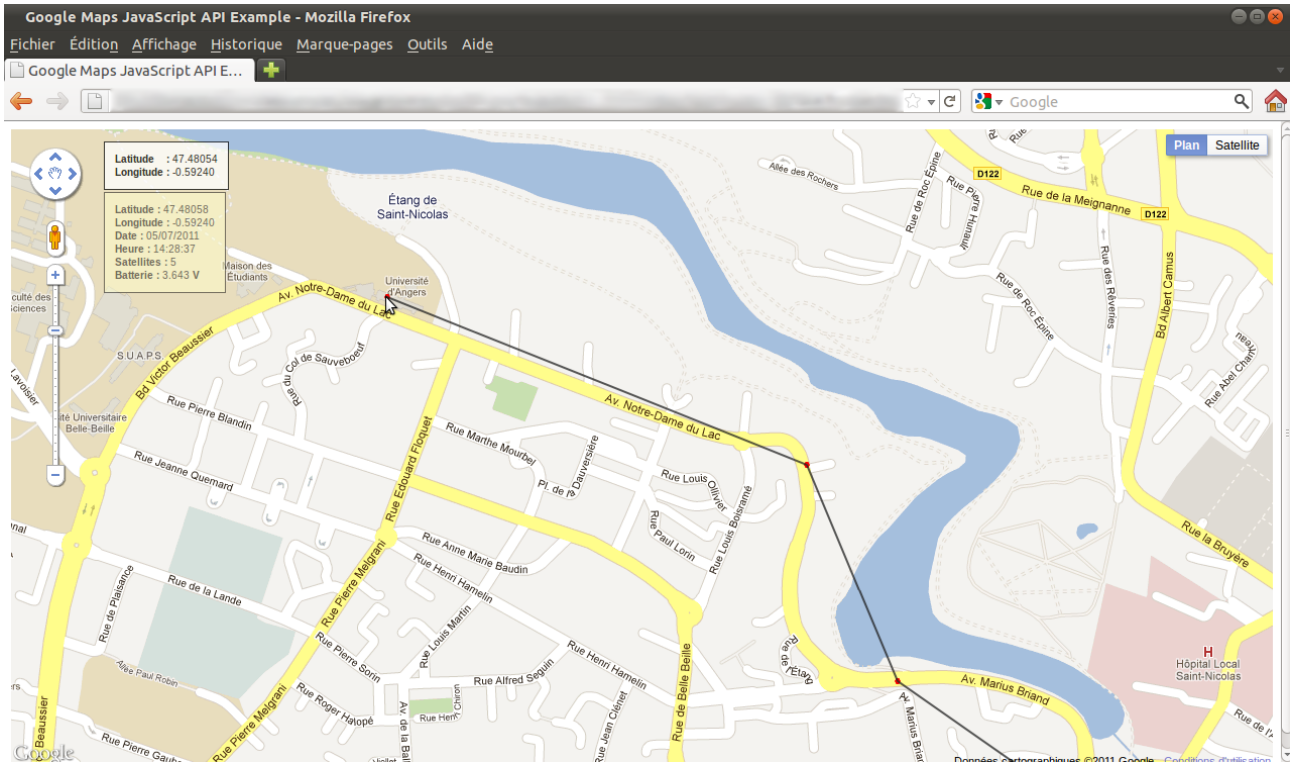

FIGURE 2.3 – La carte et tous les détails affichés

Chapitre 3

Autres classes importantes

Ce chapitre a pour but de présenter brièvement quelques classes évoquées précédemment et ayant été développées dans le but de faire des "Unités Logiques" de fonctionnement, pour garantir une clarté du code dans son ensemble.

3.1 SMS

Comme son nom l'indique, cette classe sert à représenter ce qu'est un SMS. Son principal objectif est de regrouper au sein de différentes variables les différents champs d'un SMS :

- Sa date et son heure de réception
- Le numéro de téléphone de l'émetteur
- Le message contenu dans le message

Elle se révèle très utile lorsqu'il s'agit de faire la transition entre le modem qui reçoit les SMS et la classe représentant le traqueur concerné. En effet, plutôt que de devoir passer plusieurs variables et d'avoir des fonctions avec de nombreux paramètres, on se contente juste d'un seul paramètre qu'est l'objet SMS.

3.2 Coordonnées

Pour optimiser les traitements, un objet "Coordonnées" a été créé. Cette objet se sert de l'héritage. En effet, il hérite d'un objet de type "QPointF" qui représente un couple de deux floats, servant à représenter une coordonnée planaire. Par rapport à une coordonnée planaire, cet objet possède d'autres attributs afin de représenter au mieux une position issue d'un GPS :

- La date et l'heure du fix du GPS
- La qualité du fix ('A' ou 'V' pour respectivement "bon" ou "mauvais")
- L'altitude et la vitesse
- Le nombre de satellites utilisé au moment de l'acquisition

Le but est bien entendu encore une fois d'optimiser la logique, la lisibilité et les échanges au sein du programme.

3.3 Tracker

Cette dernière classe représente le traqueur d'une manière virtuelle. Il est identifié par un numéro de téléphone et possède un numéro de modem avec lequel communiqué. Il sert aussi à définir de manière unique la base de données servant à sauvegarder toutes les coordonnées qu'il a pu émettre en fonctionnant. Enfin, il possède différents parseurs de trame, afin d'extraire les données reçues des SMS du modem.

Deuxième partie

Électronique

Le module qui contient le récepteur GPS et le modem GSM a été acheté à la société “Erco & Gener”. Ce composant se nomme “GenLoc OEM AOB” (Application On Board) comme vu sur la figure 3.1.

Ce module se présente donc dans sa forme la plus simple, permettant d'interfacer les signaux du microcontrôleur comme bon nous semble. Cependant, pour que l'ensemble fonctionne correctement, plusieurs contraintes doivent être prises en compte :

- L'alimentation doit être comprise entre 3.3V et 4.5V.
- Pour répondre aux cahier des charges, une détection de vibration/mouvement doit être possible.
- Une alimentation externe doit pouvoir être branchée sans perturber le système.
- L'ajout d'un chargeur de batterie par USB serait un plus.

Cette partie va présenter chaque sous-ensemble du montage final comme montré sur la figure 3.2. On commencera par les détails concernant l'alimentation du module. Ensuite, on traitera des ajouts supplémentaires que sont le capteur de vibration et le chargeur de batterie par USB.

FIGURE 3.1 – Le module “GenLoc OEM AOB” utilisé

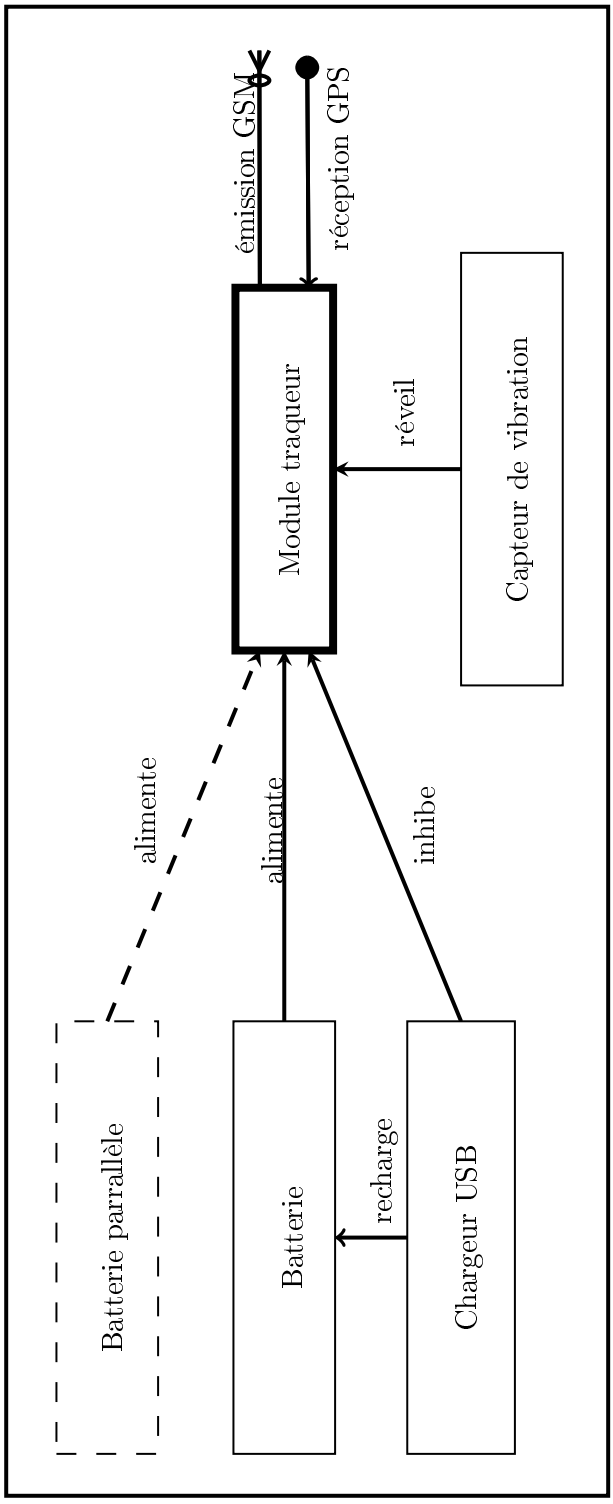

FIGURE 3.2 – Synoptique du fonctionnement de l'ensemble

Chapitre 4

Électronique générale

Afin que le module utilisé puisse fonctionner correctement, plusieurs contraintes doivent être respectées. Parmi elles, se trouvent l'alimentation et les connections aux antennes (GSM et GPS) ainsi que la possibilité de réaliser le prototype avec les moyens présents.

