

Compte-rendu de projet IF23
Système autonome de manipulation de
données GPS

Youenn Piolet Julien Nozais Alexandre Horréard

19 juin 2013

Table des matières

1	Présentation du projet	2
2	UML	3
2.1	Diagramme de cas d'utilisation	3
2.2	Diagramme de séquence	4
3	Fonctionnement du programme	4
3.1	Fonctionnement général	4
3.2	Gestion du GPS	5
3.3	Gestion du LCD	6
3.4	Gestion de la carte SD	6
4	Mode d'emploi	8

1 Présentation du projet

Le but du projet était de créer le programme tournant sur un micro-contrôleur Arduino permettant de manipuler des données GPS.

Notre boîtier récupère les informations envoyées par le module GPS et les stockent sur une carte SD dans des fichiers. Ces fichiers peuvent ensuite être récupérés sur un PC pour être utilisés avec GPSprune, un logiciel permettant d'afficher le parcours correspondant aux données envoyées.

Un premier bouton permet de démarrer ou d'arrêter le parcours. Chaque nouveau redémarrage crée un nouveau fichier où enregistrer les données. Chaque parcours est donc enregistré dans un fichier séparé. On peut également mettre le parcours sur pause et le redémarrer ensuite, sans changer de parcours.

Le boîtier dispose d'un écran LCD qui permet d'afficher plusieurs informations. En appuyant sur un deuxième bouton on change l'affichage. L'écran peut afficher les coordonnées, la vitesse, la distance parcourue ou l'heure. Toutes ces informations sont bien évidemment mises à jour en temps réel.

Un autre bouton permet de choisir le mode d'enregistrement : soit les points sont enregistrés suivant un temps fixe (toutes les trois secondes), soit les points sont enregistrés suivant leur distance avec le point d'avant. Nous obtenons ainsi une suite de points qui ne sont pas identiques.

L'ensemble fonctionne sur pile. Au démarrage du boîtier, la batterie s'affiche sur l'écran LCD.

