Acquisition et dissémination de trames GPS à des fins de cartographie libre

J.-M. Friedt (friedtj@free.fr) & É. Carry (milou@achelem.org)
Association Projet Aurore (Besançon)
17 août 2006

Nous avions présenté en Mars un circuit électronique permettant à moindre coûts d'acquérir des traces de points GPS. Le traitement et la validation des données ainsi obtenues restaient cependant des tâches relativement fastidieuses puisque la phase de comparaison des données propriétaires (www.mapquest.com) avec nos données passait par une phase de programmation Postscript délicate (recherche des facteurs d'homothétie et de translation des points afin de les ajuster sur les cartes). Nous allons ici présenter un certain nombre d'améliorations au circuit électronique pour en réduire les dimensions et la consommation électrique, et surtout les outils appropriés au partage des traces GPS acquises et à leur présentation : partage avec UPCT, fusion avec les données disposnibles dans Google Earth et Google Maps, utilisation de la librairie M_MAP de Matlab et de GRASS pour la gestion géographique d'information.

1 Récepteurs GPS utilisés

Nos objectifs dans la réalisation de ce projet sont de quatre ordres : réduire les coûts par rapport aux récepteurs commerciaux, réduire les dimensions, réduire la consommation afin d'atteindre une autonomie maximale, et maîtriser l'ensemble des éléments du projet, tant sur le plan matériel que logiciel.

Un récepteur GPS est un composant excessivement complexe dont la réalisation dépasse largement les compétences des auteurs. Nous avons donc acquis des modules de réception OEM auprès de Lextronic (www.lextronic.fr) : Laipac UV40 basé sur un composant Sony [1] et Laipac PG31 [2]. Le premier module, plus ancien, coûtait une centaine d'euros et consomme de l'ordre de 30 mA. La baisse soudaine de son prix laisse supposer qu'il sera bientôt en rupture de stock. Le second module coûte 83 euros/unité, consomme de l'ordre de 70 mA mais surtout propose le mode WAAS (États-Unis)/EGNOS (Europe) qui améliore considérablement la résolution au sol en informant notamment le récepteur au sol de satellites défectueux.

Les antennes incluent un préamplificateur et donc doivent non-seulement inclure un connecteur adapté (SSMT ou MMCX) mais aussi supporter la tension fournie par le récepteur GPS. Les anciennes antennes vendues par Lextronic (Laipac Tech GLP-1) étaient légèrement plus volumineuses que les antennes actuelles (Laipac Tech AT-65) mais capables de soulever une masse deux fois plus importante, 980 grammes pour les premières contre 500 g pour les secondes. Ce commentaire apparemment anodin a cependant une implication importante pour notre application de GPS mobile fixé à un vêtement : les nouvelles antennes nécessitent un système de fixation externe (couture, velcro) alors que les anciennes antennes pouvaient se fixer au moyen d'une pièce métallique magnétisable sous le vêtement. Un test préalable à tout achat d'une nouvelle antenne peut donc s'avérer nécessaire.

2 Améliorations du système d'acquisition

Un certain nombre d'améliorations ont été apportées au système précédemment développé : nous allons donc brièvement décrire le montage expérimental d'acquisition et de stockage des trames GPS en insistant sur les nouveautés visant à rendre ce système fiable et utilisable.

Le matériel se résume toujours aux mêmes composants : un récepteur GPS OEM fournissant des trames par RS232, un microcontrôleur recevant ces trames et se chargeant de les stocker dans une carte mémoire de stockage de masse non-volatile de type MultiMediaCard (MMC). Nous

[1] www.lextronic.fr/laipac/uv40.htm
[2] www.lextronic.fr/laipac/tf30.htm

n'entrons pas ici dans les détails du protocole de communication entre ces divers éléments qui a déja été présenté précédemment [1]. Une amélioration fondamentale cependant est le passage à des composants ne nécessitant tous qu'une alimentation de 3,3 V (contre 5 V auparavant : le récepteur GPS utilisé et décrit en détail plus bas est plus compact que le Motorola Oncore, le MAX232 chargé de générer les tensions compatibles avec le protocole RS232 est remplacé par un MAX3232) et par conséquent d'un régulateur de tension fonctionnant avec une tension d'entrée aussi basse que 3,8 V (LE33CZ [3]). Ainsi, nous pouvons alimenter ce circuit au moyen de 4 accumulateurs NiMH de 1,2 V placés en série, solution moins chère et moins dangereuse que l'utilisation d'accumulateurs LiPo.

La première contrainte à résoudre était la nécessité de reprogrammer le microcontrôleur entre les phases d'acquisition et de restitution des trames GPS. Une modification du circuit de gestion de la communication RS232 nous permet d'identifier si un câble de communication série relie le montage à un PC : nous ajoutons une résistance de pull-up entre la broche d'entrée de la communication RS232 du MAX3232 et la tension haute en sortie de la pompe de charge de ce composant (broche 2, Fig. 1).

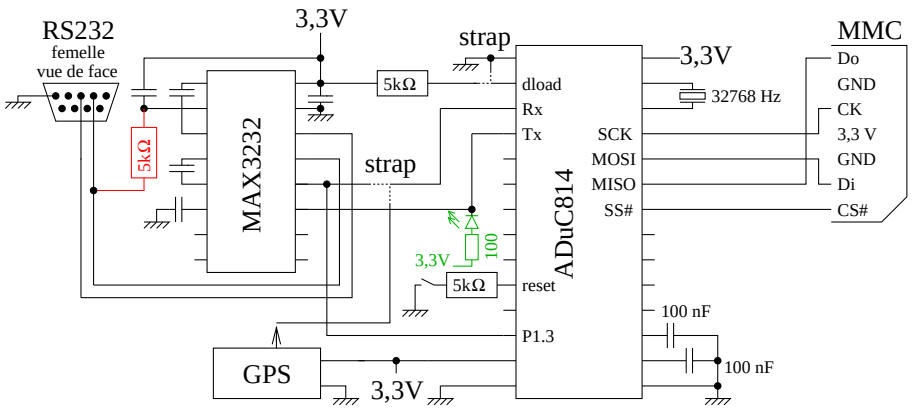

FIG. 1 – Schéma du circuit utilisé : un microcontrôleur est connecté d'une part à un récepteur GPS, et d'autre part à une carte de stockage de masse non volatile MultiMediaCard avec laquelle il communique par protocole synchrone SPI. Une résistance de tirage (en rouge sur la figure) soudée après la programmation du microcontrôleur (afin de ne pas en perturber la communication) permet d'identifier la présence d'une câble entre ce circuit et un PC et donc de détermeiner si le microcontrôleur doit initialiser une phase d'acqisition de trames GPS (absence de câble) ou de restitution des données (présence du câble détectée sur la broche P1.3/ADC1). La diode sur la ligne Tx de la liaison RS232 du microcontrôleur (en vert sur la figure) s'allume à chaque écriture d'un bloc de 512 octets sur la MMC (transmission de l'adresse du bloc nouvellement occupé) : cette capacité à valider le bon fonctionnement du montage en cours d'utilisation est apparue fondamentale à l'usage. Deux séries de straps connectent la broche DLOAD (2) soit à l'alimentation (mode programmation) soit à la masse (mode exécution une fois le programme flashé dans le microcontrôleur) ; et le port RS232 du microcontrôleur est soit connecté au PC (phase de programmation) soit au récepteur GPS (phase utilisation).

Ainsi, la polarité en l'absence de câble connecté (mode acquisition de données) est de l'ordre de +12 V, tandis qu'un câble connecté à un PC impose une tension de l'ordre de -12 V (mode restistution des données). La sortie correspodante du MAX232 est dérivée de l'entrée de communication série RxD du microcontrôleur vers une broche généraliste dont le niveau est testé au lancement du programme pour sélectionner le mode de fonctionnement approprié (noter que le MAX232 se comporte comme un inverseur en interne). Ainsi, nous flashons une seule fois un programme dans le microcontrôleur chargé des deux modes de fonctionnement, la bonne routine étant appelée en

[3] 0,80 euros chez Lextronic ou Farnell, ref. 9755349

fonction de la présence ou l'absence d'un câble.

La seconde modification majeure consiste en l'utilisation de récepteurs GPS plus récents et un peu plus coûteux que ceux présentés auparavant, mais consommant nettement moins de courant et surtout beaucoup moins encombrants. De plus, ces récepteurs fournissent des trames NMEA (décrites à `http://www.gpsinformation.org/dale/nmea.htm` par exemple) au lieu d'un flux binaire. L'avantage de cette communication ASCII est double : d'une part n'importe quel programme de communication sur port série peut être utilisé pour la récupération des données (nous utilisons `minicom` sous GNU/Linux), et d'autre part l'identification et l'élimination de trames erronées est possible et permet donc d'obtenir des traces propres. En effet, l'entropie [2] d'une trame ASCII est plus faible que celle d'un flux binaire (en d'autres termes la redondance d'information est plus élevée puisqu'une valeur de latitude ou longitude est encodée sur 11 octets au lieu des 8 octets dans un flux binaire) et il est donc possible d'identifier par des a-priori sur la forme des chaînes de caractères attendues les erreus de transmission. Dans notre cas nous éliminons toute série de données qui, après extraction du fichier NMEA tel que décrit plus bas, n'est pas de la forme

```
^[0-9][0-9][0-9][0-9]\.[0-9][0-9][0-9][0-9]\ [0-9][0-9][0-9][0-9][0-9]\.[0-9][0-9][0-9][0-9]\ ....
```

c'est-à-dire une latitude formée de 4 chiffres suivis de 4 décimales et une longitude formée de 5 chiffres suivis de 4 décimales, le tout suivi d'au moins 4 caractères formant l'altitude.

En terme de taille de fichiers, le passage d'une sauvegarde en mode binaire à une sauvegarde en ASCII se traduit bien entendu par une occupation mémoire plus importante, même si le récepteur Motorola Oncore (format binaire) sauve plus de télémétrie sur les satellites visibles que ne le fait le format NMEA. Ainsi nous avons constaté que le format binaire du Motorola Oncore génère de l'ordre de 572 KB/h tandis que le récepteur Laipac PG31 génère de l'ordre de 800 KB/h au format NMEA et le module Laipac UV40 (récepteur GPS Sony) génère de l'ordre de 1300 KB/h au format NMEA. Dans tous les cas nous constatons qu'une carte MMC de 128 MB telle que couramment disponible aujourd'hui pour moins de 20 euros contient plus d'une semaine de traces à raison de 14 heures d'acquisitions continues par jour.

Un dernier point concernant la récupération des données stockées sur une carte MMC de grande taille concerne la séparation des trajets distincts en fichiers que nous pourrons traiter indépendamment et contenant chacun une trace continue. En effet, il est probable qu'en l'absence de possibilité de recharger les batteries, nous privilégions l'accumulation de traces successives plutôt qu'utiliser l'énergie disponible à récupérer les traces accumulées dans la journée. Nous constatons que les récepteurs GPS fournissent systématiquement quelques trames d'initialisation lors de leur mise sous tension : dans le cas qui nous intéresse nous obtenons des informations du type `$TOW : 0` ou `$CHNL : 12` uniquement dans l'entête et jamais dans les trames NMEA qui suivent. L'outil `csplit` nous permet de séparer une grosse archive de traces successives accumulées sur plusieurs jours en de petits fichiers contenant chacun une trace continue :

```
csplit -f trace fichier.nmea /\$TOW/ {*}
```

coupe le fichier `fichier.nmea` tel que récupéré de la carte MMC en de petits fichiers `trace`XX (XX s'incrémente pour chaque nouveau fichier) à chaque occurence de la chaîne de caractères `$TOW`.