4.1 Alimentation

Le traqueur doit normalement être alimenté par une tension continue de 5 V qui doit pouvoir délivrer jusqu'à 1,8 A (pics de courant lors d'envoie de messages). Bien que faible, cette tension n'est pas adapté aux contraintes matérielles fixées. En effet, nous souhaitons utiliser une batterie de type li-ion (figure 4.1 ci-contre) ayant une tension nominale de 3,7 V (maximum de 4,2 V et minimum vers 3 V). Pour pouvoir atteindre les 5 volts requis, il faudrait réaliser un convertisseur DC/DC qui, en plus de prendre de la place, engendre des pertes lors de la conversion.

FIGURE 4.1 – La batterie Li-ion utilisée

Après quelques recherches dans les documentations fournies par le constructeur [3], une information concernant une alimentation entre 3,3 V et 4,5 V est en option. Plusieurs échanges d'email avec le service technique du constructeur ont ainsi pu confirmer que le module pouvait être alimenté dans cette plage, et la modification à faire a été mise en place. La seule condition est que la batterie doit pouvoir fournir un courant pic de 1,8 A. Pour alléger la tâche de la batterie concernant ce paramètre, un condensateur (C2) est placé au plus près des bornes de l'alimentation du traqueur. Ce condensateur est de technologie au tantale, permettant ainsi une très grosse capacité (1000 uF) dans un encombrement réduit (composant CMS).

Afin de laisser un maximum de souplesse à l'utilisateur final, un deuxième connecteur de batterie a été placé. Ce second connecteur permet d'ajouter une seconde batterie en parallèle. L'utilisateur peut ainsi augmenter la durée de vie de l'ensemble. Pour éviter des problèmes de court-circuit entre les batteries, une diode (D2 et D3) est placée. Enfin, pour optimiser un maximum la consommation, les diodes choisies ont une très faible chute de tension (moins de 300 mV à 2 A) pour limiter les pertes.

Pour permettre à l'utilisateur final de savoir si le traqueur est toujours en état de fonctionnement ou à besoin d'être rechargé, la tension de la batterie est mesurée. Le convertisseur analogique-numérique intégré au traqueur possède une limite à 3,3 V. La batterie pouvant délivrer une tension allant jusqu'à 4,2 V, un diviseur de tension a été placé pour diminuer la tension en entrée du convertisseur. Ce diviseur est fait à partir des résistances R1 et R3. La valeur de la division est la suivante :

$$Ratio = \frac{R2}{R1 + R2} = \frac{33}{10 + 33} = 0,77$$

Lorsque la batterie sera pleinement chargée, on lira alors une tension de 3,22V (au lieu de 4,2 V). Le ratio inverse est ensuite paramétré dans le traqueur afin de corriger la valeur lue.

Tous ces détails peuvent-êtres retrouvés sur le schéma de la figure 4.2 suivante.

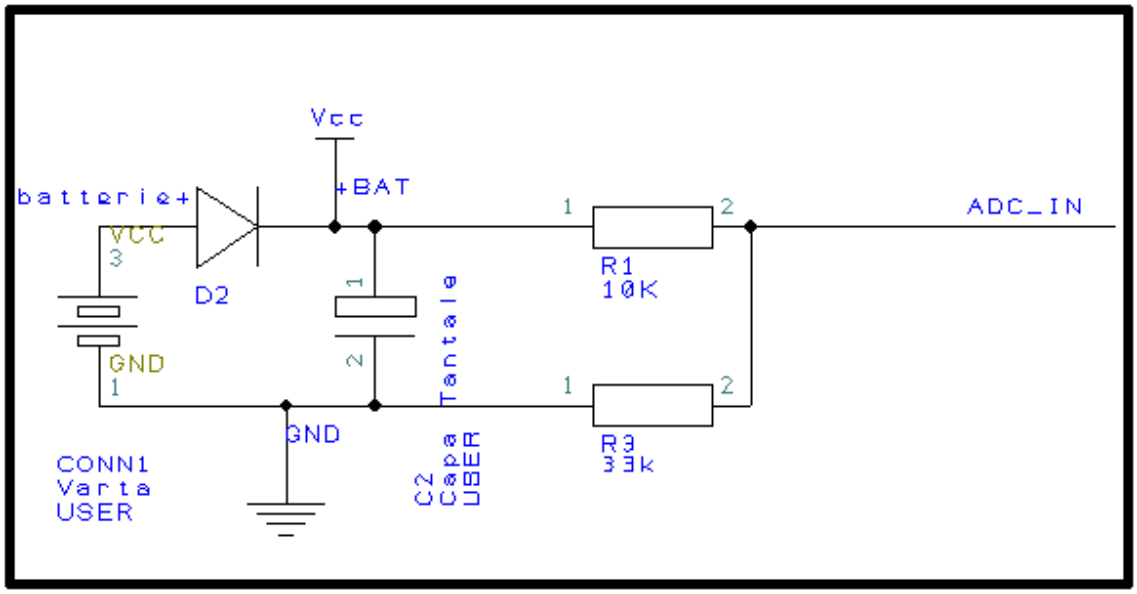

FIGURE 4.2 – Schéma électronique du circuit d'alimentation

4.2 Antennes

Le traqueur possède deux antennes distinctes. Une sert à la communication GSM (pour échanger les SMS) et l'autre sert à la réception GPS (figure 4.3). Toujours dans l'optique de faire un produit le plus compact possible, ces antennes doivent être les plus petites possibles. Le module de base est fourni avec les deux antennes, d'une taille assez conséquente (et avec 3 mètres de câble chacune). Aucun connecteur n'est utilisé, les câbles coaxiaux respectifs sont soudés directement sur le PCB.

GSM

Comme énoncé précédemment, cette antenne sert à la communication GSM. Elle est utilisée par le modem intégré pour échanger les SMS de configuration ainsi que l'envoi des positions. La communication de type SMS ne nécessite pas une trop bonne qualité de réception pour pouvoir fonctionner (contrairement aux échanges de type "DATA"). Il a donc été décidé de faire l'antenne directement sur le PCB, à titre d'essai. Sur le PCB final, on peut donc voir un "fil plat" dessiné sur le côté de la carte. Ce fil est en fait l'antenne.

À la base, il était juste réalisé à partir du cuivre de la plaque. Après quelques tests, le cuivre a été recouvert d'étain pour éventuellement améliorer la réception. Cette modification s'est avérée concluante puisqu'on a pu observer une amélioration de 2 points (sur 31 [1]) de la réception.

1. Résultat tiré du retour de la commande GSM "AT+CSQ ?" [4]

Enfin, "l'antenne" a été reliée au support de l'ancienne antenne avec un court câble coaxial. Ce fil à la très bonne caractéristique d'être bien isolé par rapport au bruit. En effet, il possède trois composants principaux :

- Le coeur, qui est un fil conducteur transportant le signal
- Une enveloppe plastique autour du coeur
- Puis un conducteur, tressé autour de l'enveloppe plastique et relié à la masse pour isoler le signal.

Pour l'immédiat, cette antenne est fonctionnelle et remplie son rôle. Cependant, si l'application évolue dans le futur et utilise un échange de données en "DATA", il sera bon de retravailler sur cette dernière, car son efficacité ne sera probablement pas suffisante.

GPS

Comme son nom l'indique, cette antenne est destinée à la réception des signaux émis par les GPS. Ici, il n'est pas question de réaliser un dessin directement sur le PCB. En effet, les antennes GPS sont en fait un ensemble intégré de récepteurs et de filtres pour décoder les signaux. La qualité de ces antennes (comme toute antenne) est déterminée par leurs sensibilités. Comme elles sont actives (intégrant un filtre), un autre paramètre à prendre en compte est le gain (l'ensemble permettant d'isoler le signal utile par rapport au bruit).

Dans un premier temps, une antenne venant d'un ancien modèle de traqueur a été utilisée. Cependant, après plusieurs tests, les résultats n'étaient pas au rendez-vous. En effet, avec l'ancienne antenne, la réception était assurée en extérieur après peu de temps et en intérieur en laissant du temps au capteur. Avec cet autre modèle, la réception en intérieur était impossible, et en extérieur, les signaux obtenus demandaient un temps d'allumage trop long (incompatible avec les modes de veille établi) ou alors était peu précis. Une décision a alors été prise de réutiliser l'antenne fournie initialement. Son câble a été revu au minimum afin de ne pas encombrer. Cependant, l'antenne est enfermée dans un boîtier, empêchant de l'extraire et de l'intégrer au mieux sur la carte réalisée.

Une dernière solution entreprise a été de commander un autre modèle d'antenne sur un site internet spécialisé. L'antenne n'a pas pu être livré avant la fin du stage, je n'ai donc pas pu faire de tests avec. Ceci est dommage puisque ses caractéristiques étaient supérieures au modèle courant (gain maximum de 40 dB contre 29 dB).