2 UML

2.1 Diagramme de cas d'utilisation

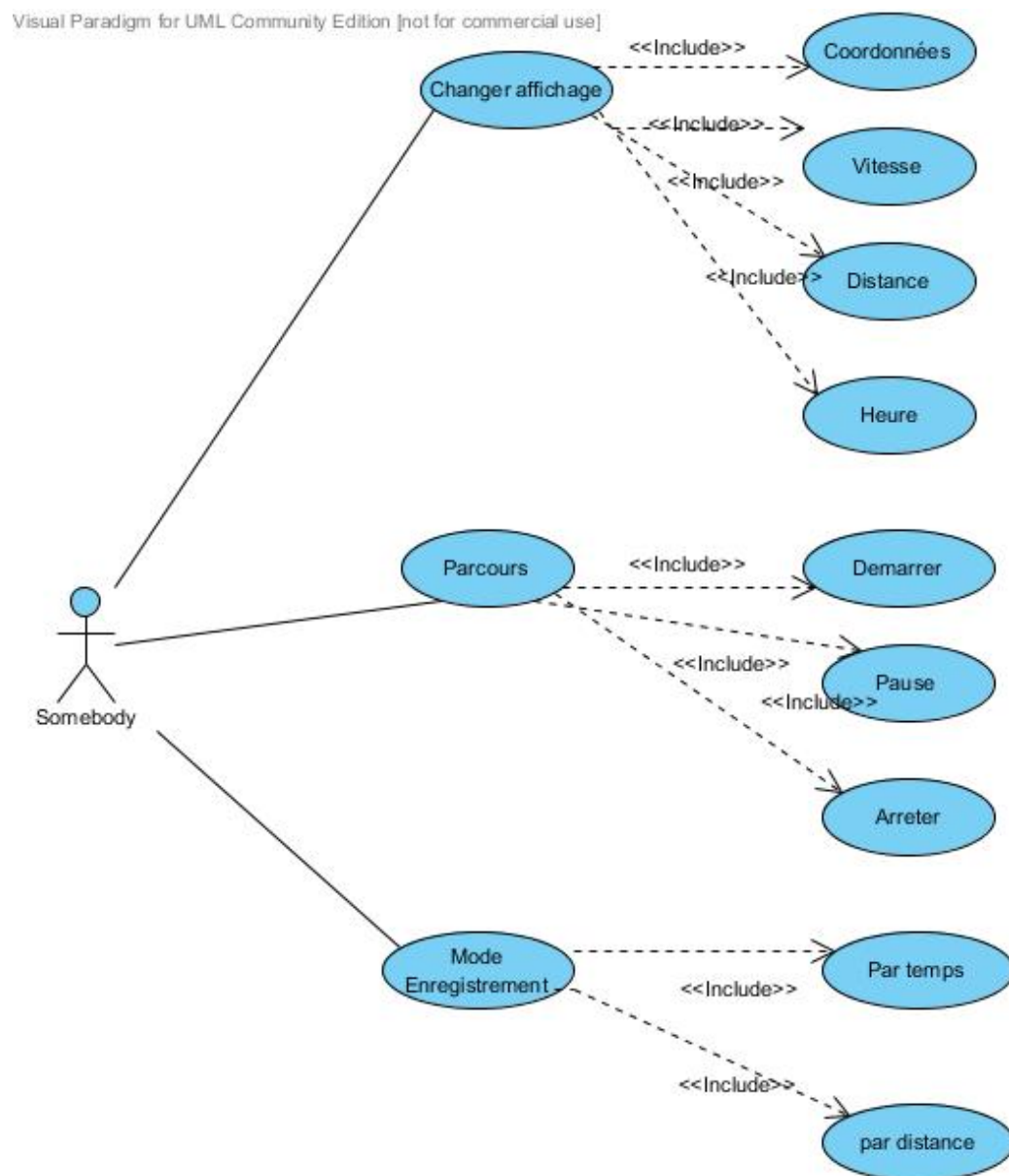


FIGURE 1 – Diagramme de cas d'utilisations

2.2 Diagramme de séquence

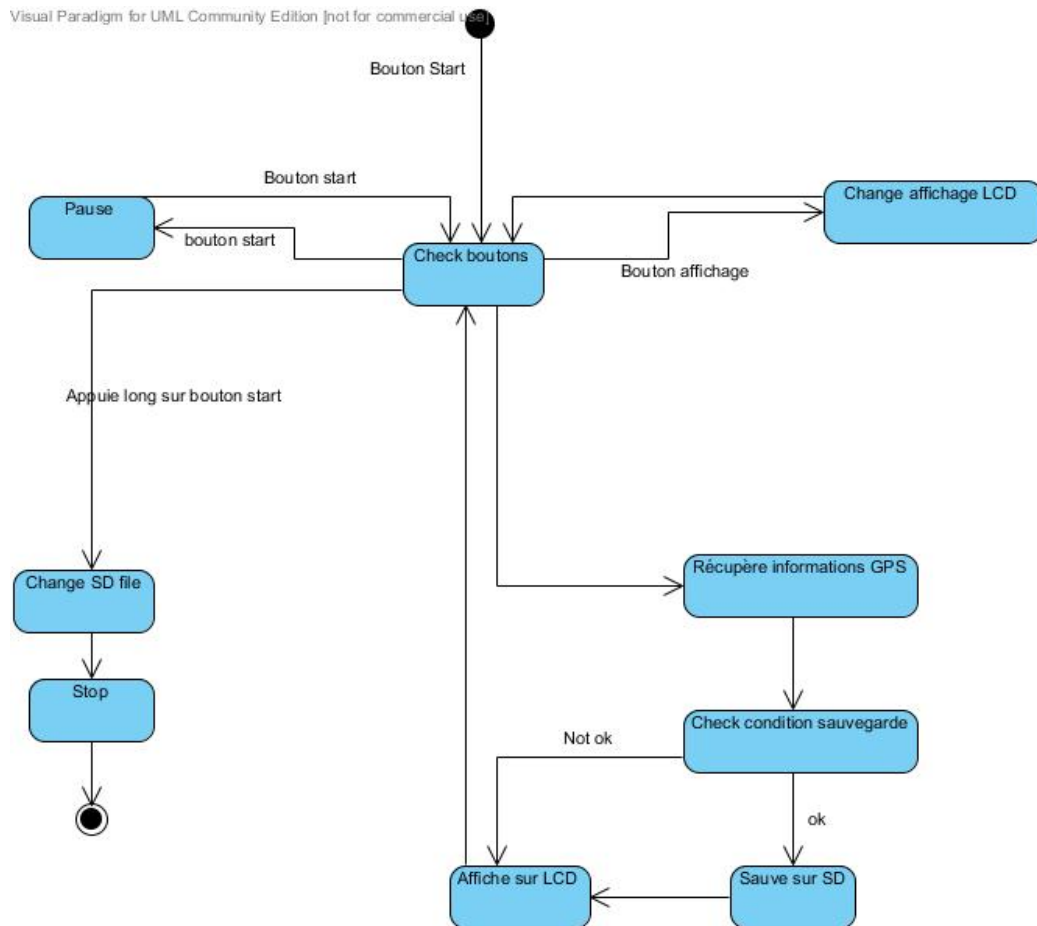


FIGURE 2 – Diagramme de séquence

3 Fonctionnement du programme

3.1 Fonctionnement général

L’affichage sur écran LCD, le système de navigation, la sauvegarde sur carte SD et la récupération des données du module GPS sont tous gérés par des classes séparées. La boucle principale ne fait qu’appeler les fonctions des instances de ces classes dans l’ordre. Cette boucle principale gère aussi le fonctionnement des boutons. À chaque passage dans la boucle le système teste si un des boutons a été appuyé et suivant quel bouton a été appuyé il

lance la methode indiqué.

3.2 Gestion du GPS

Le module GPS est géré par la classe GPSHandler. La fonction principale de cette classe est *refreshData*. Cette fonction récupère les informations du modules GPS grace à une liaison série et la fonction *encode*. Une boucle permet de récupérer tous les caractères qui arrivent de la liaison série. La fontion *encode* permet de tester si toute la trame est arrivé, en effet elle ne renvera vrai que si elle arrive à decoder la trame. Sinon la trame n'est pas complète on continue de tourner dans la boucle.

Lorsque toute la trame est arrivé, on met à jour ses attributs privés *_lat*, *_lon*, *_date*, *_time* et *_speed*. Pour gérer les problèmes dans la transmission, un *timeout* permet d'arreter la boucle si la transmission prend trop de temps.

```
// Information refreshing
void GPSHandler::refreshData(LCDhandler & lcd) {
    unsigned long timer;

5    if (_isRunning && _nss.available()) {
        // Serial Link UP
        timer = millis();
        do {
            // We try to read a full message, handling
            // transmission timeout in
10         // case of communication problems.
            int answer = _nss.read();

            // is the message fully received?
            _isReceived = _gps.encode(answer);

15         if (_isReceived) {