2.1 Circuit utilisé

Reprenons ici la procédure de programmation de l'ADuC814 tel que nous l'avions décrit auparavant [1]. Afin de permettre l'acquisition de traces multiples au cours d'un même trajet entre lesquelles le récepteur GPS est éteint pour économiser les batteries, nous stockons dans une zone de la mémoire non volatile du microcontrôleuri (adresses $BC à $BF) l'adresse du dernier bloc écrit sur MMC. Cette zone mémoire est actualisée après chaque écriture de 512 octets sur MMC. Lors de la mise en marche du circuit, une des premières tâches du microcontrôleur est de charger l'adresse du dernier bloc accédé et d'initialiser avec cette valeur le compteur d'adresse : un problème d'initialisation se pose donc. Nous avons écrit un programme chargé de mettre à zéro la mémoire non-volatile de l'ADuC814 (fonction `wrEE` du code présenté dans le tableau 2) et qui de plus nous sert à valider la connectique entre le microcontrôleur et la MMC indépendamment de tout récepteur GPS. Ce programme est présenté dans le tableau 2 et est disponible à `http://jmfriedt.free.fr/`.

```
;---------------------------------------------------------------------
HEX2ASCII:      ; converts A into the hex character representing the
                ; value of A's least significant nibble

        ANL     A,#0h0F
bbb:    CJNE    A,#0h0A,bbb+3
        JC      I00030
        ADD     A,#0h07
I00030: ADD     A,#0h30         ; '0'
        RET
;---------------------------------------------------------------------
SENDCHAR:       ; sends ASCII value contained in A to UART
        JNB     TI,SENDCHAR     ; wait til present char gone
        CLR     TI              ; must clear TI
        MOV     SBUF,A
        RET
;---------------------------------------------------------------------
SENDSPI: ; sends the content of A to the SPI port
MOV     0hF7,A          ; trigger data transfer: SPIDAT=0hF7
icispi: JNB 0hFF,icispi
CLR 0hFF ; clear ISPI
RET
;---------------------------------------------------------------------
READSPI: ; sends the content of A to the SPI port
mov     B,#8 ; max of 8 'FF' answers, otherwise continue ...
retry1: MOV A,#0hff         ; generate clocks for reception
LCALL SENDSPI

mov     A,0hF7 ; MOV A,SPIDAT: read resulting value
CJNE    A,#0hff,readend ; we only continue if the MMC answers NOT 0
        DJNZ    B,retry1
; SJMP readspi
readend:RET
;---------------------------------------------------------------------
READSPIVAL:     ; sends the content of A to the SPI port
        MOV     A,#0hff         ; generate clocks for reception
        LCALL   SENDSPI
        mov     A,0hF7          ; MOV A,SPIDAT: read resulting value
        RET
;---------------------------------------------------------------------
GETCHAR:        ; waits for a single ASCII character to be received
                ; by the UART.  places this character into A.
iciget: JNB     RI,iciget
        MOV     A,SBUF
        CLR     RI
        RET
;---------------------------------------------------------------------
DELAY:                                  ; Delays by 100ms * A
                                        ; 100mSec based on 2.097152M
                                        ; Core Clock
                                        ; i.e. default ADuC814 Clock

                MOV     R1,A            ; Acc holds delay variable
 DLY0:          MOV     R2,#0h22        ; Set up delay loop0
 DLY1:          MOV     R3,#0hFF        ; Set up delay loop1
ici:            DJNZ    R3,ici          ; Dec R3 & Jump here until R
                DJNZ    R2,DLY1         ; Dec R2 & Jump DLY1 until R
                DJNZ    R1,DLY0         ; Dec R1 & Jump DLY0 until R
                RET                     ; Return from subroutine
;_______ LIRE L'ADRESSE DE DEMARRAGE EN FLASH
rdEE:   mov     EADRL,#0h00     ; EEPROM page (1..159)
        mov     ECON,#0h01      ; read page
        mov     A,EDATA1
        LCALL   SENDCHAR
        mov     MEM0,A
        mov     A,EDATA2
        LCALL   SENDCHAR
        mov     MEM1,A
        mov     A,EDATA3
        LCALL   SENDCHAR
        mov     MEM2,A
        mov     A,EDATA4
        LCALL   SENDCHAR
        mov     MEM3,A
        ret
;_______ ECRIRE L'ADRESSE COURANTE EN FLASH
wrEE:   mov     EADRL,#0h00     ; EEPROM page (0..159)
        mov     EDATA1,MEM0
        mov     EDATA2,MEM1
        mov     EDATA3,MEM2
        mov     EDATA4,MEM3
        mov     ECON,#0h05      ; erase page
        mov     ECON,#0h02      ; write page
ret


;_______ EFFACER L'ADRESSE COURANTE EN FLASH
rmEE:   mov     EADRL,#0h00     ; EEPROM page (0..159)
        mov     EDATA1,#00
        mov     EDATA2,#00
        mov     EDATA3,#00
        mov     EDATA4,#00
        mov     ECON,#0h05      ; erase page
        mov     ECON,#0h02      ; write page
ret
```

TAB. 1 – Quelque fonctions de base utiles dans tout programme en assembleur 8051 pour ADuC814, utilisés dans les programmes qui suivent.

Une fois la mémoire non volatile initialisée, il nous reste à transférer le programme de l'application *sans modifier la mémoire non volatile* contenant l'adresse de départ de la MMC. Pour ce faire, il est fondamental de programmer la plage programme de l'ADuC814 par la commande 'C' et non 'A' qui effacerait la mémoire non-volatile pour y stocker des valeurs aléatoires. La conséquence de cette réinitialisation erronée serait une adresse de départ non nulle de la MMC, et surtout une adresse qui peut ne pas être dans la plage accessible par la MMC et donc un plantage du programme d'application qui se verrait dans l'impossibilité de stocker les trames GPS lues.

Le programme d'applicationi, présenté dans les tableaux 3 et 4, et disponible à `http://jmfriedt.free.fr`, contient deux fonctions principales appelées selon la présence ou l'absence d'un câble RS232 : en l'absence du câble (label `rs232inactif`) toute trame RS232 à 4800 bauds est capturée et stockée sur la MMC (fonction `MMCwriteBlock` qui envoie sur la MMC les données issue de la routine `GETCHAR` au niveau du label `writ2`) ; ou restitue les informations stockées sur la MMC à la vitesse de 9600 bauds si le cable est présent (label `xfer`). Dans ce second cas la lecture commence à l'adresse 0x00000000 et continue jusqu'à atteindre l'adresse du dernier bloc accédé lors de la phase d'enregistrement, tel que stocké en mémoire non volatile, Une fois toutes les données transférées au PC, la mémoire non-volatile du microcontrôleur est réinitialisée afin de permettre la capture d'une nouvelle trace (fonction `rmEE`).

Nous avons choisi d'embarquer le convertisseur de niveaux MAX3232 [4] car sa consommation est négligeable devant celle du récepteur GPS et il garantit la disponibilité d'un circuit totalement autonome pour transférer les données accumulées (fig. 2).

[4] disponible chez Farnell pour 2,08 euros/pièce

```
        SS      = 0hb5          ; P3.5 drives slave device's SS pin
        LED     = 0hb3
        CHAN    = 0             ; convert this ADC input channel (0 thru 6)

        MEM0    = 0h30          ; RAM locations: MMC address and a counter
        MEM1    = 0h31
        MEM2    = 0h32
        MEM3    = 0h33
        CNT     = 0h34

        EADRL=0hc6
        ECON=0hb9
        EDATA1=0hbc
        EDATA2=0hbd
        EDATA3=0hbe
        EDATA4=0hbf

        .area code (ABS)
        .org 0h0000
RSTirq: ljmp    MAIN ; 3 bytes
IE0irq: nop nop nop nop nop nop nop nop
TF0irq: nop nop nop nop nop nop nop nop
IE1irq: nop nop nop nop nop nop nop nop
TF1irq: nop nop nop nop nop nop nop nop
TIirq:  nop nop nop nop nop nop nop nop
TF2irq: nop nop nop nop nop nop nop nop
ADCirq: setb LED reti nop nop nop nop nop
ISPirq: nop nop nop nop nop nop nop nop
PSMirq: nop nop nop nop nop nop nop nop
TIIirq: nop nop nop nop nop nop nop nop
WDSirq: nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop

MAIN:   MOV     RCAP2H,#0hFF ; config UART for 9600 baud
        MOV     RCAP2L,#-7   ;
        MOV     TH2,#0hFF
        MOV     TL2,#-7
        MOV     SCON,#0h52
        MOV     T2CON,#0h34

        setb    ss
        mov     0h9c,#0h01      ; cfg814=0h9C
        mov     0hF8,#0h3D      ; SPICON=0hF8, CPHA=CPOL=1=>0h3D

        lcall rmEE              ; erase current flash mem adress
        lcall rdEE              ; erase current flash mem adress

;-- START INITALIZING MMC: send 10 times 0xff ----------------------
        MOV     A,#0h0ff        ; send 10 times $ff with SS#=hi
        mov     R0,#0h0a
init:   LCALL   SENDSPI
        DJNZ    R0,init
        NOP
        NOP
        NOP

;-- now reset MMC: send 0x40/00/00/00/00/0x95=cmd(00) --------------
        clr     ss              ;
        MOV     A,#0h40         ; RESET MMC
        LCALL   SENDSPI         ;

        MOV     A,#0h00         ; RESET MMC: send 4 times '0x00'
        mov     R0,#0h04        ;
rst:    LCALL   SENDSPI         ;
        DJNZ    R0,rst

        MOV     A,#0h95         ; ... and send CRC
        LCALL   SENDSPI
        NOP
        NOP
        NOP

loop2:  LCALL   READSPI
        LCALL   SENDCHAR
        CJNE    A,#0h01,loop2   ; we only continue if the MMC answers '0x01'

cmd1:   setb    ss
        MOV     A,#0hff
        LCALL   SENDSPI

        clr     ss              ;
        MOV     A,#0h41         ; command 0x01 & 0x40
        LCALL   SENDSPI
        MOV     A,#0h00
        LCALL   SENDSPI
        MOV     A,#0            ; A,#0hc0
        LCALL   SENDSPI
        MOV     A,#0            ; A,#0hff
        LCALL   SENDSPI
        MOV     A,#0h00
        LCALL   SENDSPI
        MOV     A,#0hff
        LCALL   SENDSPI

loop3:  LCALL   READSPI
        LCALL   SENDCHAR
        CJNE    A,#0h01,SPIok   ; we continue as long as MMC answers '0x01'
        SJMP    cmd1            ; otherwise MMC should answer 0x00: SPI mode OK
                                ; AT THIS POINT MMC IS IN SPI MODE
SPIok:  setb    ss
        MOV     A,#0hff
        LCALL   SENDSPI

        mov     MEM0,#0         ; RAM location 0, 1, 2 and 3 store the address
        mov     MEM1,#0
        mov     MEM2,#0
        mov     MEM3,#0
contadc:mov     A,MEM1
        LCALL   SENDCHAR
        mov     A,MEM2
        LCALL   SENDCHAR
        LCALL   MMCwriteBlock   ; write block at address 0
        inc     MEM1
        inc     MEM1
        mov     A,MEM1
        JNZ     contadc         ; jmp if accumulator is not 0
        inc     MEM2
        mov     A,MEM2
        JNZ     contadc
        inc     MEM3
        mov     A,MEM3
        JNZ     contadc

theend: SJMP    theend
;-----------------------------------------------------------------
; 1/ write command + 4 bytes writing address, multiple of 512, MSB firs
; 3/ write $FF (CRC) + wait until answer is 0
; 5/ write $FE, followed by 512 bytes and 2x$FF as CRC
; 6/ read answer and check that it finishes with 5 (ie answer=$X5)
; 7/ read while 0 -> continue when answer is non-0 (*** might be $FF
; care of not locking at point 7 ***)
MMCwriteBlock:
        clr     ss
        MOV     A,#0h58         ; command 0d24=0x18 & 0x40
        LCALL   SENDSPI
        MOV     A,MEM3          ; hi byte first ...
        LCALL   SENDSPI
        MOV     A,MEM2
        LCALL   SENDSPI
        MOV     A,MEM1
        LCALL   SENDSPI
        MOV     A,MEM0          ; ... and lo byte last
        LCALL   SENDSPI
        MOV     A,#0hff         ; CRC
        LCALL   SENDSPI

        LCALL   READSPI         ; should be 0x00
        LCALL   SENDCHAR

        MOV     A,#0hfe         ; answer for writing data
        LCALL   SENDSPI
        MOV     A,#0hff         ; DELETE ALL DATA
        mov     R1,#0h02
writ3:  mov     R0,#0hff
writ2:  LCALL   SENDSPI         ; read value
        DJNZ    R0,writ2
        DJNZ    R1,writ3

        MOV     A,#0hff         ; CRC (0xff)
        LCALL   SENDSPI         ; x2
        LCALL   SENDSPI
        LCALL   READSPI         ; write response: should finish with 5=ACK

lpwrit: LCALL   READSPI         ; read SPI ...
        CJNE    A,#0h00,endwrit ; ... while answer is 0x00
        SJMP    lpwrit
endwrit:setb    ss
        MOV     A,#0hff         ; CRC (0xff)
        LCALL   SENDSPI
        RET
```