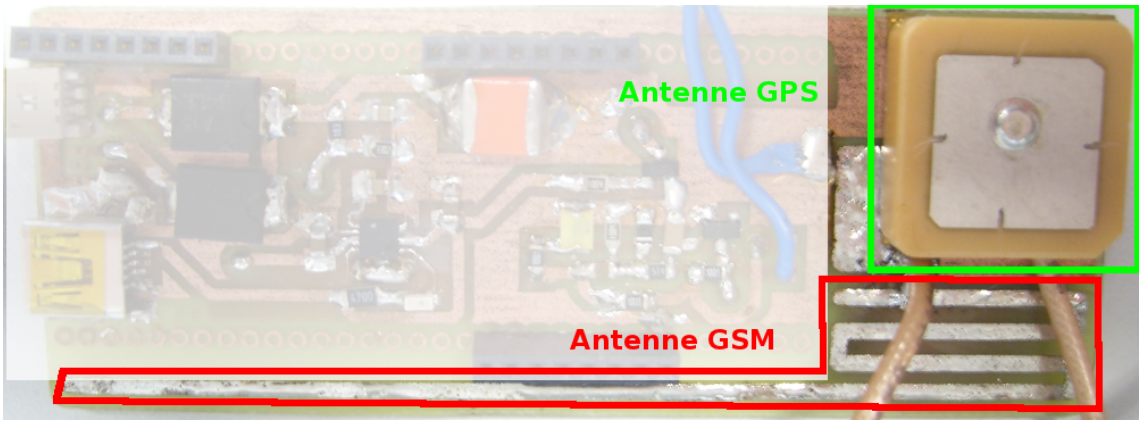

FIGURE 4.3 – Aperçu des antennes utilisées

4.3 Prototypages

Trois prototypes ont été réalisés durant ce stage.

Le premier a été réalisé assez tôt et avait pour but de prouver le côté "alimentation", en validant l'alimentation depuis une source de tension entre 3,3V et 4,5V au lieu des 5V par défaut. Il a aussi permis de vérifier que le condensateur de 1000 uF était efficace et que l'empreinte du module était exact. Enfin, le convertisseur analogique numérique a pu être mis en place de manière plus fine (en mettant en place la correction du diviseur de tension de manière logicielle par exemple).

Le second prototype devait embarquer toutes les fonctions restantes (capteur de vibration, chargeur USB, seconde alimentation...). Il a permis de corriger plusieurs erreurs :

- Certaines empreintes étaient fausses et ont dû être refaites.
- Des composants étaient manquants (transistor pour bloquer le module pendant la charge USB et résistance de tirage à l'état haut.

Le dernier prototype représente une version finale du produit. Chaque composant est soudé avec l'empreinte qui convient, ce qui permet de vérifier que tous les connecteurs sont accessibles facilement et qu'aucun composant ne se chevauche. Le design de l'antenne GSM a aussi été rajouté (et vérifié) ainsi que de la place pour venir poser l'antenne GPS.

Des photographies du dernier PCB réalisé, ainsi que des schémas électroniques complets peuvent être retrouvés sur l'annexe A de ce rapport.

Chapitre 5

Capteur de vibration

Une des caractéristiques du produit développé est sa capacité à détecter des vibrations ou un mouvement. Cet ajout permet de gérer des cycles de veille de manière intelligente. Par exemple, tant qu'il n'y a pas de vibration, le module reste en veille. Cette fonction est rendue possible grâçe à l'utilisation d'un capteur passif, que je vais maintenant présenter.

5.1 Composant utilisé

Le composant utilisé est fabriqué par "Sensolute" sous la référence "MVS0608.02". Ce composant est passif, il n'a pas besoin d'être alimenté pour fonctionner. En réalité, c'est un simple interrupteur. En effet, à l'intérieur se trouve une bille en métal et quatre plots métalliques aussi (constituant les bornes du dipôles) (figure 5.1). Lorsque le composant est secoué, la bille se déplace et fait contact avec les plots. Ainsi, le signal passe. Pour des raisons évidentes, un circuit de filtrage du bruit doit-être ajouté. Ce circuit permet ainsi de filtrer les vibrations parasites. Par exemple, si le traqueur est posé sur une voiture, stationnée près d'une voie de tramway, il ne doit pas se déclencher à cause de la vibration générée par le passage d'une rame. Le constructeur met à disposition une note d'application dans une documentation à cet effet [8].

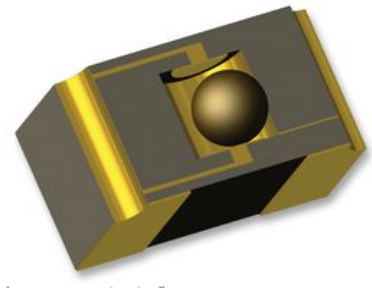

FIGURE 5.1 – Vue en coupe du capteur de vibration

5.2 Circuit de filtrage

Ce circuit possède plusieurs composants importants (figure 5.2). Tout d'abord, les résistances placées le plus à gauche (R2 et R5). Elles servent à limiter le courant qui passe dans le capteur de vibration. Un courant trop important pourrait le détruire. L'objectif est bien entendu d'avoir un courant suffisamment important pour le reste du circuit de filtrage, mais aussi suffisamment faible pour des contraintes d'économies d'énergie.

Suite à ce petit ensemble, on trouve un condensateur de liaison avec la partie de filtrage. Cette partie est articulée autour d'un condensateur (C3), d'une résistance qui lui est associée (R7), d'un transistor (Q2) et sa résistance de polarisation (R6). Lorsque le capteur va être passant (un contact a lieu, dû à une vibration), le condensateur va se charger au travers de la résistance. La tension aux bornes du condensateur va rendre le transistor plus ou moins passant. Plus il est chargé, plus le transistor sera passant et plus le signal observé sur la broche "Vibration" sera proche de 0. Si le capteur arrête de bouger, le condensateur va alors commencer à se décharger et donc le transistor sera moins passant.

Afin de changer la sensibilité du capteur, deux paramètres sont facilements modifiables.

- La capacité du condensateur C3. Plus elle sera importante et plus il mettra de temps à se charger. Il faudra donc plus de vibrations pour qu'il rende le transistor passant.
- La résistance R7. Elle fait partie du circuit de charge du condensateur. En conclusion, plus sa valeur sera élevée et plus il mettra de temps à se charger.

Enfin, de manière logicielle, il est aussi possible d'établir un filtre sur les entrées numériques. Le logiciel embarqué sur le module permet d'établir une "durée d'activité avant déclenchement d'action" au niveau de ses entrées numériques.

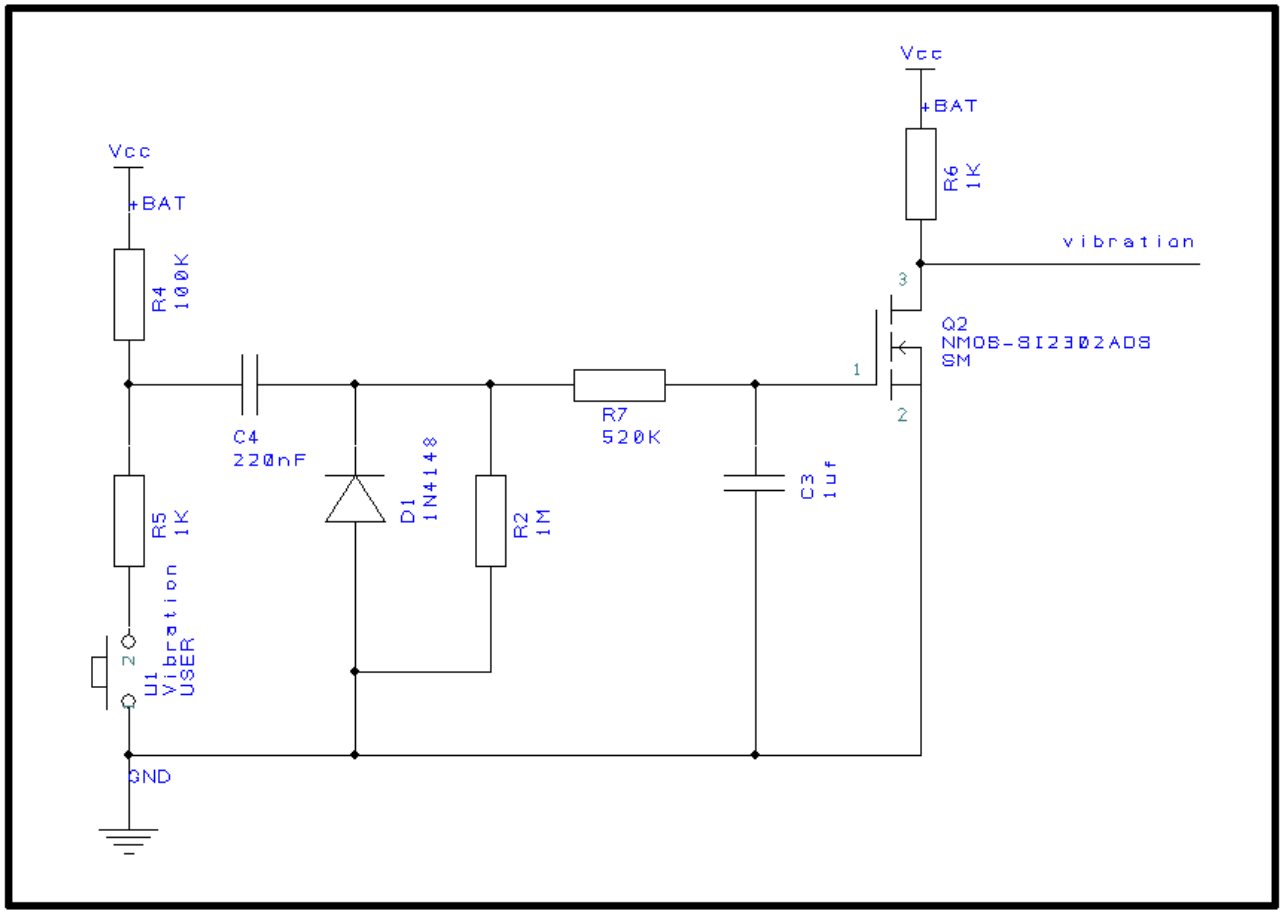

FIGURE 5.2 – Circuit de filtrage du capteur de vibration

Chapitre 6

Chargeur de batterie USB

6.1 Principe

Afin de rendre l'utilisation plus facile à l'utilisateur, il a été proposé d'intégrer un chargeur de batterie par USB. Comme dit plus tôt, nous utilisons une batterie de type li-ion. Ces batteries ont un cycle de charge particulier à respecter pour ne pas les endommager (figure 6.1).
Il faut d'abord les charger avec un courant constant, jusqu'à atteindre la tension limite de l'élément, soit 4,2 V. Ensuite, une deuxième étape prend place et charge la batterie sous une tension constante, de 4.2 V, avec un courant qui diminuera progressivement. Lorsque le courant est descendu en dessous de 3% du courant nominal de décharge, on considère la batterie chargée.