            _gps.get_position(&_lat, &_lon, &_fixAge);
            // Time format: hhmmsscc
            // Date format: jjmmaa
            _gps.get_datetime(&_date, &_time, &_fixAge)
20             ;

            // Converting speed
            _speed = _gps.speed() * KNOT_CONV;

25         // Stats : nb chars fed to the gps / nb
            // sentences processed / nb failed checksum
```

```

30         tests
            _gps.stats(&_chars, &_sentences, &
                _failed_checksum);
        }
    } while (_nss.available() && !_isReceived && (
        millis()-timer < TIMEOUT));
    if (millis()-timer > TIMEOUT)
        lcd.notify("Timeout", "ERR");
}

```

Listing 1 – refreshData

Par ailleurs, on dispose des méthodes permettant de récupérer ces valeurs. La boucle principale appelle donc à chaque passage *refreshData* et on utilise les fonctions *get* pour récupérer les dernières valeurs mises à jour.

3.3 Gestion du LCD

Le LCD est géré par la classe *LCDHandler*. Les différentes méthodes servent à faire de la mise en forme et afficher de manière claire les informations. Par exemple la fonction *notify* permet d'afficher pendant quelques secondes une information particulière sur deux lignes.

```

void LCDHandler::notify(String s, String type) {
    cls();
    _lcd.print("[ " + type + " ]");
    _lcd.setCursor(0, 1);
5    _lcd.print(s);

    // "A while"
    _isAvailable = false;
    _time = millis();
10 }

```

Listing 2 – notify

3.4 Gestion de la carte SD

Toute la gestion de la carte SD se trouve dans la classe *SDHandler*. L'initialisation de la carte se trouve dans la méthode *init* et non pas dans le constructeur pour des raisons d'initialisation de la mémoire.

Le système enregistre chaque parcours dans un fichier différent dont le nom est incrémental (*gpslog01.txt*, *gpslog02.txt*, etc). Pour savoir quel est

le fichier à utiliser, notamment lorsque l'on redemarre le système, un autre fichier contient le numero du fichier à utiliser. Ce fichier est créé s'il n'existe pas, sinon une seule valeur est écrite dedans à chaque changement de fichier.