TAB. 2 – Programme en assembleur 8051 pour ADuC814 chargé de communiquer avec une MultiMediaCard, d'effacer la mémoire non volatile de données du micricontrêur puis de remettre à $FF tous les emplacements mémoire de la MMC. Les 3 octets de poids fort de l'adresse d'un bloc effacé avec succès sont transférés sur le port série (9600, N81) : l'affichage rapide de successions de 3 octets est donc une indication du bon fonctionnement de ce programme qui valide donc les connexions électriques entre l'ADuC814 et le MAX3232 d'une part, la MMC d'autre part. Les fonctions appelées dans ce programme sont incluses dans le tableau 1.

```
        SS      = 0hb5          ; P3.5 drives slave device's SS pin
        LED     = 0hb3
        CHAN    = 0             ; convert this ADC input channel (0 thru 6)

        MEM0    = 0h30          ; RAM locations: MMC address and a counter
        MEM1    = 0h31
        MEM2    = 0h32
        MEM3    = 0h33
        dmpMEM0 = 0h34          ; RAM locations: MMC address and a counter
        dmpMEM1 = 0h35
        dmpMEM2 = 0h36
        dmpMEM3 = 0h37

        EADRL=0hc6
        ECON=0hb9
        EDATA1=0hbc
        EDATA2=0hbd
        EDATA3=0hbe
        EDATA4=0hbf

        .area code (ABS)
        .org 0h0000
RSTirq: ljmp    MAIN ; 3 bytes
IE0irq: nop nop nop nop nop nop nop nop
TF0irq: nop nop nop nop nop nop nop nop
IE1irq: nop nop nop nop nop nop nop nop
TF1irq: nop nop nop nop nop nop nop nop
TIirq:  nop nop nop nop nop nop nop nop   ; RI+TI interrupt
TF2irq: nop nop nop nop nop nop nop nop
ADCirq: setb LED reti nop nop nop nop nop
ISPirq: nop nop nop nop nop nop nop nop
PSMirq: nop nop nop nop nop nop nop nop
TIIirq: nop nop nop nop nop nop nop nop
WDSirq: nop nop nop nop nop nop nop nop nop nop nop nop nop nop

MAIN:
        MOV     RCAP2H,#0hFF ; config UART for 9600 baud
        MOV     TH2,#0hFF
        MOV     SCON,#0h52
        MOV     T2CON,#0h34

        MOV     0hEF,#0h80   ; power up ADC -- ADCCON1=0hEF
        SETB    0hAF         ; enable interrupts -- EA=0hAF
        SETB    0hAE         ; enable ADC interrupt -- EADC=0hAE

        MOV     P1,#0hF7     ; P1.3=digital input
        CLR     LED          ; turn the LED off/no AD conversion occured yet
        setb    ss
        mov     0h9c,#0h01      ; cfg814=0h9C
        mov     0hF8,#0h3D      ; SPICON=0hF8, CPHA=CPOL=1=>0h3D

;-- START INITALIZING MMC: send 10 times 0xff ----------------------

[...]
SPIok:  setb    ss
        MOV     A,#0hff
        LCALL   SENDSPI
        LCALL   rdEE
        ; on teste le port serie : si niveau haut alors ... sinon ...
        jnb     P1.3,rs232inactif       ; branch if bit is 0 (see also jb)
        LJMP    xfer

rs232inactif:
        MOV     RCAP2L,#-14
        MOV     TL2,#-14

contadc:mov     A,MEM1
        LCALL   SENDCHAR
        mov     A,MEM2
        LCALL   SENDCHAR
        LCALL   MMCwriteBlock   ; write block at address 0
        lcall   wrEE            ; sauver l'adresse qui vient d'etre remplie
        inc     MEM1
        inc     MEM1
        mov     A,MEM1
        JNZ     contadc         ; jmp if accumulator is not 0
        inc     MEM2
        mov     A,MEM2
        JNZ     contadc
        inc     MEM3
        mov     A,MEM3
        JNZ     contadc

theend: SJMP    theend

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
MMCwriteBlock:
        clr     ss
        MOV     A,#0h58         ; command 0d24=0x18 & 0x40
        LCALL   SENDSPI
        MOV     A,MEM3          ; hi byte first ...
        LCALL   SENDSPI
        MOV     A,MEM2
        LCALL   SENDSPI
        MOV     A,MEM1
        LCALL   SENDSPI
        MOV     A,MEM0          ; ... and lo byte last
        LCALL   SENDSPI
        MOV     A,#0hff         ; CRC
        LCALL   SENDSPI

        LCALL   READSPI         ; should be 0x00
        LCALL   SENDCHAR

        MOV     A,#0hfe         ; answer for writing data
        LCALL   SENDSPI
        mov     R1,#0h02
writ3:  mov     R0,#0h00
writ2:  LCALL   GETCHAR ; ... OR store value received from RS232 port
        CJNE    A,#0h40,pasA    ; ajout : on lit ADC si on a '@'=0x40
        JNB     LED,premier     ;   LED=1 => conversion done, read 2nd byte
 ; rque : on peut continuer comme ca avec d'autres compteurs pour d'autres ADC
        MOV     A,0hD9          ;   ADCDATAL=0hD9
        CLR     LED             ;   reset converter
        SJMP    sauve           ;   and save value
premier:;CLR    LED             ;   ... OR read ADC: turn the LED off
        MOV     0hD8,#CHAN      ;   select channel to convert -- ADCC0
        SETB    0hDC            ;   innitiate single ADC conversion --
                                ;   ADC ISR is called upon completion
iciadc: JNB     LED,iciadc
        MOV     A,0hDA          ;   ADCDATAH=0hDA
        sjmp    sauve           ;   garde LED a 1 pour le 2eme octet
pasA:   clr     LED             ; remet a 0 le compteur de sauvegarde ADC
        ;LCALL  SENDCHAR        ;   ici A contient le ADC au lieu de @
sauve:  LCALL   SENDSPI         ; read value
        ;MOV     A,0hD9          ; ADCDATAL=0hD9
        ;LCALL  SENDSPI         ; read value
        ;MOV     A,#0h10         ; Delay length
        ;LCALL   DELAY           ; delay 100ms
        DJNZ    R0,writ2
        DJNZ    R1,writ3

        MOV     A,#0hff         ; CRC (0xff)
        LCALL   SENDSPI         ; x2
        LCALL   SENDSPI
        LCALL   READSPI         ; write response: should finish with 5=ACK

lpwrit: LCALL   READSPI         ; read SPI ...
        CJNE    A,#0h00,endwrit ; ... while answer is 0x00
        SJMP    lpwrit
endwrit:setb    ss
        MOV     A,#0hff         ; CRC (0xff)
        LCALL   SENDSPI
        RET
```

TAB. 3 – Première partie du programme en assembleur 8051 pour ADuC814 chargé de recevoir les trames GPS transmises sur le port RS232 (4800 bauds) et de les stocker dans une MultiMediaCard, ou de restituer vers un PC par RS232 (9600 bauds) les informations accumulées sur la MMC. Notez que ce circuit et le programme associé servent d'enregistreur de données universel et ne se restreignent pas à l'enregistrement d'informations issues d'un récepteur GPS : nous avons volontairement laissé en commentaire les fonctions d'enregistrement des données issues de convertisseurs analogique-numérique pour stockage sur la MMC dans la fonction `MMCwriteBlock`. Les fonctions appelées dans ce programme sont incluses dans le tableau 1, tandis que dans un souci d'allègement de la présentation l'initialisation de la MMC, identique à celle présentée dans le tableau 2, a été remplacée par les symboles "[...]" après la 45ème ligne de texte dans ce code.

2.2 Les trames issues des récepteurs GPS

Notre première tâche après transfert des trames brutes au format NMEA sur le PC est d'extraire les informations pertinentes et les convertir en un format utilisable.

```
xfer:
        MOV     RCAP2L,#-7
        MOV     TL2,#-7
mov     A,#0haa
lcall   SENDCHAR
mov     A,#0haa
lcall   SENDCHAR
mov     A,MEM1
lcall   SENDCHAR
mov     A,MEM2
lcall   SENDCHAR
mov     A,MEM3
lcall   SENDCHAR

        mov     dmpMEM0,#0        ; RAM location 0, 1, 2 and 3 store the address
        mov     dmpMEM1,#0        ; 64 or 0
        mov     dmpMEM2,#0
        mov     dmpMEM3,#0

        mov     A,MEM1
        jnz     plus2
        mov     A,MEM2
        jnz     plus2
        mov     A,MEM3
        jnz     plus2
        mov     MEM3,#0h01      ; si on est arrive ici, MEM1=MEM2=MEM3=0 ie
                                ; on a fait un reset trop vite => dump 16 MB

plus2:  inc     MEM1            ; necessairement paire donc +1 != 0
        inc     MEM1            ; on lira juskau bloc suivant
        mov     A,MEM1
        JNZ     incmem2
        inc     MEM2
        mov     A,MEM2
        JNZ     incmem2
        inc     MEM3
incmem2:

litout: LCALL   MMCreadBlock    ; read block at address 0x00

        inc     dmpMEM1
        inc     dmpMEM1
        lcall   compare
        mov     A,dmpMEM1       ; set bit Z
        JNZ     litout
        inc     dmpMEM2
        lcall   compare
        mov     A,dmpMEM2       ; set bit Z
        JNZ     litout
        inc     dmpMEM3
        lcall   compare
        sjmp    litout          ; au cas ou' ...

compare:mov     A,MEM3
        CJNE    A,dmpMEM3,pasegal
        mov     A,MEM2
        CJNE    A,dmpMEM2,pasegal
        mov     A,MEM1
        CJNE    A,dmpMEM1,pasegal
        lcall   rmEE                    ; MEM2{2,3}=dmpMEM{2,3} : fini
        mov     A,#0h45         ; E
        LCALL   SENDCHAR        ;
        mov     A,#0h4E         ; N
        LCALL   SENDCHAR
        mov     A,#0h44         ; D
        LCALL   SENDCHAR
theend2: SJMP    theend2;       fin de dumpall
pasegal:ret
;-------------------------------------------------------------------
MMCreadBlock:
        clr     ss              ;
        MOV     A,#0h51         ; command 0d17=0x11 & 0x40
        LCALL   SENDSPI
        MOV     A,dmpMEM3          ; hi byte first ...
        LCALL   SENDSPI
        MOV     A,dmpMEM2
        LCALL   SENDSPI
        MOV     A,dmpMEM1
        LCALL   SENDSPI
        MOV     A,dmpMEM0          ; ... and lo byte last
        LCALL   SENDSPI
        MOV     A,#0hff         ; CRC
        LCALL   SENDSPI

        LCALL   READSPI         ; should be 0x00
        LCALL   READSPI         ; should be 0xfe

        mov     R1,#0h02        ; here we read 512=2x256 values
read1:  mov     R0,#0h00        ;
read0:  LCALL   READSPIVAL      ; read value, even if it is 0xFF => we keep
        LCALL   SENDCHAR        ; all values and cannot use READSPI here
        DJNZ    R0,read0
        DJNZ    R1,read1

        LCALL   READSPIVAL              ; read CRC              POURQUOI PAS ?
        LCALL   READSPIVAL              ; read CRC              POURQUOI PAS ?
        setb    ss

        RET
```

TAB. 4 – Seconde partie du programme en assembleur 8051 pour ADuC814 (à concaténer au code présenté au tableau 3) chargé de restituer vers un PC par RS232 (9600 bauds) les informations accumulées sur la MMC. Les fonctions appelées dans ce programme sont incluses dans le tableau 1.