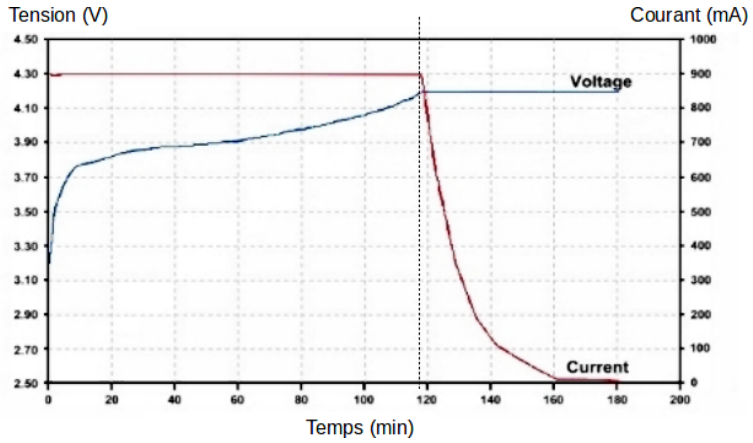

FIGURE 6.1 – Exemple de courbe de charge de batterie Li-Ion

La batterie que nous utilisons possède les caractéristiques suivantes :
- Tension nominal de 4,2 V
- Capacité de 1230 mAh
- Courant de décharge nominal de 1,2 A.

Pour la première étape de charge, un chargeur standard sans limite de courant chargera la batterie à 1C (équivalent à une fois la capacité de la batterie, dans notre cas 1,230 A). Le chargeur que nous souhaitons réaliser utilisera un port USB. Les caractéristiques électriques de ce dernier sont les suivantes :
- Tension de 5 V (± 5%)
- Courant maximum de 500 mA (dans la norme USB 2)

Afin de ne pas endommager le port ou de le placer en sécurité, la charge sera limitée à 400 mA. La charge sera donc moins rapide que celle avec un chargeur conventionnel branché sur une prise secteur.

6.2 Composant utilisé et contraintes

Le coeur de cette application est représenté par le composant MCP73855 (figure 6.2) fabriqué par le constructeur MicroChip. Ce composant est programmé de façon à déterminer en premier lieu le niveau de charge de la batterie. Ensuite, selon ce niveau, il mettra en place le cycle de charge à courant constant (phase 1) ou à tension constante (phase 2).
Tout au long de la procédure de charge, une led est allumée afin de prévenir l'utilisateur que la batterie est en train de se charger.

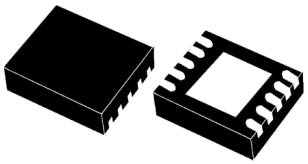

FIGURE 6.2 – Le composant MCP73855

Contrainte de consommation

L'intégration du chargeur pose plusieurs problèmes.
Tout d'abord, celui du risque d'endommager un port USB. En effet, le composant de chez MicroChip consommera au maximum 400 mA (plus quelques microAmpères pour son fonctionnement). Cependant, lorsque le traqueur est en fonctionnement, il peut consommer 150 mA en régime normal (pas d'envoi de SMS). On obtient donc une consommation de 550 mA, ce qui est trop pour de l'USB. Pour palier à cela, le branchement de l'USB active un transistor qui met ainsi la broche de reset du traqueur à la masse. Le module ne se met alors pas en marche et ne consomme que quelque microAmpères (veille).

Contrainte thermique

L'alimentation du chargeur se fait à partir d'une tension de 5 V (USB). La batterie peut-être chargée sous une tension allant de 3,3 V à 4,2 V et avec un maximum de 400 mA. Comme précisé dans la documentation technique du composant [6], ce dernier est linéaire.
Cela signifie que dans le pire des cas que le composant doit dissiper 680 mW.

$$P = (V_{USB} - V_{Bat}) * I_{Charge} = (5 - 3,3) * 0.4 = 680mW$$

Or ce composant n'est vendu que dans un boîtier de type QFP. Ce boîtier carrée mesure 3 mm de côté. La dissipation thermique ne peut donc pas se faire entièrement par ce dernier. Afin de l'améliorer, on a laissé un peu de cuivre du plan de masse sous le composant, puis, à côté de ce dernier, placé plusieurs vias connectés au plan de masse sur la couche inférieure. Ainsi, la chaleur se diffuse mieux et le composant peut fonctionner de manière optimale. Si la chaleur n'était pas évacuée, le composant finirait par se mettre en sécurité et le chargement de la batterie serait fait à une intensité moindre (mode de sécurité) voire même complètement stoppée.

Troisième partie

Informatique embarquée

La dernière partie du projet est représentée par le module qui sera embarqué sur le matériel à surveiller ou sur la personne à suivre. Ce module est programmé en C et embarque un programme de base fourni par le fournisseur. Pour faciliter le développement, une platine de développement à été utilisée, interfaçant ainsi toutes les entrées/sorties (figure 6.3). Comme expliqué ci-dessus, un ensemble de fonctions de base est proposé. Nous aurions donc pu l'utiliser ainsi mais, d'origine il n'existe aucune gestion intelligente de l'alimentation, ce qui rend son fonctionnement quelque peu inefficace lors de l'utilisation avec une batterie. Mon rôle ici a donc été d'implémenter des modes de fonctionnements permettant de garder l'ensemble de base, qui est très efficace et fonctionnel, mais aussi des modes de veille, permettant de fonctionner avec une batterie le plus longtemps possible.

Afin de gagner du temps sur l'exécution et de faire des économies en terme de nombre de SMS, j'ai dû aussi modifier la configuration par défaut du traqueur. La configuration des entrées/sorties est ainsi faite automatiquement ainsi que tous les paramètres relatifs à la gestion du convertisseur analogique.

Le développement sur le module se fait à l'aide de l'API fournit par le constructeur [5]. Cette banque de fonctions permet d'accéder facilement à de nombreux éléments tels que la mémoire flash, le GPS, la voie série ou d'autres matériels. Sans cet ensemble de fonctions, il serait très difficile de développer de nouvelles choses. En effet, le distributeur du module utilise un système d'exploitation embarqué sur ce dernier (système ECOS). S'il ne fournissait pas d'API, il m'aurait fallu apprendre à utiliser ce système avant de pouvoir produire des fonctions simples (et donc perdre énormément de temps).

Enfin, afin de garantir une exécution optimale et un gain de temps en terme de développement, j'ai réussi à obtenir près du constructeur le code-source de base utilisé. Ainsi, je peux toujours utiliser l'ensemble des commandes AT réalisées en amont. Cependant, n'ayant pas fait la formation proposée par leur service, je n'ais aucun accès à leur service technique en cas de problème lié au développement.

Note : Dans les extraits de codes suivants, on retrouve la variable "TabIdent". Cette variable est issue d'une structure est une variable globale définie par le constructeur. Elle regroupe de nombreuses informations, allant des configurations des entrées/sorties jusqu'aux paramètres d'acquisitions régulièrs.

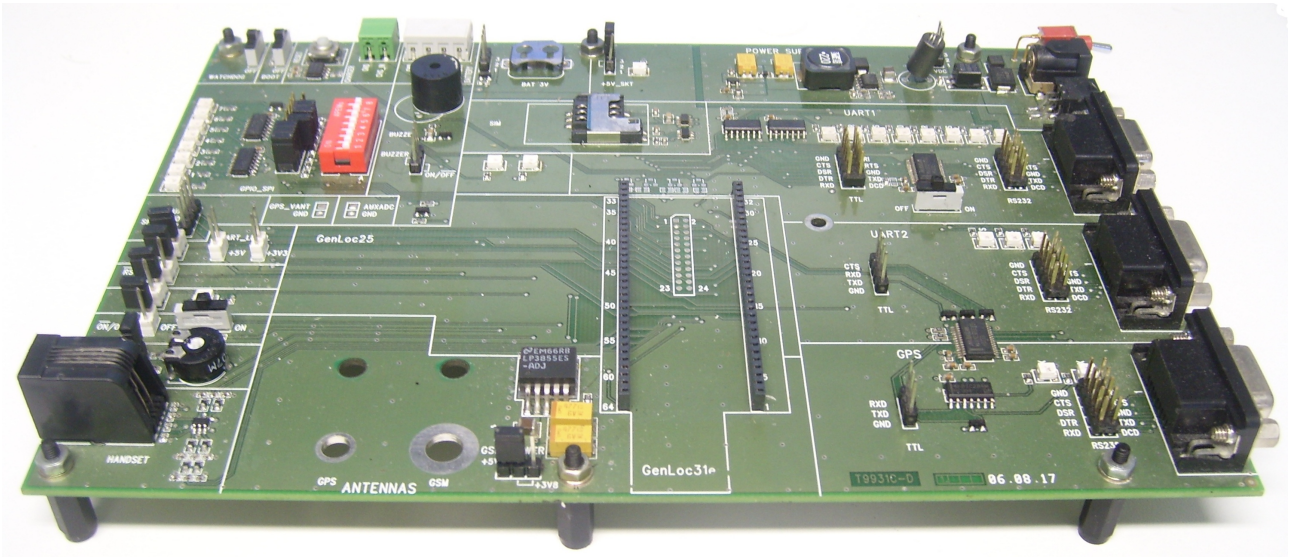

FIGURE 6.3 – Plate-forme de développement du module

Chapitre 7

Chargement de la configuration

Pour gagner du temps au démarrage de l'application et éviter des envois de SMS inutiles, les paramètres sont chargés depuis un espace mémoire non volatile du module (mémoire flash). Á chaque fois qu'un paramètre utile à l'application est modifié, son état est enregistré dans cette mémoire. Si l'application démarre pour la première fois (la mémoire flash n'a pas été paramétrée), des valeurs par défaut sont chargées :