En plus de la methode d'initialisation, la classe à deux methodes : *writeCoordinates* et *changeFile*. La première écrit dans le fichier courant les valeurs passées en paramètres et la deuxième change de fichier (et met à jour le fichier de numérotation donc). On appellera cette dernière méthode lorsque l'on stoppe un parcours.

```
5  /**
   * @fn void SDhandler::changeFileName()
   * @brief Increment the name of the file and create it
   */
   int SDhandler::changeFile() {

       _logFile.close();

       _numFile = _numFile + 1;
10      sprintf (_nameFile, "gpslog%d.txt", _numFile);

       _lastFile = SD.open("lastfile", FILE_WRITE);
       _lastFile.seek(0);
       _lastFile.write(_numFile);
15      _lastFile.close();

       _logFile = SD.open(_nameFile, FILE_WRITE);
       if (!_logFile.println("latitude;longitude;date;time;
                               speed;"))
           return errWrite;
20      return 1;
   }
```

Listing 3 – changeFile

4 Mode d'emploi

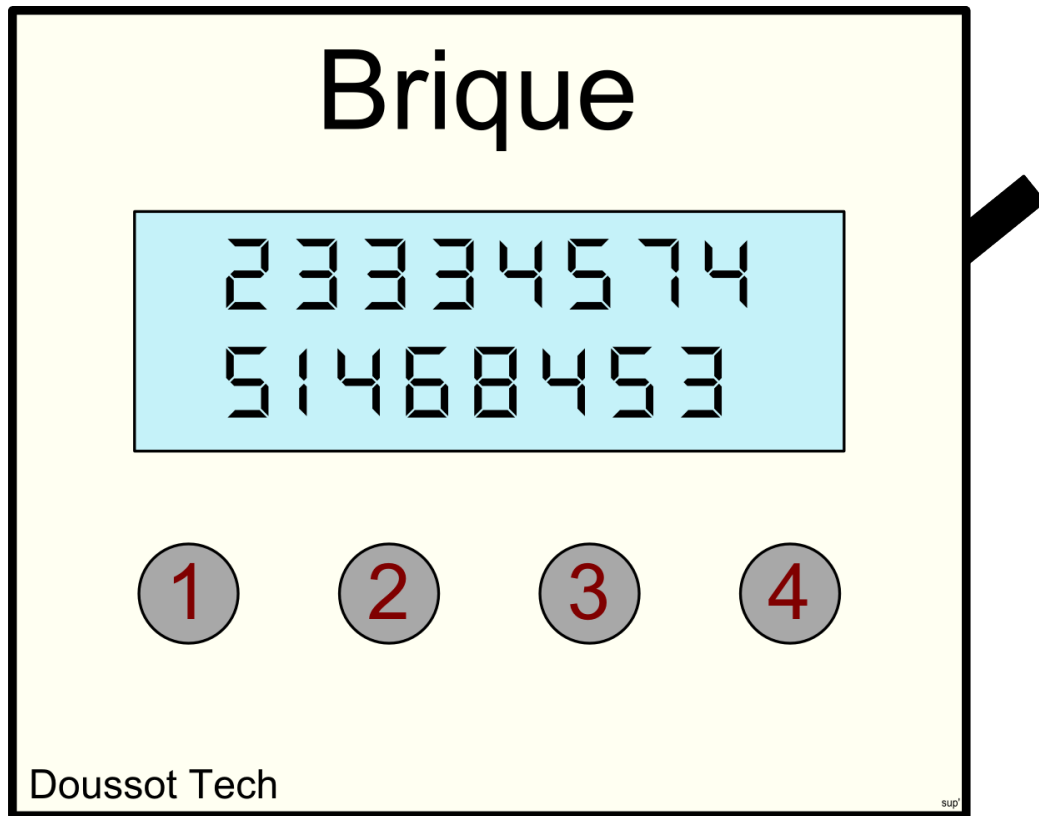


FIGURE 3 – Schéma de la brique

La brique se présente sous un boîtier doté d'un switch sur le côté, un écran LCD et quatre boutons, qu'on nommera bouton 1, 2, 3 et 4.

Le switch fait passer le circuit sur un fonctionnement par pile. L'allumant à l'occasion s'il n'est pas branché.

Une première pression sur le bouton 1 lance la capture de donnée. Une deuxième pression le met en pause. Appuyer longuement sur le bouton arrête la capture.

Chaque pression sur le bouton 2 fait defiler l'affichage. Dans l'ordre, l'écran affiche :

- La distance parcourue
- Les coordonnées
- la vitesse

Le bouton 3 permet de changer le mode de capture.
Le bouton 4 ne permet rien. Rien ! RIEN !