Seules les lignes commençant par les caractères `$GPGGA` vont nous intéresser ici, les autres lignes contenant principalement de la télémétrie sur les satellites visibles le long du parcours. Cette ligne GPGGA contient notamment la latitude, la longitude et l'altitude du récepteur GPS : nous allons extraire ces informations au moyen de `grep` puis `cut`. Nous constatons cependant que de nombreux points sont aberrant : nous attribuons ces erreurs au stockage des informations sur MultiMediaCard. En effet, cette dernière place en mémoire tampon 512 octets avant de physiquement les écrire en mémoire non-volatile. Cette écriture prend un certain temps, pendant lequel nous plaçons le microcontrôleur en attente d'un acquittement de la carte confirmant la bonne écriture des données. Or l'UART de l'ADuC814 ne peut conserver que le dernier octet reçu sur le port série : tout nouvel octet écrase le précédent. Nous attribuons donc les lignes erronées aux caractères perdus lors du stockage des trames en mémoire non volatile de la MMC. Ces points, relativement rares mais nettement visibles au tracé des parcours, sont éliminés par un filtrage suivant un *a priori* fort sur la forme des points tel que mentionné auparavant : 4 chiffres et 4 décimales pour la latitude, 5 chiffres et 4 décimales pour une longitude.

Nous voulons donc ne conserver que les informations de latitude, longitude et altitude (champs 3, 5 et 10 respectivement de la trame GPGGA), retirer les virgules et les lignes contenant encore des lettres (par exemple à cause d'une erreur d'enregistrement des trames qui accole une lettre à ce qui devrait être un nombre), ne conserver que les nombres contenant 4 ou 5 chiffres avant la virgule et 4 décimales, et finalement retirer les trames qui contiennent encore la valeur ASCII 0xff qui peut se trouver dans les trames du fait de notre implémentation du protocole de sauvegarde sur MultiMediaCard. Ces opérations se résument dans la ligne suivante :

```
grep -a GGA ${nom}.nmea | cut -d, -f3,5,10 | sed 's/,/ /g' | grep -v "[A-Z]" |
```

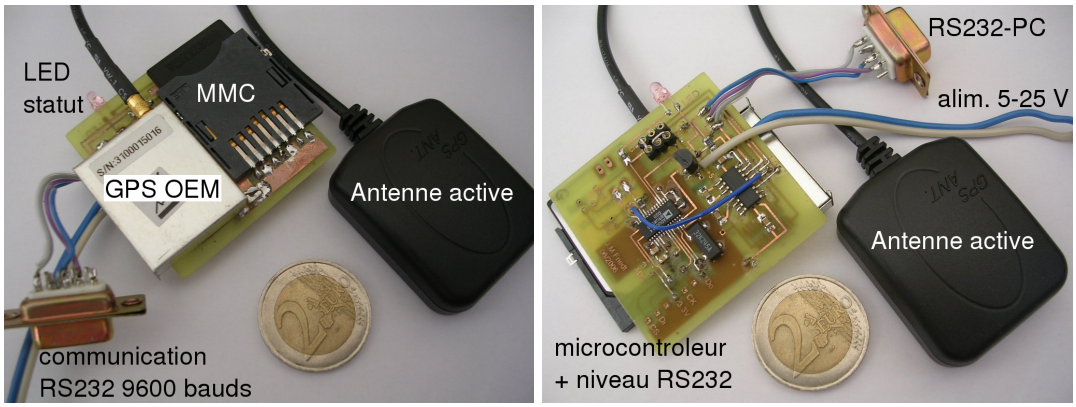

FIG. 2 – Circuit utilisé au cours de ces expériences. Le fil bleu additionel sert à la détection de la présence ou absence d'une connexion RS232 avec un PC, déterminant si le microcontrôleur lance le programme d'acquisition de données depuis le récepteur GPS (absence de connexion) ou le programme de restitution des trames stockées sur la MMC (présence du câble). La diode électroluminescente (LED) fournissant le statut de fonctionnement du récepteur est apparue à l'usage comme une option fondamentale pour déterminer si le circuit fonctionne correctement : la diode s'allume à chaque écriture sur le port RS232, notamment après chaque stockage d'un bloc de 512 octets lors de l'acquisition des trames.

```
grep "^[0-9][0-9][0-9][0-9]\.[0-9][0-9][0-9][0-9]\ [0-9][0-9][0-9][0-9][0-9]\.[0-9][0-9][0-9][0-9]\ ...." |
grep -v $'\xff' | grep "^4"
```

Nous avons ajouté une hypothèse concernant la localisation des traces : nous supposons que la latitude commence par un 4. Ceci est dû au fait que divers récepteurs GPS fournissent des informations variables au cours de l'initialisation : par exemple les récepteurs Sony utilisés donnent 35 degrés, 37.8333 minutes Nord et 139 degrés, 44.6667 minutes Est comme valeurs par défaut, position que nous n'avons pu identifier, se trouvant à mi-chemin entre les sièges de Sony et de NEC [5] à Tôkyô. Cette hypothèse sur la localisation des traces devra être adaptée pour l'utilisation des informations de cet article à des pays autres que la France.

Une fois les trames appropriées sélectionnées, il nous faut les convertir en une information utilisable pour le tracé : au lieu de degrés suivis de minutes (entre 0 et 60) et de fractions de minutes tels que fournis par NMEA, il nous faudrait des degrés suivis de fractions de degrés (entre 0 et 1) afin de déjà tracer les trames sous `gnuplot` ou `Matlab/Octave`. Nous avons pour cela écrit un petit programme de conversion fonctionnant sous `Matlab/Octave` nommé `deg2dec.m` :

```
function sortie=deg2dec(entree)
sortieX=floor(entree(:,1)/100);
sortieX=sortieX+((entree(:,1)/100)-sortieX)/60*100;
sortieY=floor(entree(:,2)/100);
sortieY=sortieY+((entree(:,2)/100)-sortieY)/60*100;
sortie=[sortieY sortieX entree(:,3)];
```

Nous conservons systématiquement la sortie en format ASCII (fonction `save -ascii` de `octave` de cette fonction comme base de travail ultérieur. Par exemple ce programme tranforme

```
4803.0632 150.5188 173.7
4803.0826 150.5236 173.5
4803.102 150.5285 173.4
4803.1214 150.5336 173.2
4803.1408 150.5388 173.1
```

en

[5] `http://www.jref.com/gallery/showphoto.php/photo/1312`

```
1.84198 48.0510533333333 173.7
1.84206 48.0513766666667 173.5
1.84214166666667 48.0517 173.4
1.84222666666667 48.0520233333333 173.2
1.84231333333333 48.0523466666667 173.1
```

après inversion des latitudes et longitudes tel que requis par exemple par le format KML de Google Earth que nous présenterons plus loin.

3 Les bases de données UPCT et Openstreetmap

Un des formats acceptables par la base de données du projet UPCT [6] est celui utilisé par `gpsman`. Bien que relativement bien documenté, le format de fichier doit être *scrupuleusement* respecté pour être acceptable par le serveur web qui se charge de rapatrier les points des traces GPS. Nous avons, avec l'aide de Michel Bondaz [3] qui nous a gracieusement fourni un fichier correctement formaté, pu écrire un convertisseur de données brutes binaires issues d'un récepteur Motorola Oncore vers un format ASCII compris par le serveur de UPCT (Fig. 3). La principale subtilité tient au respect scrupuleux des tabulations qui ne peuvent en aucun cas être remplacées par des espaces (dans la plus pure tradition du Makefile ou des formats de données pour programmes Fortran). Nous obtenons ainsi un formatage qui se décrit comme suit :

- une série de commentaires commençant par un % de la forme
  ```
  % Written by GPSManager 18-Mar-2006 15:06:57 (CET)
  % Edit at your own risk!
  ```
- une définition du format et de la norme de projection des données :
  ```
  !Format: DMS 1 WGS 84
  !Creation: n
  ```
- finalement la création d'une trace sous la forme
  ```
  !T: ACTIVE LOG      width=2 colour=#8b0000  GD310:display=~|Z
      01-Jan-2006 00:00:00    N47 14 53.77    E5 59 19.87 0
      01-Jan-2006 00:00:00    N47 14 53.79    E5 59 19.86 0
  !TS:
      01-Jan-2006 00:00:00    N47 14 53.78    E5 59 19.83 0
      01-Jan-2006 00:00:00    N47 14 53.78    E5 59 19.81 0
  ...
  ```
  où "..." marque bien sur la suite de points à inclure dans la trace. Le point fondamental ici est la présence d'une tabulation entre `!T :` et le nom de la trace (ici ACTIVE LOG). La date et l'heure ne sont incluses que pour respecter le format mais n'ont aucune importance dans la validité des données qui sont elles fournies sous la forme de degré, minute, seconde et fraction de seconde d'angle.

Nous avons décidé d'automatiser la génération de ces fichiers destinés à UPCT plutôt que d'utiliser la fonction `Import` de `gpsman` car d'une part nous n'arrivions pas à la faire fonctionner, mais surtout nous pouvons inclure ainsi nos propres filtres destinés à épurer les acquisitions de valeurs aberrantes. Nous faisons appel à une séquence de 3 opérations pour transformer nos fichiers au format NMEA en un fichier UPCT :

- soit un fichier à traiter dont le nom est supposé avoir pour extension `.nmea` tel que obtenu dans un script bash par
  ```
  #!/bin/bash nom=‘basename $1 .nmea‘
  ```
- de ce fichier nous allons extraire uniquement les trames contenant le terme GPGGA tel que expliqué auparavant (section 2.2).
- Nous avons constaté pour le moment que cette ligne épure la trace de tout point aberrant. Il nous reste maintenant à convertir ces données de la forme degrés-minute-fraction de minutes en un format intelligible par UPCT. Pour cela nous faisons appel à `octave` qui exécute le script suivant en incluant la commande `octave upct.m` où `upct.m` contient :
  ```
  % ici prend en entree la sortie de go.csh sur un fichier nmea
  ```

[6] `http://www.upct.org`

```
%function upct(nom)
%eval(['load ',nom,'.upct']);
%eval(['x=',nom,';']);
%eval(['clear ',nom]);
load x
mx=deg2dec(x);
a=dec2deg(mx(:,1));
b=dec2deg(mx(:,2));
d=ones(1,length(a))*99999;
c=[b d' a];
save -ascii nom.degres c
```

où les fonctions `dec2deg.m` et `deg2dec.m` ont respectivement pour objectif de convertir des décimales de minutes en secondes (telles que présentées auparavant, section 2.2), puis de séparer les contributions des degrés, minutes et secondes en trois colonnes distinctes. Cette seconde fonction se présente sous la forme

```
function sortie=dec2deg(entree)
degre=floor(entree);
reste=(entree-degre)*60;
minute=floor(reste);
seconde=(floor((reste-minute)*60*100))/100;
sortie=[degre minute seconde];
```

L'intérêt de séparer ainsi ces deux fonctions apparaîtra plus tard avec la génération de fichiers KML pour Google Earth : seule la première étape sera alors nécessaire, la seconde étant spécifique à UPCT.

– Finalement nous convertissons le format de sauvegarde ASCII de octave en un format utilisable par UPCT en éliminant les commentaires de la sauvegarde, en ajoutant une altitude nulle pour tous les points et une date permettant de respecter la convention du format de fichier :

```
grep -v ^\# nom.degres | sed 's/^ / 01-Jan-2006 00:00:00    N/g' | sed 's/$/    0/g' | sed 's/99999 /   E/g' > x
```

Ici encore il faut prendre soin de placer des tabulations avant la date, entre l'heure et le N et avant les 0 et E du second et troisième `sed` respectivement.

– Finalement nous ajoutons en début de fichier l'entête constant de UPCT présentée plus tôt dans ce paragraphe (`cat entete.upct x > $nom.upct`), et incluons 2 lignes après le début de la trace la commande `!TS :` par un appel à vim (`vi -n -s upct.vi $nom.upct`) où `upct.vi` contient les commandes telles que nous les tapperions sous cet éditeur :

```
:8
dd
p
ZZ
```

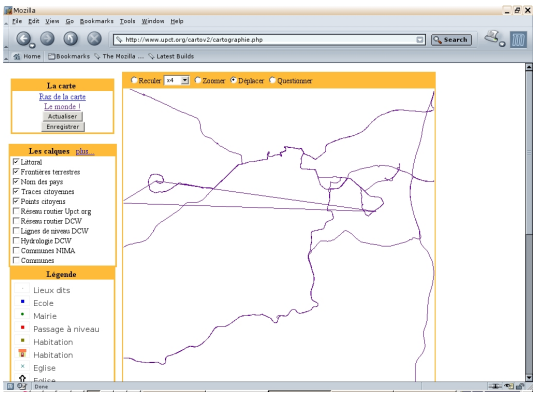

FIG. 3 – Exemple de traces autour de Clermont-Ferrand – la spirale sur la gauche représente la route montant le Puy de Dôme – extraites de la base de données de `upct.org`.