- Filtre logiciel du bouton de SOS : 3 secondes
- Filtre logiciel du capteur de vibration : 100 millisecondes
- Durée d'un cycle : 5 minutes
- Réveil sur détection de mouvement désactivé

Voici le code permettant cette mise par défaut :

```
TabIdent.IO_int_prg[0]= 1;  //immediat pour le capteur de vibration
TabIdent.IO_int_prg[1]= 30; //3 secondes pour le bouton SOS
[...]
nbcycles = 0;
checkmouvement = 0;
dureecycle = 5;
```

Si le module a déjà été mis sous tension auparavant, l'application utilise la mémoire flash et les valeurs stockées dedans. La partie suivante présente donc les méthodes d'accès à la mémoire flash en lecture et en écriture.

7.1 Lecture/Écriture en mémoire flash

Afin de pouvoir sauvegarder/charger des paramètres utiles à l'application, le constructeur nous fournit des fonctions de base. Ces fonctions permettent de :

- Initialiser l'espace mémoire à réserver
- Sauvegarder des données
- Charger des données
- Supprimer un espace mémoire de sauvegarde

Cette partie va maintenant présenter l'intérêt de ces différentes fonctions ainsi que leur utilisation.

7.1.1 Initialisation de la zone réservée

Lors de son démarrage, le module va initialiser toutes les variables à leurs valeurs de défaut. Ensuite, le programme fait appel à une fonction de démarrage de la mémoire flash. Si la flash a déjà une (des) adresse(s) d'allouée(s), alors dans ce cas les valeurs des variables stockées sont lues et remplacées. Si la flash n'a aucune adresse de prête (le "Handler" n'a jamais été crée), alors on le crée puis stocke l'état des variables (donc leurs valeurs par défaut). Les fonctions appelées sont les suivantes :

```
flash_start_flash();   // demarre la flash (constructeur)
flash_start_myflash();  // demarre la flash (mon application)
```

La seconde fonction a été rajoutée ensuite, afin de stocker les variables rajoutées pour l'application modifiée. Son comportement est le même que la première, mais l'adresse mémoire en flash qui sera utilisé est différente. Voici les grandes commandes permettant le paramétrage de la flash (dans le fichier source flash.c) :

```
void flash_start_myflash()
{
  s16 flashSubsResponse=-1;
  // Verifie l'existence du Handle "MyFlash"
  if(egm_flhGetIDCount("MYFLASH_handle") == EGM_RET_ERR_UNKNOWN_HDL)
  {
    // On creer le Handle
    while (flashSubsResponse != OK)
    {
      flashSubsResponse = egm_flhSubscribe("MYFLASH_handle", 4);
      if (flashSubsResponse != OK) //probleme de creation
      {
        ascii bufDebug[100];
        sprintf(bufDebug,"Erreur Myflash : %d\r\n",flashSubsResponse);
      }
    }

    // On enregistre les valeurs par defaut en flash
    flash_write_myflash(1,sizeof(nbcycles),&nbcycles);
    flash_write_myflash(2,sizeof(checkmouvement),&checkmouvement);
    flash_write_myflash(3,sizeof(dureecycle),&dureecycle);
  }
  else //Sinon on charge les valeurs precedemment enregistrees
  {
    flash_read_myflash(1,sizeof(nbcycles),&nbcycles);
    flash_read_myflash(2,sizeof(checkmouvement),&checkmouvement);
    flash_read_myflash(3,sizeof(dureecycle),&dureecycle);
  }
}
```

7.1.2 Lecture et écriture

Dans le code précédent, on peut voir deux lignes qui se répètent et se ressemblent :
- flash_read_myflash()
- flash_write_myflash()

Ces deux fonctions ont une signature et un fonctionnement similaires. Comme leurs noms l'indiquent, elles servent respectivement à lire et à écrire dans la flash, avec le handle "My Flash". Les arguments qu'elles utilisent sont les suivants :
- Le numéro de la position dans la flash de la variable à lire
- La taille de la mémoire à lire (en nombre d'octets à lire)
- Un pointeur vers la variable à éditer

Ces deux fonctions utilisent ensuite le même code que les fonctions fournies par Erco & Gener, à la seule différence qu'elles utilisent le secteur flash "My Flash" au lieu du secteur par défaut.

7.2 Initialisation

Comme expliqué précédemment, lorsque le microcontrolleur démarre, il charge les valeurs par défaut puis ensuite depuis la mémoire flash. Ceci constitue l'initialisation des valeurs de base. Cependant, elle n'est pas suffisante. Une initialisation complète se passera en deux temps. Tout d'abord, les valeurs de base sont chargées, puis les entrées/sorties sont réglées de manière physique. Ensuite, le module se place dans le bon mode de fonctionnement (acquisition régulière ou non). Ces différentes étapes constituent l'initialisation automatique. Ensuite, pour que le module soit utile, il a besoin de connaitre un numéro de téléphone auquel envoyer les SMS de position. Cette deuxième étape est l'initialisation par SMS.

7.2.1 Initialisation automatique

Cette étape est liée à la lecture de la mémoire flash. Lorsque cette dernière a lieu, on met en place les entrées numériques pour le bouton de SOS ainsi que pour le capteur de vibration (ainsi que la durée de leurs filtres respectifs). Ensuite, cette fonction rappelle depuis la mémoire le dernier mode d'acquisition utilisé et ses paramètres. Ce mode peut-être :

- Pas d'acquisition régulière
- Acquisition tout les *xx* mètres (*xx* de 1 à 65535 mètres)
- Acquisition toutes les *yy* heures et *zz* minutes

Dans ce dernier cas, la variable *nbcycles* est calculé afin de savoir si un envoi de position doit-être fait durant le cycle. Ce cas sera présenté dans le chapitre suivant traitant des cycles de fonctionnement et plus précisément dans la section sur les cycles longs.

7.2.2 Initialisation par SMS

Pour pouvoir fonctionner correctement, le traqueur à besoin de connaître un numéro de téléphone auquel il pourra envoyer les messages de positions. Typiquement, ce numéro de correspondant sera celui du modem branché sur l'ordinateur exécutant l'interface de supervision. Si le traqueur n'a jamais reçu de commande lui donnant un numéro de correspondant, il ne pourra pas envoyer de SMS de localisation.

Pour régler ce paramètre et optimiser le nombre d'envoi de SMS, une commande AT a donc été rajouté. Cette commande s'appelant *AT+INIT* reçoit en paramètre une chaîne de caractère représentant le numéro de téléphone avec lequel communiquer. En réponse, le module renvoie un SMS avec tous les paramètres courant de l'application (détails des filtres, durée d'un cycle...). Ainsi, lorsque l'interface reçoit la réponse avec tous les détails, elle peut ajuster les différents composants de l'objet *Tracker* concerné.

L'utilisation d'une commande AT personnalisée permet de se passer de nombreux échanges de SMS. En effet, avant d'éditer l'application embarquée il fallait un échange de six messages (donc 12 au total) pour avoir tous les détails de la configuration. Cet échange pouvait prendre jusqu'à deux minutes pour se terminer. Maintenant, l'échange ne prend que deux SMS (un pour l'envoi et un pour la réception) et est terminé en moins de 30 secondes. Cela permet une économie financière (si l'envoi de SMS n'est pas dans un format "Messages illimités") mais surtout une économie de batterie. En effet, la communication GSM est très énergivore et donc doit être limitée au minimum pour garantir une durée de vie maximum de la batterie.

Une fois le chargement des variables effectué et le message d'initialisation traité, on peut considérer que le module est pleinement opérationnel. On peut donc faire l'utilisation d'un fonctionnement par cycle, afin de gérer au mieux l'énergie de la batterie. Ces modes de gestion sont présentés au chapitre suivant.

Chapitre 8

Cycles de fonctionnement

Dans un but d'économie d'énergie, le fonctionnement "non-stop" initial de l'application a été remplacé par un fonctionnement "cyclique". Cette partie va maintenant illustrer cette méthode et montrer pourquoi elle s'avère plus efficace pour gérer la batterie dans notre cas.

8.1 Principe et organigramme du fonctionnement

Le principe du fonctionnement par cycle est de fonctionner un temps minimum sur un cycle d'une durée connue. Pour un cycle définit de x minutes, le traqueur mettra par exemple y minutes à effectuer toutes les opérations demandées. Ensuite, une fois les opérations terminées, il se mettra en veille pour compléter le temps $z = x - y$. Plus ce temps z sera élevé et plus long sera la durée en veille donc meilleur sera la consommation d'énergie.

Un cycle commence toujours par les mêmes opérations. Le module démarre, s'initialise (charge les variables sauvegardées) puis vérifie la présence de message au bout de 45 secondes [1]. Suite à cela, si aucune acquisition de position n'est nécessaire, le cycle dit "court" se termine et le module part en veille. Si une acquisition de position est nécessaire (acquisition ponctuelle ou régulière) un cycle dit "long" commence. Le module vérifie la validité du fix GPS toutes les 10 secondes et envoie la position si elle est correcte. Si au bout de trois minutes aucun fix correct n'est recupéré, le module envoie une trame vide, permettant d'assurer à l'utilisateur que le module fonctionne toujours mais que le GPS n'a pas réussi à capter.

En résumé, un cycle de fonctionnement total peut être résumé de la manière suivante :

$$Cycle = T_{Cycle(court\ ou\ long)} + T_{Veille}$$

Et la durée de veille peut-être définie comme ceci :

$$T_{Veille} = DureeDeCycle - T_{Cycle}$$

Par défaut, la durée d'un cycle $DureeDeCycle$ est fixée à 5 minutes.

Ce fonctionnement est résumé sur l'organigramme 8.1 page suivante.

1. Cette durée a été choisie empiriquement. Elle garantie que la puce GSM est bien relié au réseau et donc que les messages ont bien été reçues

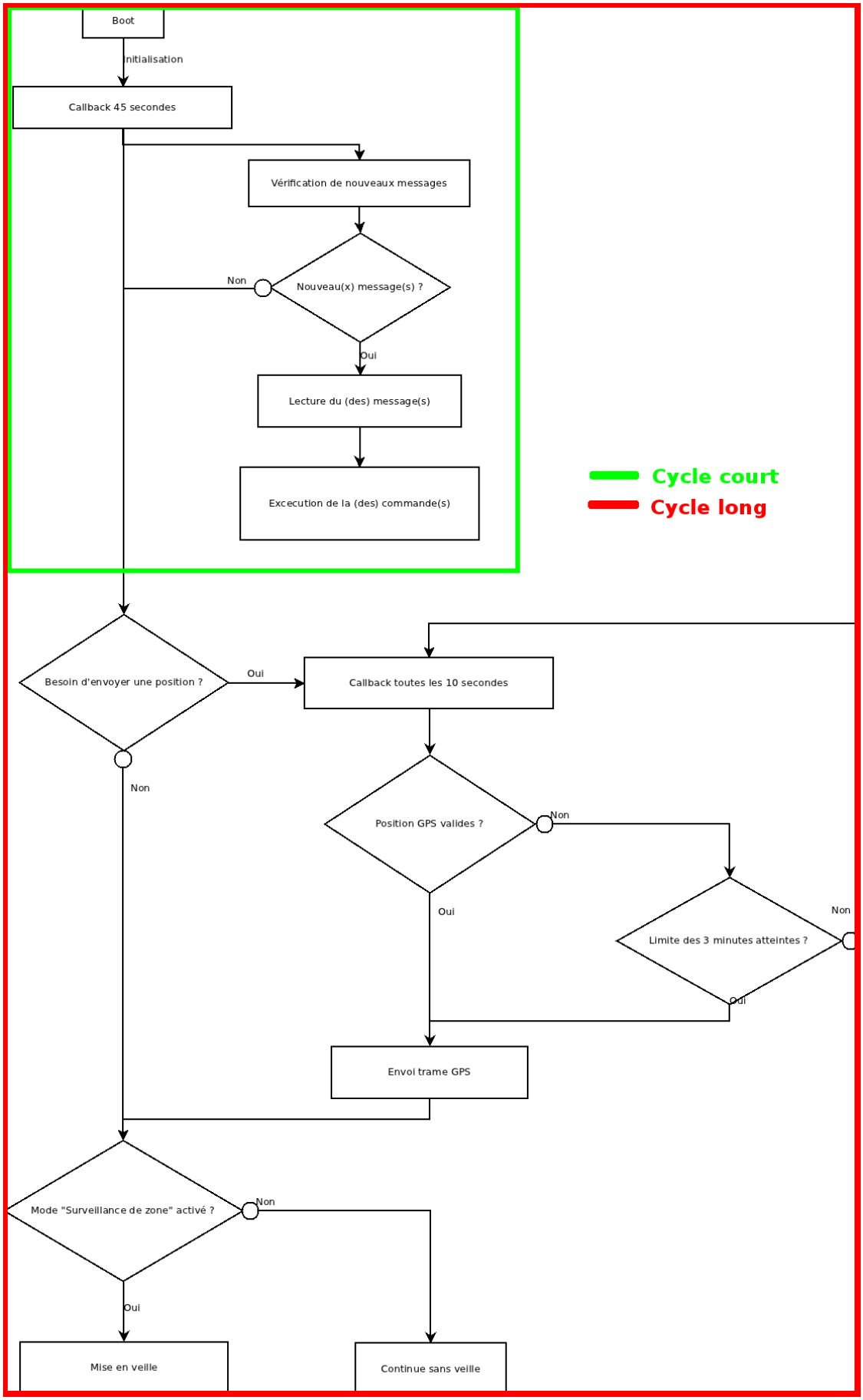

FIGURE 8.1 – Organigramme du fonctionnement des cycles

8.2 Cycle court

Comme vu ci-dessus, le but de cette partie de cycle sert à déterminer si des messages ont été reçus et surtout s'il faut envoyer une position. Après le démarrage du traqueur, le circuit GSM va chercher à obtenir une connexion au réseau. Ensuite, au bout de 45 secondes plusieurs cas peuvent se produire :

- La carte sim a besoin d'un code pin et ce dernier n'est pas valide ⇒ mise en veille
- La connexion au réseau à échouée ⇒ mise en veille
- Des messages sont reçus (SANS demande de position) ⇒ Exécution des commandes contenues puis réponse puis mise en veille
- Des messages sont reçus (AVEC demande de position) ⇒ Exécution des commandes contenues puis réponse puis départ de cycle long

La recherche de message se fait de manière similaire (même commande AT) à celle utilisée par le modem sur l'interface de supervision. Cependant, le support étant différent, l'implémentation est quelque peu différente. Ainsi, chaque commande AT exécutée sur la voie série du module GSM recevra une réponse qui sera interpretée par une fonction dite de *callback*. Cette fonction sera appelée sitôt que la le circuit GSM répond sur la voie série. On effectue ainsi quatre niveaux d'appels :

1. On demande la liste des messages dans la mémoire de message
2. On parse le résultat de cette commande puis lit les messages un par un s'il y en a
3. On parse le message courant pour trouver la commande et l'exécuter
4. On supprime le message lu de la memoire

Suite à la lecture des messages (si nécessaire), le traqueur se mettra en veille s'il n'a pas besoin d'envoyer de position, sinon il passera en cycle long.

8.3 Cycle long

Lorsque le traqueur doit faire un envoi de positions, il passe dans un cycle long. La durée de ce cycle n'est pas prévisible puisqu'elle dépend de l'efficacité du GPS à recevoir une position. Cette position est consultée toutes les 10 secondes et ce dans une limite de trois minutes suite au démarrage du module. Si aucune position n'est trouvée au bout de trois minutes, le traqueur se mettra en veille après l'envoi d'un message avec une trame vide (sécurité permettant d'assurer à l'utilisateur que le traqueur fonctionne toujours). Nous allons passer en revue les différentes situations du cycle long.

8.3.1 Envoi ponctuel

Lorsque l'utilisateur envoie un SMS contenant la commande "SENDPOSI", le traqueur enverra une position de manière ponctuelle (opposé à l'envoi régulier). Dans ce cas simple, le traqueur effectue le cycle expliqué précédemment. Il vérifie toutes les 10 secondes si le fix est correct. Si oui, la position est envoyée, sinon on attend de nouveau 10 secondes (dans la limite des trois minutes).

8.3.2 Envoi régulier

On retrouve deux types d'envoi réguliers. L'un se fait en fonction du temps, et l'autre se fait selon une distance. Dans les deux cas, leur traitement est un peu particulier.

Selon le temps

Ce mode permet d'obtenir une position selon un intervalle de temps prédéfini. Comme le module fonctionne selon un système de cycle régulier et à la durée connue, l'intervalle de temps entre deux acquisitions dans ce mode ne peut-être qu'un multiple de la durée totale d'un cycle. Pour cette fonction on utilise donc une variable représentant le nombre de cycle à effectuer avant la prochaine mesure. Cette variable est stockée dans la mémoire flash et est décrémentée à chaque démarrage de l'application. Si la variable vaut zéro, cela signifie que l'on est dans un cycle ou la position doit être envoyée. La variable sera donc recalculée afin de redéfinir le nombre de cycle à faire sans envoyer de position.

Exemple :
- Durée d'un cycle : 10 minutes
- Faire une acquisition toutes les heures (60 minutes)

On aura donc un nombre de cycle entre deux mesures qui sera de :

$$N_{Cycles} = \frac{Interval}{Duree}$$

$$N_{Cycles} = \frac{60}{10}$$

$$N_{Cycles} = 60\ minutes$$

Ici, le nombre de cycles entre deux acquisitions automatique sera donc de 6 cycles.