L'épuration des traces de tout point aberrant et la validation des traces sur des outils tels que Google Maps ou Google Earth nous semblent fondamentales avant de partager ces traces avec la communauté d'utilisateurs de UPCT : en effet, la représentation des traces se fait en reliant

les points entre eux. Ainsi, le moindre octet erroné lors du stockage des informations issues du récepteur se traduit par une immense trace peu esthétique et surtout sans réalité physique (Fig. 4).

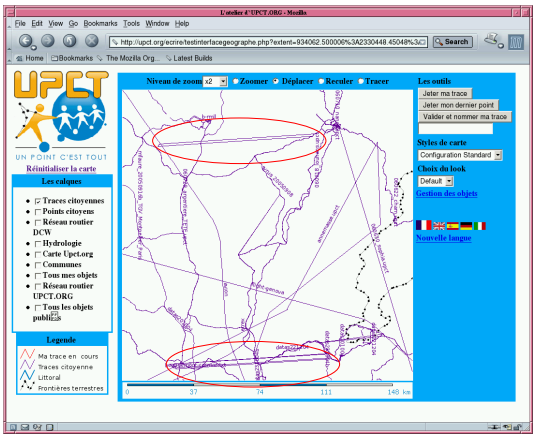

FIG. 4 – Exemple de trace erronée obtenue lors d'un trajet Besançon-Annemasse au moyen d'un récepteur Motorola Oncore. En deux endroits – indiqués ici par des ellipses rouges – la trace a été translatée pour des raisons que nous ne pouvons expliquer. La continuité de la forme de la trace semble indiquer qu'il s'agit soit d'une erreur lors du stockage des informations, soit lors de la conversion depuis le format binaire.

Ces points erronés n'influent pas sur la validité du travail de cartographie puisque nous allons voir plus bas que les traces brutes sont ensuite manuellement vectorisées pour une identification des voies de communication, mais ils diminuent la clareté de la carte UPCT dans son ensemble et rendent le travail de vectorisation plus difficile.

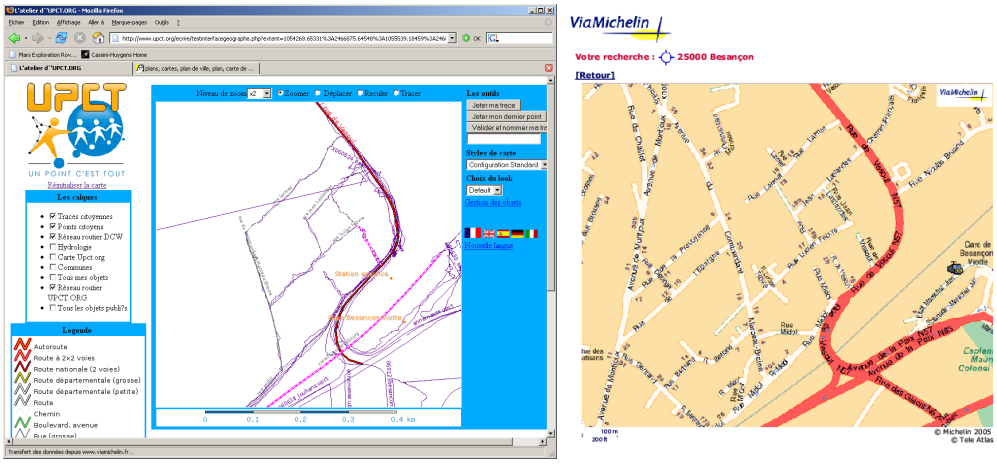

FIG. 5 – Exemple de vectorisation d'une région autour de Besançon et comparaison avec la base de donnée disponible auprès de `www.viamichelin.fr`. À titre d'illustration de l'indépendance au système d'exploitation de l'interface de vectorisation, ces captures d'écrans ont été obtenues sous MS-Windows.

Nous constatons donc que UPCT se comporte dans un premier temps comme un dépôt de traces GPS avec éventuellement la reconnaissance de quelques sites remarquables sur un trajet (points). Une fois suffisamment de traces accumulées sur un trajet donné afin d'atteindre une statistique

suffisante, la seconde étape tient dans la vectorisation et l'identification des traces afin de réaliser une cartographie au sens habituel du terme. Cette seconde phase, relativement fastidieuse, est la plus satisfaisante puisqu'elle conclut le travail d'acquisition des traces. La suite des étapes pour cette seconde tâche se fait par :

- connexion à `www.upct.org` et sélectionner "Dessiner sur la carte". Activer "Traces citoyennes" et "Réseau routier UPCT.ORG",
- se déplacer jusqu'à la zone à vectoriser et zoomer jusqu'à atteindre une barre pleine échelle de 120 m (×96, taille de l'unité d'échelle de 30 m),
- activer le bouton "Tracer" et sélectionner successivement des points sur la trace à vectoriser [7]. Le principe de mise en commun des traces implique que seules les traces suivies plusieurs fois sont vectorisées afin de limiter les erreurs de biais sur la position de la route en cours de vectorisation par une statistique suffisante. Une fois le segment de trace vectorisé, introduire le nom identifiant ce segment dans l'emplacement prévu à cet effet sous "Valider et nommer ma trace", puis conclure en cliquant sur ce bouton. La trace vectorisée est alors placée dans la liste d'objets publiables sur la carte.
- Il reste alors à définir les propriétés pour la publication de ces données : cliquer sur "Gestion des objets" pour visualiser la liste des objets en attente de publication.
- Pour chaque objet en attente de publication, sélectionner le Type de la trace *qu'il faut valider après sélection* en cliquant sur "Enregistrer". Le poids est indicatif de l'importance de cet axe de transport pour la population locale.
- La publication s'achève par "Demander la publication", qui met en attente la trace vectorisée d'une validation de l'administrateur du site.

UPCT ne fournit qu'une interface utilisable sur le web pour vectoriser les traces. L'avantage de cette solution est de ne pas nécessite l'installation d'un logiciel sur la machine locale pour ajouter ses traces, d'autant plus que l'interface est rapide et souple d'emploi. Les inconvénients majeurs de cette solutions sont :

- les risques de lenteur associées aux connexions réseau. Alors que l'interface a été pensée en vue d'une connexion bas débit avec un minimum d'information transférées à chaque ajout de point lors de la vectorisation, nous avons observé des problèmes lors de la connexion depuis un site à bande passante élevée (Renater) mais derrière une passerelle particulièrement mal configurée qui nécessite quelques secondes pour acquitter chaque nouvelle connexion (proxy de la zone étudiante de l'université de Franche-Comté à Besançon). Étant donné que l'ajout d'un nouveau point nécessite une nouvelle connexion, la séquence de tracé d'un nouveau chemin est tellement lente qu'il en devient inutilisable.
- L'impossibilité de travailler en l'absence de connexion réseau telle que dans le train par exemple, qui serait pourtant un endroit approprié pour une telle activité.

Au contraire Openstreetmap propose plusieurs logiciels à installer sur sa machine locale pour la vectorisation des traces hors connexion réseau avant d'envoyer le résultat de cette opération vers le serveur (`http://wiki.openstreetmap.org/index.php/Compare_editors`). Les traces sont ici superposées aux images satellites de Landsat publiquement accessibles, de moins bonne résolution en milieu urbain que les images disponibles sur les serveur de Google (Maps et Earth) mais pouvant fournir une aide à la vectorisation si nécessaire. Le format requis par OpenStreetMap est de type XML – GPX – fourni par GPSBabel, que nous n'avons pas exploré de façon approfondi pour le moment. L'ensemble des traces vectorisée peut ensuite être téléchargé pour une utilisation locale auprès de `http://wiki.openstreetmap.org/index.php/Planet.osm`.

L'intérêt de ces projets est visible dans la liste de disponibilité de bases de données de routes dans le monde : alors que librement disponibles aux États-Unis, ces bases de données ne sont disponibles que pour des somme exorbitantes en Europe (à l'exception du Danemark) et en tout cas inaccessible à l'utilisateur individuel [4, p.397].

[7] À titre indicatif, on peut considérer qu'en sélectionnant une fois par seconde des points séparés de 30 m, la vectorisation se fait à la vitesse de l'ordre de 100 km/h : il faudra donc sensiblement la même durée pour parcourir une route et la vectoriser sous UPCT.

4 Affichage des points sur Google Earth

Suite à la publication de notre article précédent [1], un lecteur – Benoît Rolland – nous a informé de la capacité de Google Earth à insérer des points fournis par un utilisateur dans les cartes affichées par le logiciel. Nous nous sommes donc proposés là encore d'écrire un convertisseur de formats de données, binaire pour le récepteur Motorola Oncore, et NMEA pour les autres récepteurs, vers le format *kml* compris par Google Earth. Contrairement au cas précédent, le formatage de ce fichier ASCII est très libre et son format parfaitement décrit à `http://www.keyhole.com/kml/kml_tut.html` qui vient d'être remis à jour à `http://earth.google.com/kml/kml_21tutorial.html`. Nous nous inspirerons fortement de ces exemples pour générer nos propres fichiers :

- une entête que nous garderons constante pour tous les fichiers :
```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Placemark>
  <description>Trajet Besancon-Chamonix, 23-06-2006</description>
  <name>Absolute Extruded</name>
```
  dont nous pourrons éventuellement prendre soin de modifier la description du document,
- une définition de point de vue et de la direction de visée initiale après chargement de la trace :
```
<LookAt>
    <longitude>6.02194166666667</longitude>
    <latitude>47.24624</latitude>
    <range>4451.842204068102</range>
    <tilt>44.61038665812578</tilt>
    <heading>-125.7518698668815</heading>
</LookAt>
```
  généralement nous nous contentons ici de modifier de façon automatique la latitude et longitude de départ pour conserver la direction de visée fixe,
- un certain nombre d'options de tracé du polygone représentant la trace sur les données issues de Google Earth :
```
<visibility>1</visibility>
  <open>0</open>
  <Style>
     <LineStyle> <color>ff00ffff</color> </LineStyle>
     <PolyStyle> <color>7f00ff00</color> </PolyStyle>
  </Style>
  <LineString>
  <extrude>1</extrude> <tessellate>1</tessellate>
  <altitudeMode>absolute</altitudeMode>
```
- finalement le cœur du fichier : une suite de points de la forme {longitude,latitude,altitude} fournis sous forme décimale
```
<coordinates>
6.02194166666667,47.24624,100
6.02194333333333,47.2462416666667,100
6.02194333333333,47.2462433333333,100
...
</coordinates> </LineString> </Placemark> </kml>
```
  où `...` indique bien entendu la suite des points à insérer dans la trace.

L'insertion des traces GPS dans Google Earth est d'autant plus pertinente depuis la mi-juin que Google vient de fournir à la communauté d'utilisateurs de GNU/Linux une version de son logiciel fonctionnant sur ce système d'exploitation `http://earth.google.com/download-earth.html`.

Les résultats sont impressionnants de précision, résultats qu'il faut attribuer à l'excellente qualité des signaux GPS d'une part, et aux concepteurs de Google Earth (originellement KeyHole) pour leur alignement des images satellites sur les coordonnées GPS.

5 Affichage de cartes sur Google Earth

Google Earth propose, en plus d'ajouter des polygones et des points à ses cartes, de superposer en transparence des images au format JPEG. Des applications tout à fait étonnantes de ces options ont été réalisées par les utilisateurs de Google Earth, notamment pour la visualisation géographique

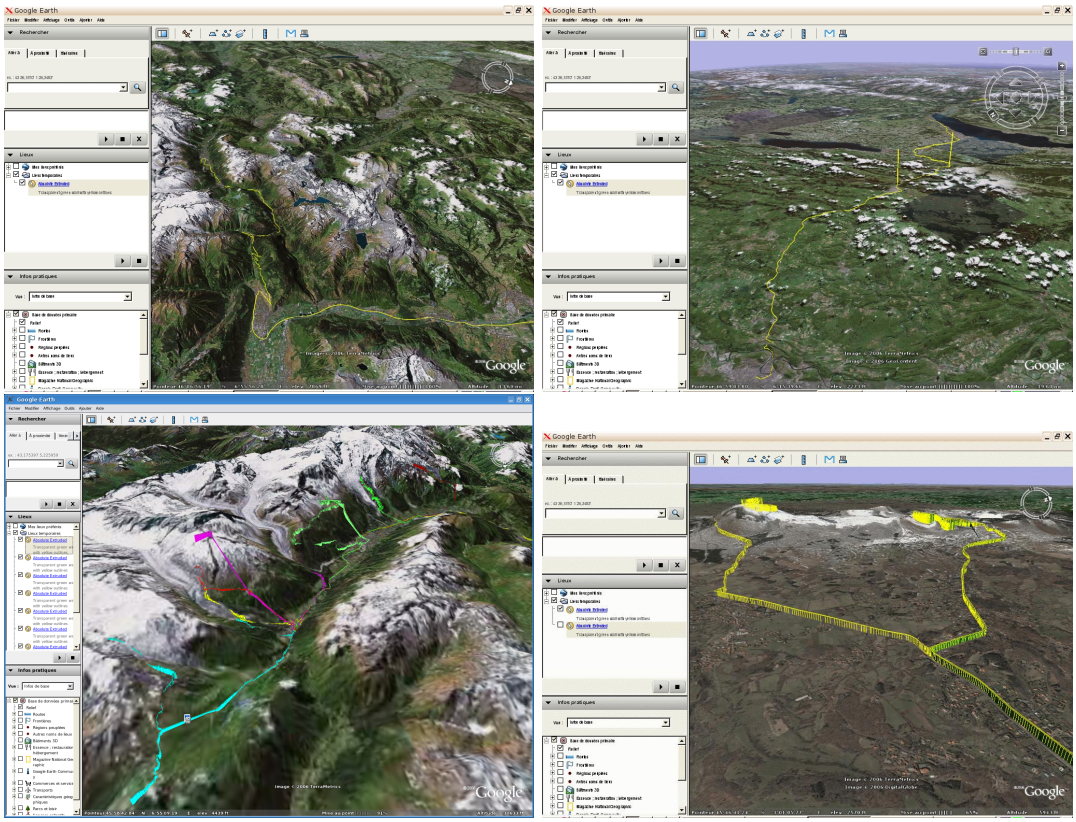

FIG. 6 – Quelques exemples d'images issues de Google Earth présentant l'intérêt en présence de reliefs remarquables de visualiser la troisième dimension. Sur toutes les figures, la trace jaune correspond à la trace acquise par notre montage et incluse par conversion au format KML. De gauche à droite et de haut en bas : arrivée dans la vallée de Chamonix par le côté suisse ; le trajet depuis Besançon vers le lac Léman pour ensuite atteindre Chamonix, mettant en évidence deux points d'altitudes aberrantes ; randonnées dans la vallée de Chamonix (chaque trace est ici encodée par une couleur différente) ; trajet sur les puys près de Clermont Ferrand.

de données accumulées et disponibles par internet (voir dans Google Earth le forum Dynamic Data Layers [8].

Superposer des images de cartes à Google Earth nécessite d'utiliser une projection appropriée, dite Projection Cylindrique (ou Plate-Carrée [5]). Nous avons utilisé une toolbox gratuitement disponible pour Matlab, `m_map` [9] pour inclure nos traces GPS dans une carte d'une partie de l'Europe de l'Ouest et projeter l'image résultante sur Google Earth (Fig. 7). L'intérêt est ici d'avoir accès à toute la puissance en terme de calculs matriciels et en traitement du signal de Matlab pour extraire les informations pertinentes de grandes quantités de points, par exemple pour un réseau de capteurs géographiquement distribués. L'exemple proposé ici se borne à valider la capacité de plaquer une carte convenablement projetée, incluant nos propres points, sur le globe de Google Earth.

Nous utilisons les commandes suivantes sous Matlab pour tracer la carte :

```
path(path,'c:\matlab\m_map');
m_proj('Miller Cylindrical','longitudes',[-10 10],'latitudes',[37 57]);
m_coast; axis square;              % trace du fond de carte
```

[8] `http://bbs.keyhole.com/ubb/postlist.php/Cat/0/Board/EarthExternalData`

[9] `http://www.eos.ubc.ca/~rich/map.html` a été testé sous Matlab mais ne semble pas fonctionner en l'état sous Octave

```
load 060622_cham.dat;              % fichier long,lat,alti issu du grep
s=deg2dec(X060622_cham);           % conversion en format decimal
u=diff(s(:,1)); v=diff(s(:,2));    % vecteur vitesse
lat=s(1:length(s)-1,2); lon=s(1:length(s)-1,1);
hold on;m_quiver(lon,lat,u,v,'r'); % iterer pour chaque nouvelle trace
```

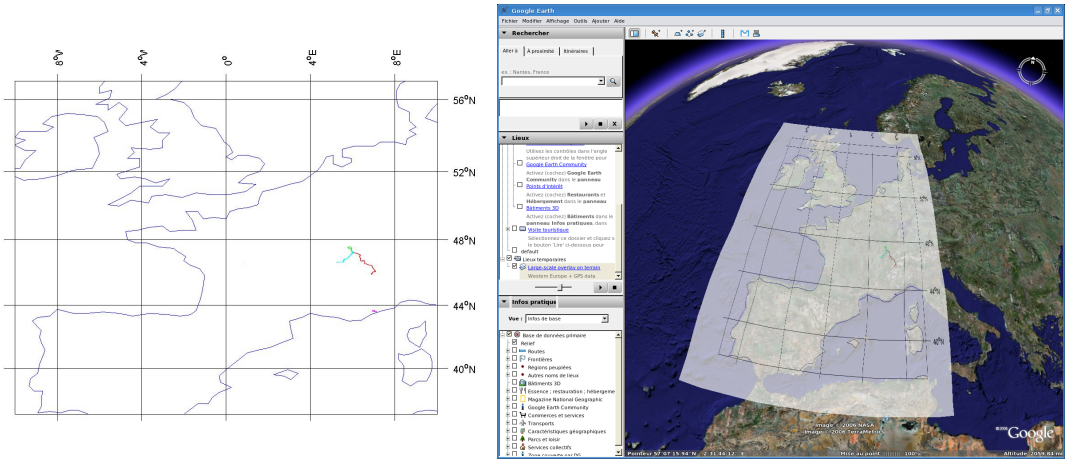

FIG. 7 – Gauche : image issue de la toolbox M_MAP de Matlab en projection cylindrique de l'une partie de l'Europe de l'Ouest sur laquelle a été ajoutée quelque traces GPS. Droite : projection des cette image sur le globe de Google Earth.

Nous utilisons l'attribut `color` pour afficher l'image en transparence sur le globe de Google Earth au moyen de la balise `Icon` de la façon suivante :

```
<color> a9ffffff </color>
  <Icon>
    <href>http://mmyotte.free.fr/france_mmpap2.jpg</href>
  </Icon>
  <LatLonBox id="khLatLonBox751">
    <north>59.57</north>
    <south>34.333</south>
    <east>12.70</east>
    <west>-13.7666</west>
    <rotation>0</rotation>
  </LatLonBox>
```

6 Affichage des points sur Google Maps

Contrairement à Google Earth qui nécessite l'installation d'un logiciel propriétaire sur sa machine locale, Google Maps fournit une interface de programmation (API) accessible en javascript : la plupart des navigateurs web sont ainsi capables de tracer une carte issue du serveur Google Maps et d'afficher la vue aérienne ainsi que la carte routière associée sur l'ordinateur local. L'inconvénient majeur de Google Maps par rapport à Google Earth est de ne pas fournir la 3ème dimension qu'est l'altitude des points, information dont nous pourrons nous passer pour de nombreuses applications.

Nous partirons de l'hypothèse que nos points sont stockés au format `kml` tels que reconnus par Google Earth. La discussion qui suit sera ensuite étendue à des séries de points stockés en format NMEA. Le premier problème à résoudre est l'accès au serveur Google Maps et le rapatriement d'une carte aux coordonnées sélectionnées, munies d'une interface utilisateur minimale (changement du type d'affichage entre route et photographies aériennes, translation et homothétie de la carte). Le second problème est l'ajout de nos propres points à ces cartes.

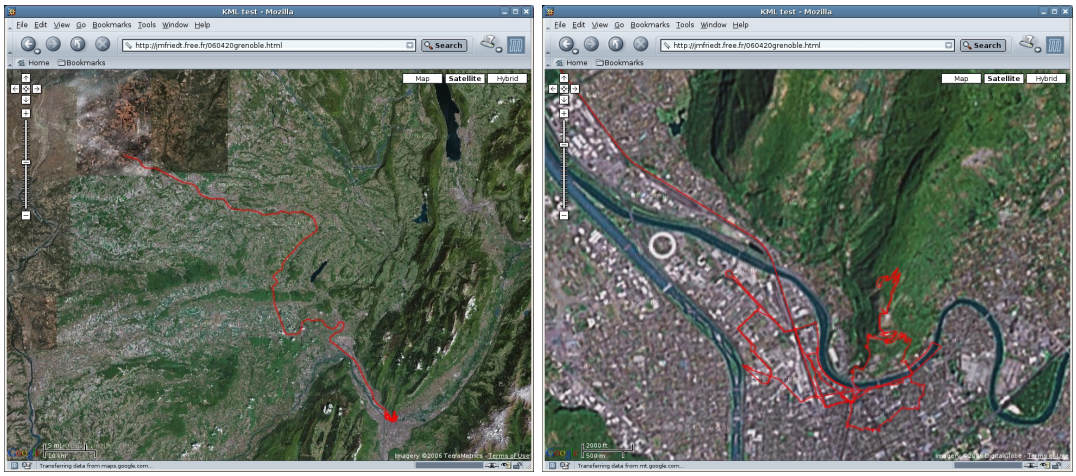

FIG. 8 – Page web générée manuellement depuis un fichier contenant une série de couples {latitude,longitude} afin d'afficher ces points sur la base de données de vues aériennes Google Maps. À gauche le trajet en train de Lyon à Grenoble, à droite un zoom sur Grenoble et les trajets à pieds qui y ont été effectués. L'information d'élévation n'est *pas* disponible dans Google Maps.

6.1 L'interface javascript de Google Maps

Afin de nous familiariser avec l'API de Google Maps nous allons générer manuellement une page web faisant appel au serveur et ajoutant sur les cartes ainsi rapatriées nos points tracés sous forme de polygones (Fig. 8).