Selon une distance

Le but de ce type d'acquisition est de faire une surveillance de zone. Lorsque le module parcourt une distance supérieure à celle donnée en paramètre, il envoie sa position. Dans ce cas très particulier, le traqueur ne peut pas être mis en veille.

En effet, afin de pouvoir mesurer une distance, le traqueur doit se "souvenir" de la position d'origine par rapport à laquelle il fait la comparaison. Dûs à certaines limitations (certaines fonctions déjà intégrées par le constructeur sont compilées en librairie dont le code source n'est pas accessible), cette fonction du traqueur est devenu problématique suite aux diverses modifications effectuées sur le code pour rendre possible l'utilisation de cycle.

Après de nombreuses recherches cette fonction n'a pas pu être remise en service pour l'instant. Une communication avec le constructeur serait peut-être nécessaire pour se renseigner sur la méthode de fonctionnement de cette fonction et pouvoir intégrer cette dernière au fonctionnement par cycle développé.

Bilan

Ces trois mois de stage sont maintenant achevés. Ils ont été riches sur de nombreux plans, que je vais maintenant exposer. J'exprimerais aussi en parrallèle mon ressenti vis-à-vis de tous ces aspects.

8.4 Travail effectué

Par rapport au travail effectué et à la mission proposée, je pense avoir eu une rare chance.
En effet, pour mon projet professionnel je cherchais un stage dans le domaine des systèmes embarqués. Durant ce projet, j'ai pu être confronté à tous les aspects de ce type de développement, allant de la conception "visible" par le client pour ce qui concerne l'interface graphique jusqu'à celle "invisible" en rapport à la programmation du microcontrolleur du traqueur. De plus, j'ai aussi eu l'occasion de retravailler sur de l'électronique, ce qui m'a permis de reprendre un domaine que j'avais quelque peu mis de côté depuis ma formation d'IUT. J'ai ainsi pu constater que toute expérience peut toujours servir un jour !

Au-delà d'un travail uniquement "technique", j'ai aussi pu mettre en oeuvre d'autres aspects de la formation qui ne paraissent pas si déterminant lorsque l'on est à l'école. En effet, travaillant dans une petite entreprise (5 employés), le travail d'un ingénieur est réellement pluridisciplinaire. Par exemple, il m'a été demandé de faire une étude de coût par rapport à la production du produit, partant du prix des composants ("la matière première") jusqu'au coût de production des PCBs et de l'assemblage (main d'oeuvre). Bien entendu, mon maître de stage m'a aidé pour me fournir des estimations par rapport à des projets précédemment réalisés. En effet, en tant qu'étudiant ce sont des contraintes auxquelles nous ne sommes pas souvent confronté et il n'est pas évident d'avoir une idée des frais qui sont très spécifiques.

Un autre aspect très intéressant et "inattendu" a été celui de la présentation à des clients. En effet, il n'était pas rare que des clients de l'entreprise (actuels ou futurs) rendent visite au directeur. Il s'en suivait souvent une présentation des différents projets en cours. J'avais donc un rôle de vulgarisation de mon travail, ce qui est très intéressant.
De plus, il arrive que certains membres de la société participent à des salons. Durant ces derniers, les projets sont présentés au travers d'un diaporama powerpoint. J'ai donc préparer une présentation pour mon projet, en suivant les demandes du directeur. C'est un travail intéressant, nouveau, permettant de se rendre compte d'une autre réalité du projet : c'est un futur produit commercial.

Par rapport à ce côté "multi-discipline", il ne me restait que la facette de l'encombrement mécanique et la mise en boîtier pour considérer avoir "tout parcouru". Durant le développement électronique, j'ai fait attention à la gestion de l'espace, en surveillant l'encombrement et optimisant les positions des différents composants pour obtenir un traqueur homogène et aussi petit que possible. Cependant, je n'ai pas eu à me soucier de la recherche d'un boîtier à utiliser et je n'ai pas pu faire de test d'un produit "totalement fini".

8.5 Méthodes de travail

Durant ce stage, un maitre mot était "autonomie". En effet, il était attendu que je sois capable de réaliser mon travail "d'ingénieur bureau d'étude" en réelle autonomie. L'entreprise ayant un effectif réduit, il n'était pas possible d'avoir quelqu'un pour m'assister en permanence.
J'ai vraiment apprécié que cela se passe ainsi, car j'ai ainsi pu mettre en oeuvre les méthodes de travail comme il me semblait, permettant ainsi de me tester "sans filet".

De plus, une grande liberté m'a été laissée sur le choix des outils et de la chronologie des actions. Le fait de pouvoir travailler dans l'ordre souhaité a vraiment des avantages. Par exemple, pouvoir faire de l'électronique lorsque le développement de l'interface devient long est appréciable. Cela permet de se "changer les idées" mais aussi de prendre du recul, qui permet ainsi de se poser des questions sous un regard plus neuf et critique. Cependant, des choix d'ordres logiques apparaissent naturellement aussi et doivent-être pris en compte. Par exemple, avant de finir le développement embarqué il est bon de réaliser des prototypes électroniques pour pouvoir valider certaines fonctions. Cela évite d'aller trop loin d'un côté et de risquer de faire du travail pour rien car une partie n'est pas physiquement possible.

Pour ce qui est des techniques de travail et d'organisation, le travail en petite entreprise a un côté très agréable. Tout va vite. En effet, la prise de décision est plutôt rapide puisque les intermédiaires sont limités au minimum, tout le monde se connait et se parle. Sur un autre aspect tel que la réalisation de prototype électronique, là encore les choses se font rapidement. La machine à réaliser les cartes étant sur place, je pouvais réaliser moi-même mes prototypes. C'est un gros avantage d'un point de vue intégration car en faisant soi-même les choses on se rend mieux compte de l'ensemble. Toujours à propos de la vitesse d'action, on réalise aussi plus vite ses erreurs. En effet, au moment de souder les composants du second PCB je me suis rendu compte qu'une partie de mes empreintes de composants étaient fausses. Ayant fait le typon quelques heures avant, j'ai facilement pu revenir dessus pour le modifier et l'adapter au fur et à mesure.

8.6 Expression personnelle

Je pense avoir vraiment eu de la chance avec ce stage. En effet, le dirigeant de la société, M. Barguidjian et mon maître de stage M. Haigron m'ont accordé une grande confiance par rapport à mes compétences et une importante liberté sur les plans des méthodes de travail et de l'organisation. Grâce à ce stage, j'ai pu me rendre compte que j'avais vraiment choisi un domaine d'études qui me plait et une optique de spécialisation correspondant à ce que j'aime.

Durant l'année scolaire, j'ai eu l'occasion de réaliser plusieurs projets d'équipes (projet d'année et concours informatique). Durant ce stage, j'étais plus en solitaire. Ainsi, j'ai pu comparer les avantages et les inconvénients. L'un m'a appris qu'il était important de connaitre ses collaborateurs et savoir se repartir un travail correctement pour être efficace. L'autre m'a montré que travailler sans "contrainte humaine" n'a pas que des avantages. En effet, pouvoir déléguer une tâche pour pouvoir prendre du recul est un avantage dans certaines situations. Cependant, sans facteur humain il y a aussi un gain de temps pour toutes les prises de décisions puisqu'aucun membre d'équipe n'est là pour les nuancer. Dans mon cas, les objectifs ont été établis au début du stage comme des grandes lignes directrices, accompagnées de l'ancien projet comme exemple initial. L'autonomie a ensuite été mise en jeu, ponctuée par des échanges durant le stage afin de s'assurer que je n'étais pas trop en retard ou sur une mauvaise voie.

Ce fut donc une expérience vraiment enrichissante. De plus, pouvoir travailler dans un domaine qui m'intéresse, sur un projet moderne et innovant est quelque chose de vraiment captivant.