Google requiert, afin de permettre l'affichage de ses cartes sur une page web accessible au public, de s'enregistrer afin d'obtenir une clé associée à un domaine spécifique (dans l'exemple qui va suivre, la clé obtenue est valide pour l'ensemble du domaine `jmfriedt.free.fr` – notez que pour une utilisation locale telle qu'une page web stockée sur le disque dur de la machine exécutant l'interpréteur HTML, n'importe quelle clé convient) : la page pour ce faire est disponible à `http://www.google.com/apis/maps/signup.html`. À l'issue de cet enregistrement, nous obtenons un exemple de code qui va servir de base aux développements ultérieurs. Par exemple pour générer une clé pour l'ensemble du site `friedtj.free.fr`, nous obtenons comme réponse :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=ABQIAAAAQ5iKlBqfwApuWu7aIzUiPRTELJePVJ4bAZX18VT1uB
      type="text/javascript"></script>
    <script type="text/javascript">

    //<![CDATA[

    function load() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
      }
    }

    //]]>
    </script>
  </head>
  <body onload="load()" onunload="GUnload()">
    <div id="map" style="width: 500px; height: 300px"></div>
  </body>
```

```
</html>
```

Nous constatons dans cet exemple que nous pouvons
- nous connecter au serveur Google Maps et en rapatrier une carte centrée sur des coordonnées fournies sous forme de latitude/longitude,
- sélectionner un mode d'affichage et un taux de grossissement parmi un ensemble de valeurs prédéfinies,
- ajouter à la page un certain nombre de commandes telles que la translation et l'homothétie de la carte ou le changement du type d'affichage (carte routière ou vue aérienne).

L'API Google Maps est décrite de façon très générale à `http://www.google.com/apis/maps/documentation/` mais les fonctions pour l'ajout de nos propres points sont issues de lectures de codes source sur le web et notamment `http://www.obviously.com/gis/gpx_loader.html`.

Avant d'automatiser la génération des pages webs au moyen de scripts PHP, nous allons nous familiariser avec la syntaxe en générant une page manuellement. Partant de l'entête précédent, nous désirons ajouter à la carte fournie par le serveur Google Maps un polygone incluant nos propres traces GPS. Pour ce faire, nous déclarons un tableau de points : `var points = new Array() ;` que nous allons remplir petit à petit des coordonnées des points acquis par `points.push(new GPoint(5.04382,45.6604383333333)) ;` pour par exemple ajouter le point de coordonnées {5,044 ;45.660}. L'idée la plus simple consistant à considérer d'accumuler tous les points d'une trace en un seul polygone ne fonctionne pas car l'affichage de cartes devient excessivement lent au point de s'interrompre lorsqu'un polygone contient plus d'une centaine de points, le maximum admissible avant d'interrompre l'affichage étant de l'ordre du millier de segments.

Il nous faut donc découper notre trace en plusieur polygones que nous avons choisi de taille 60 points, ce qui revient à 1 minute de parcours par polygone : cette opération a été réalisée manuellement dans un premier temps pour cet exemple dont le but est de valider le concept, et nous le verrons inclus dans les scripts PHP de génération automatique de pages web plus bas. La création du polyone se conclue par son affichage : `map.addOverlay(new GPolyline(points, "#FF0000",2,.75)) ;`, puis la réinitialisation de la variable en un tableau vide avant de réitérer le processus jusqu'à la fin de la trace : `points=[] ; points.push(new GPoint(5.125925,45.63738)) ;` ....

Une page se génère facilement à partir d'instructions `sed` mais nous voyons déjà apparaître plusieurs problèmes qui vont être résolus plus tard :
- les pages ainsi générées sont énormes. Un parcours d'un peu plus de 5 heures à raison d'un point par seconde génère une page de 1,1 MB (qui s'affiche malgré tout raisonnablement vite dans Google Maps). Pour rapidement citer les causes de cette tailles excessive : l'absence de fonctions qui réduiraient la taille de la page et l'excès de décimales dans les coordonnées,
- nous désirons pouvoir générer les pages au vol *via* une interface web sans imposer à l'utilisateur l'installation d'outils avec lesquels il n'est pas nécessairement familier.

En conclusion, il nous faut retenir les commandes importantes qui sont :
- la définition de l'objet contenant la carte : `var map = new GMap(document.getElementById("map")) ;`
- la définition de la visualisation des images aériennes au lieu du mode par défaut qui est le tracé de routes : `map.setMapType(G_SATELLITE_TYPE) ;`.
- l'ajout des commandes permettant à l'utilisateur d'interagir avec les cartes : `map.addControl(new GLargeMapControl()) ; map.addControl(new GMapTypeControl()) ; map.addControl(new GScaleControl()) ;` ,
- `map.centerAndZoom(new GPoint(6.0, 47.0), 5) ;` qui définit la position du centre de l'objet `map`, ici le point de longitude et latitude {6,0 ;47.0} avec un grossissement de 5. En pratique nous remplacerons ces valeurs arbitraires par la première coordonnée pertinente de la trace (point de départ de la trace).

6.2 L'ajout automatique des traces au format KML à Google Maps

Nous avons séquentiellement abordé deux approches à l'ajout de traces aux cartes : la génération de pages HTML contenant les points et les commandes associées à leur affichage sous forme de po-

lygone selon l'API Google Maps (voir section précédente), et l'utilisation d'un script PHP chargé de lire un fichier disponible sur le web, contenant des données au format KML ou NMEA, et de générer la page HTML au vol. La seconde solution nous semble plus souple mais nécessite un serveur supportant le PHP.

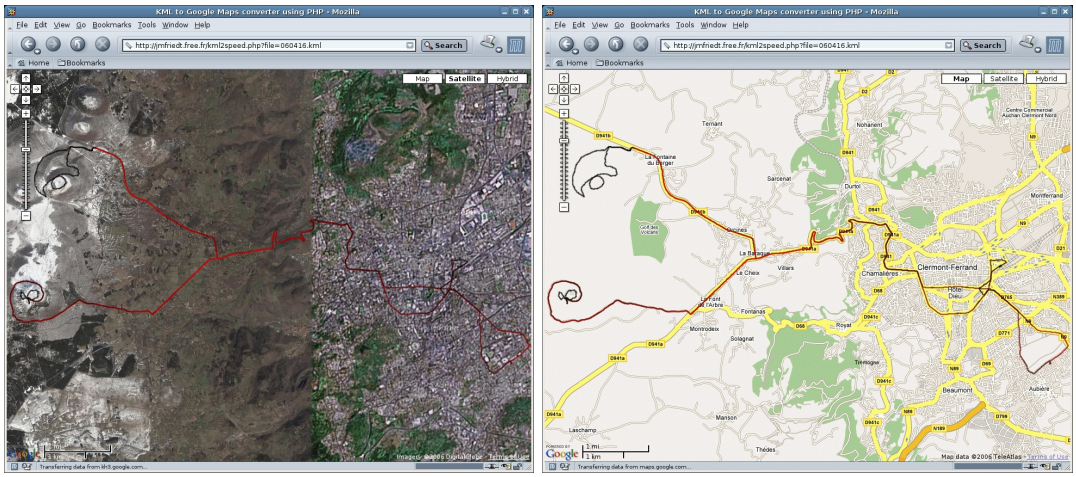

FIG. 9 – Tracé d'un parcours réalisé en voiture puis à pieds dans les puys près de Clermont-Ferrand : nous représentons ici par une évolution de couleurs la vitesse de dépacement du récepteur GPS, du plus foncé (le plus lent) au plus rapide (rouge). Google Maps permet soit de représenter la vue aérienne du parcours (gauche), soit la route correspondante (droite). La moyenne de vitesse a été effectuée sur 60 points (1 minute), taille de chaque polygone tracé pour lequel une même couleur est définie.

Ayant présenté auparavant la génération de fichiers au format KML pour Google Maps qui se résume principalement en l'extraction depuis le fichier brut d'enregistrement des trames GPS (binaire pour Motorola Oncore, NMEA sinon), ils nous a semblé simple de nous familiariser avec un script PHP se contentant de prendre ces série de coordonnées et de les inclure dans l'API Google Maps pour affichage dans une interface web. Nous verrons plus tard comment améliorer ce script PHP pour lire directement des fichiers au format NMEA.

Le PHP consiste en un langage interprété du côté du serveur afin de générer dynamiquement une page : nous conservons les entêtes imposées par l'API Google Maps telles que décrites auparavant, mais au lieu d'inclure des points au moyen de `sed` puis d'exporter la page résultante sur un serveur web, nous allons cette fois passer comme argument au script un nom de page contenant un fichier au format KML qui sera utilisé pour générer la page web. La partie dynamique de la page est incluse entre les balises `<?php ... ?>`. Le passage de paramètre se fait dans l'URL appelant le script : par exemple `http ://jmfriedt.free.fr/kml2maps.php?file=monfichier.kml` appelle le script PHP `kml2maps.php` disponible sur le serveur `jmfriedt.free.fr` en passant comme argument une variable `file` contenant le nom du fichier à traiter (supposé ici présent sur le même serveur, mais qui pourrait aussi être une URL vers un fichier disponible sur un autre serveur web). Cet argument est récupéré côté PHP dans la variable nommée `$file`. Notre script PHP doit effectuer deux tâches :

- dans un premier temps, identifier la première coordonnée pertinente du fichier KML afin de définir sur quel point centrer la carte requise auprès du serveur Google Maps (argument de la méthode `centerAndZoom()`), notamment en éliminant l'entête du fichier KML : `do $buffer=fgets($handle, 4096) ; while (strpos($buffer, "coordinates")===false) ;` qui signifie que l'on élimine l'entête du fichier (adressé par la variable `$handle`) jusqu'à atteindre la balise `coordinates`.
- Une fois la zone des coordonnées atteinte dans le fichier KML, il nous faut les récupérer et les transformer en un format acceptable pour la page web appelant le serveur Google Maps, et ce

jusqu'à atteindre une nouvelle fois la balise `coordinates` indiquant la fin de la trace : `while (strpos($buffer,"coordinates")===false) { $buffer = fgets($handle, 4096) ; list($lon, $lat, $alt)=explode(",",$buffer) ; }` et en plus périodiquement tracé le polygone et réinitialisé le tableau de points afin de ne pas dépasser la taille limite au-delà de laquelle l'affichage perd de sa fluidité. Ainsi cette fonction se résume en

```
while (strpos($buffer,"coordinates")===false) {
  $n++;
  if ( $n == 100) {
     $n=0;
     printf("map.addOverlay(new GPolyline(points, \"#FF0000\", 2, .75));
     points=[];");
    }
  list($lon, $lat, $alt)=explode(",",$buffer);
  print "points.push(new GPoint($lon,$lat));";
  $buffer = fgets($handle, 4096);
}
```

Le script PHP dans son ensemble est proposé dans le tableau 5.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:v="urn:schemas-microsoft-com:vml">
<head>
<title>KML to Google Maps converter using PHP </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="description" content="test - powered by Google Maps" />
<script src="http://maps.google.com/maps?file=api&amp;v=2&amp;
key=ABQIAAAAQ5iKlBqfwApuWu7aIzUiPRRsafyVTyOwfc9A1qQgm5QZxWMvmxQR2gKn15-3mqugeybg6Yf1Za_yIQ"
type="text/javascript">
</script>

<style type="text/css">
            v\:* {
            behavior:url(#default#VML);
            }
            html, body, #map
            {
              width: 100%;
              height: 100%;
            }
          body            {
            margin-top: 0px;
            margin-right: 0px;
            margin-left: 0px;
            margin-bottom: 0px;
            }

</style>
</head>

<body>
<div id="map"> </div>

<script type="text/javascript">
//<![CDATA[            // check for compatibility

if (GBrowserIsCompatible()) { // call the info window opener

  function makeOpenerCaller(i) {
     return function() {showMarkerInfo(i);};} // open an info window
  function showMarkerInfo(i) {
     markers[i].openInfoWindowHtml(infoHtmls[i]);
  }               // create the map

     var map = new GMap(document.getElementById("map"));
     map.setMapType(G_SATELLITE_TYPE);
     map.addControl(new GLargeMapControl());
     map.addControl(new GMapTypeControl());
     map.addControl(new GScaleControl());
```

```
<?php
if ($file=="")  {$file="060416.kml";}
$handle = fopen($file, "r");   // ou http://jmfriedt.free.fr

 if ($handle) {
       do {$buffer = fgets($handle, 4096);}
          while (strpos($buffer,"coordinates")===false);
       $buffer = fgets($handle, 4096);
       list($lon, $lat, $alt)=explode(",",$buffer);
       print "map.centerAndZoom(new GPoint($lon, $lat), 5)";
       fclose($handle);
 }
?>

 if (window.attachEvent) {
    window.attachEvent("onresize", function() {this.map.onResize()} );
 } else {
    window.addEventListener("resize",function() {this.map.onResize()},false);
 }               // add a polyline overlay
 var points = new Array();

<?php
 if ($file=="")  {$file="060416.kml";}
 $handle = fopen($file, "r");        // au lieu de http://jmfriedt.free.fr ...

 if ($handle) {
      do {$buffer = fgets($handle, 4096);}
         while (strpos($buffer,"coordinates")===false);
      $buffer = fgets($handle, 4096);
      $n = 0;
      while (strpos($buffer,"coordinates")===false) {
       $n++;
       if ( $n == 100) {
         $n=0;
         printf("map.addOverlay(new GPolyline(points, \"#FF0000\", 2, .75));points=[];");
       }
       list($lon, $lat, $alt)=explode(",",$buffer);
       print "points.push(new GPoint($lon,$lat));";
       $buffer = fgets($handle, 4096);
      }
      fclose($handle);
 }
?>
   map.addOverlay(new GPolyline(points, "#FF0000", 2, .75)); }
 else {
   document.getElementById("quicklinks").innerHTML ="web browser not compatible"
 }
//]]>
</script>
</body>
</html>
```

Tab. 5 – Script PHP pour la génération de pages web incluant les points d'un fichier KML pour superposition aux images aériennes issues du serveur Google Maps.