Quatrième partie

Annexes

Annexe A

Carte électronique de support du module

A.1 Vue schématique de la carte

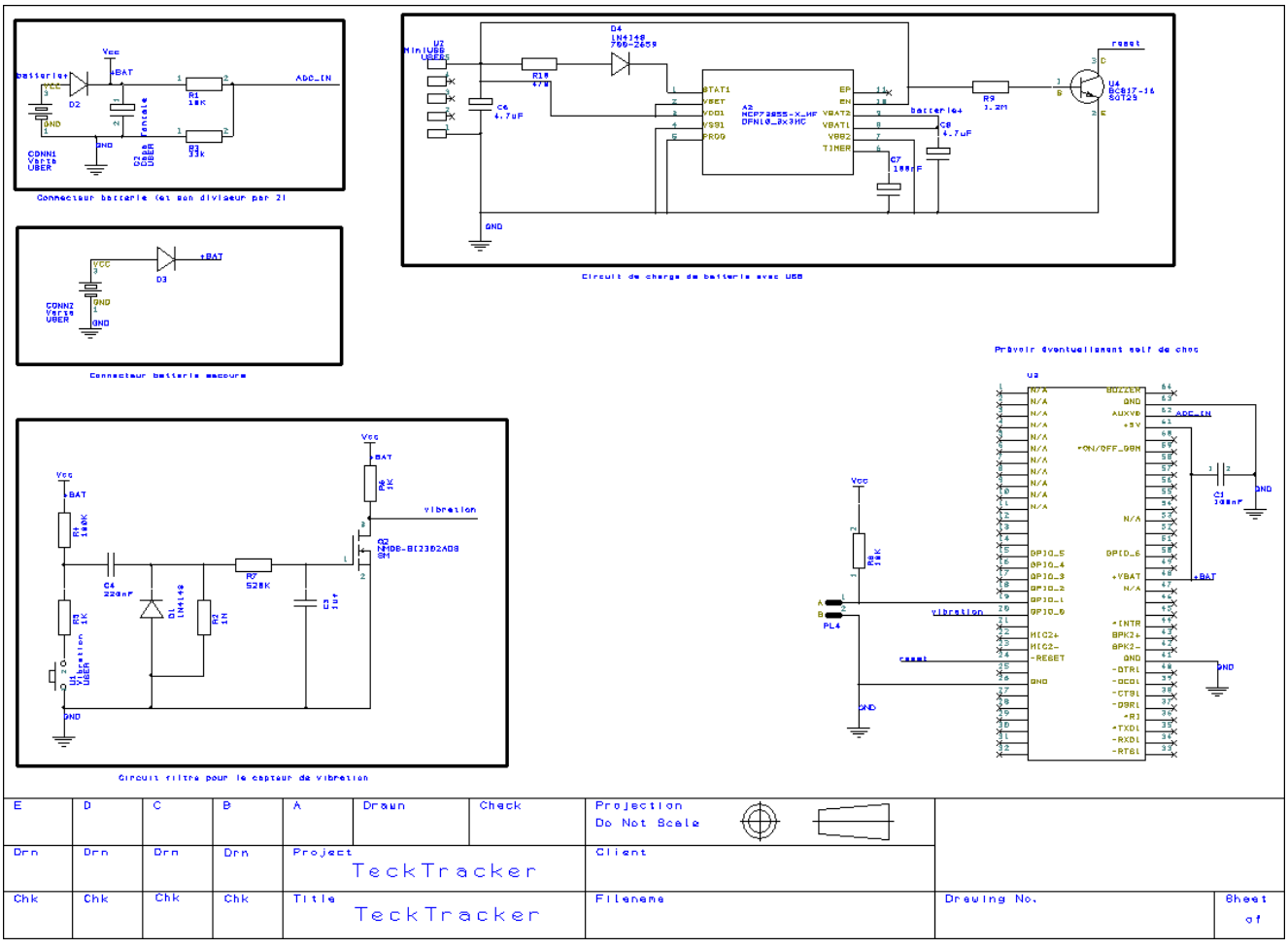

FIGURE A.1 – Schéma électronique du circuit réalisé

A.2 Vue du typon

A.2.1 Top

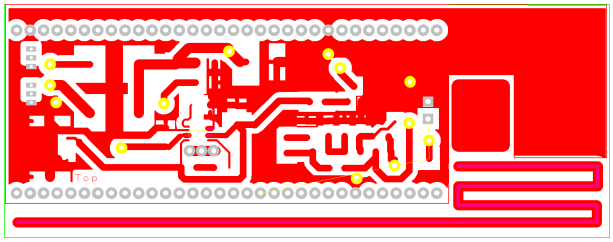

FIGURE A.2 – Vue de la face supérieure du typon

A.2.2 Bottom

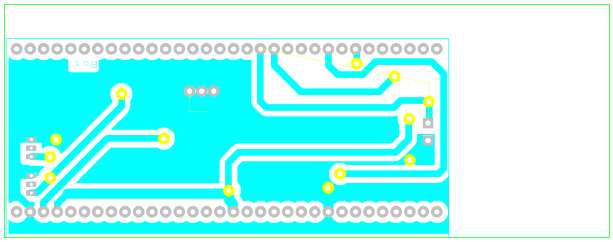

FIGURE A.3 – Vue de la face inférieure du typon

A.3 Réalisation finale

A.3.1 Top

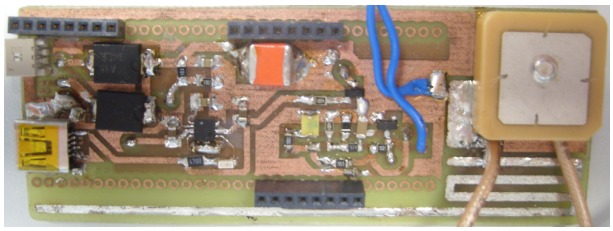

FIGURE A.4 – Vue de la face supérieure de la carte

A.3.2 Bottom

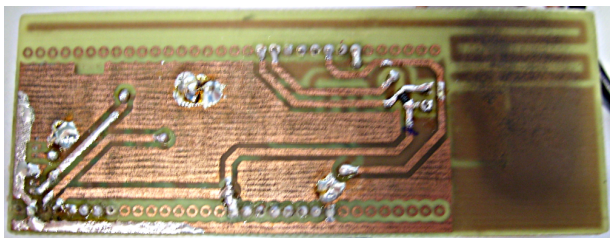

FIGURE A.5 – Vue de la face inférieure de la carte

Annexe B

Outils utilisés

B.1 Réalisation de l'interface

- Environnement de développement QtCreator (v. 2.2.1)
- Librairies Qt (v. 4.7.3)
- Librairie tiers "QextSerialPort" (Librairie pour utilisation de voie série avec Qt) [2]
- Librairie tiers "LibQxt" (v. 0.6.1) (Librairie ajoutant de nouveaux widgets graphiques à Qt) [1]

B.2 Électronique

- Logiciel de schéma et dessin de PCB "DesignSpark" [7]

B.3 Informatique embarqué

- Environnement de développement Yagarto IDE (Éclipse C++)
- Jeu de commandes UNIX émulés avec Cygwin
- HyperTerminal Windows
- Advanced Serial Port Monitor
- Free Virtual Serial Port Emulator

Bibliographie

[1] Libqxt. http ://dev.libqxt.org/libqxt/wiki/Home.

[2] Qextserialport. http ://code.google.com/p/qextserialport/.

[3] Erco & Gener. *User Guide - GenLoc OEM AOB.* Erco & Gener, 2010. EG_GenLocOEM_AOB_1008_UG_001_FR.pdf.

[4] Erco & Gener. *Command List - EaseLoc V2.* Erco & Gener, 2011. EG_EaseLoc_V2_CL_007_UK.pdf.

[5] Erco & Gener. *EGM Library User Guide (for AOB products).* Erco & Gener, 2011. EG_EGM_UG_225_UK.pdf.

[6] MicroChip. *MCP73853/55, USB Compatible Li-Ion/Li-Polymer - Charge Management Controllers.* MicroChip, 2004. MCP73853/55.

[7] RadioSpares. Designspark. http ://www.designspark.com/.

[8] Sensolute. *APPLICATION NOTE, Micro Vibration Sensor.* Sensolute, 2011. page 10.

[9] TechnologuePro. Chapitre 5 - commandes at, sept 2010. http ://www.technologuepro.com/gsm/commande_at.htm.

[10] Wikipedia. Hayes command set, jun 2011. http ://en.wikipedia.org/wiki/Hayes_command_set.

[11] Wikipédia. Le format gpx. http ://fr.wikipedia.org/wiki/GPX_(format_de_fichier).

[12] Wikipédia. Le format kml. http ://fr.wikipedia.org/wiki/Keyhole_Markup_Language.

Table des figures

1 Logo de l'entreprise "TeckniSolar SENI" 5
2 Le logo du framework Qt 9

1.1 L'interface utilisateur 10
1.2 La zone de commande des traqueurs 11
1.3 La barre de menu 12

2.1 Exemple de Modem USB 13
2.2 La carte initialement chargée 18
2.3 La carte et tous les détails affichés 19

3.1 Le module "GenLoc OEM AOB" utilisé 22
3.2 Synoptique du fonctionnement de l'ensemble 23

4.1 La batterie Li-ion utilisée 24
4.2 Schéma électronique du circuit d'alimentation 25
4.3 Aperçu des antennes utilisées 26

5.1 Vue en coupe du capteur de vibration 28
5.2 Circuit de filtrage du capteur de vibration 29

6.1 Exemple de courbe de charge de batterie Li-Ion 30
6.2 Le composant MCP73855 31
6.3 Plate-forme de développement du module 33

8.1 Organigramme du fonctionnement des cycles 38

A.1 Schéma électronique du circuit réalisé 44
A.2 Vue de la face supérieure du typon 45
A.3 Vue de la face inférieure du typon 45
A.4 Vue de la face supérieure de la carte 45
A.5 Vue de la face inférieure de la carte 45

Résumé

Ce rapport présente l'activité de stage effectué au sein de l'entreprise TeckniSolar situé à Saint-Malo, en Bretagne, entre le 2 mai et le 29 juillet 2011.

Le but du projet était de réaliser un traqueur GPS/GSM. Un traqueur permet d'effectuer de la surveillance de matériel (sensible au vol par exemple) ou de personnel (gestion de flotte...). Il est généralement composé d'une antenne GPS permettant de rechercher la position du module et d'un modem GSM servant à envoyer la position à un récepteur.
Ce projet est composé de trois parties :

La première est le développement d'une interface utilisateur pour la supervision des traqueurs. Elle est faite en C++ et avec le framework Qt.

La seconde est la réalisation d'une carte électronique servant de support au module utilisé. Elle permet d'assurer l'alimentation et sert d'interface avec les capteurs à rajouter.

Enfin, la dernière partie consiste en la reprogrammation du module embarqué afin d'ajouter de nouvelles fonctions et surtout d'ajouter des modes de mise en veille pour une gestion optimale de l'énergie de la batterie.

Une étude sur la pluridisciplinarité du travail du développeur de système embarqué sera la base de la réflexion de ce rapport. Ensuite, un bilan sera fait par rapport à cette question, puis d'un point de vue personnel par rapport à l'expérience de stage.

Mots-clés : C++/Qt, GPS/GSM, SMS, embarqué, communication