Il apparaît que nous pouvons ajouter des fonctionnalités à ce programme en remplaçant la couleur constante des polygones successifs par une couleur associée à une quantité physique mesurée simultanément à l'acquisition des trames GPS (par exemple notre microcontrîôleur est équipé d'un capteur de température LM35 et est capable de compter les impulsions d'un compteur Geiger pour la détection de rayonnements ionisant). À titre d'illustration nous proposons d'encoder la vitesse du véhicule portant le récepteur GPS, moyennée sur une minute, dans la couleur de chaque polygone tel que présenté sur la figure 9. Cette fonctionnalité a été ajoutée par

```
do {
    list($lon, $lat, $alt)=explode(",",$buffer);
```

```
    $vit+=floor(sqrt(($lono-$lon)*($lono-$lon)+($lato-$lat)*($lato-$lat))*20000);
    print "points.push(new GPoint($lon,$lat));";
    $i++;
    if ($i==120) {
//      if ($vit > 255) $vit=255;
        $couleur=sprintf("%02X0000",$vit);
   // will ignore unnecessary 0 in the color code
        print "map.addOverlay(new GPolyline(points,\"#$couleur\",2,.75));\n";
        print "points=[];";
        print "points.push(new GPoint($lon,$lat));";
        $i=0; $vit=0;
    }
    $lono=$lon;
    $lato=$lat;
    $buffer = fgets($handle, 4096);
   } while (strpos($buffer,"coordinates")===false);
   fclose($handle);
  }
```

qui se résume donc à mémoriser le point précédent et à calculer la norme du vecteur vitesse pour chaque nouveau point, valeur qui après moyennage sur 1 minute est utilisée pour sélectionner la couleur du polygone. Le reste du script présenté auparavant (table 5) reste le même et seule la fonction principale de génération de la page web a été modifiée.

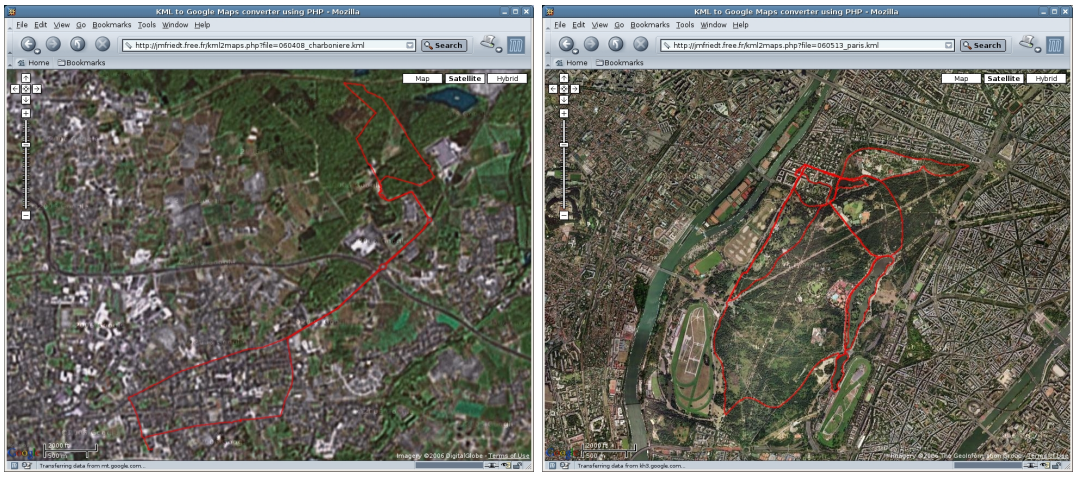

FIG. 10 – Exemple de deux tracés issus de fichiers KML disponibles sur le web pour Google Earth et ici convertis de façon transparente pour l'utilisateur pour un affichage en 2D sur Google Maps (ne nécessitant donc pas une installation locale d'un logiciel dédié tel que Google Earth). À gauche le parc de Charbonnière près d'Orléans, à droite le parc de Longchamps près de Paris. Notez que le grossissement est le même mais que la vue de Paris est disponible en haute résolution.

6.3 L'ajout automatique des traces au format NMEA à Google Maps

Finalement la solution la plus simple pour un utilisateur est de pouvoir simplement mettre ses trames NMEA sur une page web et d'utiliser un convertisseur en PHP qui se charge de lire le fichier, le transformer au format approprié et envoyer la page résultante au client qui interprète alors le script javascript pour superposer aux cartes de Google Maps le trajet parcouru. Cette solution a cependant un inconvénient majeur en terme de bande passante : le fichier NMEA contient beaucoup d'informations inutiles pour l'application qui nous intéresse ici, et le fichier placé sur le web pour traitement est plus volumineux que nécessaire. La première tâche du script PHP sera donc de ne conserver que les trames GPGGA qui nous concernent. Nous utiliserons

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:v="urn:schemas-microsoft-com:vml">
<head>
<title>NMEA to Google Maps converter using PHP </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="description" content="test - powered by Google Maps" />

<script src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=ABQIAAAA5Q7z5crwfl...[illegible]...hgiECJ2hRRd1mQtnWmN97u8JRq0x8jxwTz8w"
  type="text/javascript">
</script>

<style type="text/css">
            v\:* {
            behavior:url(#default#VML);
            }
            html, body, #map
            {
              width: 100%;
              height: 100%;
            }
          body            {
            margin-top: 0px;
            margin-right: 0px;
            margin-left: 0px;
            margin-bottom: 0px;
            }
</style>
</head>

<body>
<div id="map"> </div>

<script type="text/javascript">
//<![CDATA[            // check for compatibility

if (GBrowserIsCompatible()) { // call info window opener for the given index

  function makeOpenerCaller(i) {
     return function() { showMarkerInfo(i); };
  }               // open an info window

  function showMarkerInfo(i) {
     markers[i].openInfoWindowHtml(infoHtmls[i]);
  }               // create the map

     var map = new GMap(document.getElementById("map"));
     map.setMapType(G_SATELLITE_TYPE);
     map.addControl(new GLargeMapControl());
     map.addControl(new GMapTypeControl());
     map.addControl(new GScaleControl());

     if (window.attachEvent) {
       window.attachEvent("onresize", function() {this.map.onResize()} );
     } else {
       window.addEventListener("resize",function() {this.map.onResize()},false);
     }               // add a polyline overlay
     var points = new Array();
     // fonctions pour reduire la taille de la page HTML generee
     function p(lon,lat) {points.push(new GPoint(lon,lat));}
     function o() {
         map.addOverlay(new GPolyline(points, "#FF0000", 2, .75));points=[];}

<?php
$file = "test.nmea";
$handle = fopen($file, "r") or exit("Unable to open file!");
$n = 0;
$commence=0;
$france=1;

   do {
     do {$buffer = fgets($handle, 4096);}
         while ((strpos($buffer,'GPGGA')===false) && (!feof($handle)));
     if (!feof($handle)) {
       $n++;
       if ( $n == 100) {
          $n=0;
        if ($commence == 1) {
            printf("o();\n");
            $commence=0; // ne PAS faire de Polyline sans point
            }
       }
       list($rien1,$heure,$lat,$N,$lon,$E,$pasalt)=explode(",",$buffer);
    if ((preg_match("/[0-9]{4}\.[0-9]{4}/i",$lat)) &&
        (preg_match("/[0-9]{4}\.[0-9]{4}/i",$lon))) {
       $lati=floor($lat/100);
       $loni=floor($lon/100);
       $lati=$lati+floor( (($lat/100)-$lati)/60*100*1000000)/1000000;
       $loni=$loni+floor( (($lon/100)-$loni)/60*100*1000000)/1000000;
       if (($lati > 40 )&&($lati<90)&&($loni>0)&&($loni<60)&&$france!=0) {
            print "map.centerAndZoom(new GPoint($loni, $lati),5);";
            $france=0;}
       if ($lati > 40 ) {
            $commence=1; // on a commence a tracer la courbe
            print "p($loni,$lati);";}
       }
     }
   } while (!feof($handle));
   printf("map.addOverlay(new GPolyline(points, \"#FF0000\", 2, .75));");
   fclose($handle);
?>

   map.addOverlay(new GPolyline(points, "#FF0000", 2, .75));
   } else {
   document.getElementById("quicklinks").innerHTML="web browser not compatible"
   }
//]]>
</script>

</body>
</html>
```

TAB. 6 – Script PHP pour la génération de pages web incluant les points d'un fichier NMEA pour superposition aux images aériennes issues du serveur Google Maps.

dans ce nouveau script PHP (tableau 6) les mêmes expressions régulières que nous avions utilisé auparavant dans les scripts shell, accessibles grâce à la fonction `preg_match()` : bien que le script PHP soit très ressmblant à celui chargé de traiter les fichiers KML, nous le reproposons ici dans son intégrité par souci de clareté. La structure de la page a changé puisque nous ne faisons maintenant plus qu'un unique appel à PHP et l'ouverture de fichier cible associé, et la distinction entre la phase recherche de la première coordonnée (afin de centrer la carte à afficher sur la trace) et la suite du traitement du fichier s'obtient en définissant un drapeau (`france=0 ;` dans cet exemple). Nous avons ici dû ajouter la vérification de la validité des informations de latitude-longitude (hypothèse sur le nombre de décimales présentes) et la conversion en fractions de degrés. Une capture d'écran du résultat de ce script est présentée sur la Fig. 11.

Un second point lié à la bande passante concerne la taille de la page générée : un trajet long peut facilement générer une page de quelques mégaoctets qui, sur une connexion réseau un peu lente, engendre un timeout de la part du client web [10]. Il est donc fondamental de viser à générer une page aussi petite que possible. Nous avons adopté 3 solutions :

1. utilisation de fonctions aux noms très courts appelées pour chaque nouveau point ajouté à la liste d'éléments de chaque polygone, par exemple par `function p(lon,lat) {points.push(new`

[10] une façon de cacher ce problème sur Firefox est décrite à `http://www.itchyhands.com/2006/01/07/fix-firefox-unresponsive-script-warnings` : accéder à l'url `about :config et modifier la variable dom.max_script_run_time à une valeur de l'ordre de 20. Cette procédure ne semble pas fonctionner avec Mozilla.`

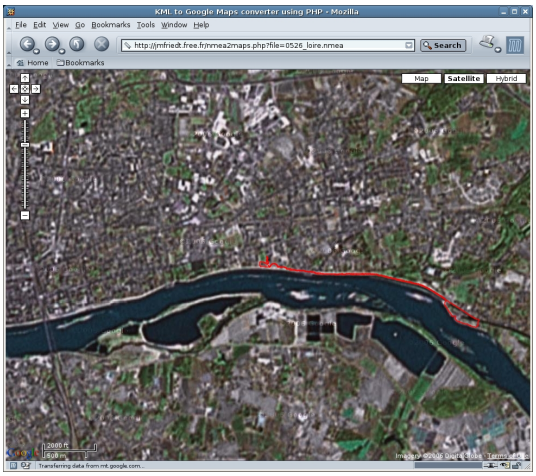

FIG. 11 – Exemple d'une conversion d'un fichier brut NMEA mis à disposition sur le web pour un affichage sur Google Maps. Saint Jean de Braye et Combleux près de Orléans.

`GPoint(lon,lat));}` et `function o() {map.addOverlay(new GPolyline(points, "#FF0000", 2, .75));points=`

`;}` qui remplace donc respectivement 23 caractères et 64 par 1 seul, résultant en une économie considérable de bande passante,

2. réduction du nombre de décimales pour ne garder qu'une précision significative. Par défaut PHP nous founit le résultat en calcul flottant d'un quotient avec 13 décimales. Un degré de latitude correspond à $\frac{40000}{360} = 111$ km donc pour avoir des points définis avec une résolution de 10 cm (bien en deça de la variance sur la position des récepteurs utilisés ici) nous nous contenterons de ne garder que 6 décimales par l'utilisation de la commande `$lat=floor($lat*1000000)/1000000;`,
3. enfin une amélioration que nous n'avons pas pris le temps d'implémenter du fait de nos trajets majoriairement pédestres est l'élimination de points successifs colinéaires et donc apparaissant comme alignés sur une trace. Ce calcul sera surtout efficace lors de la présence de longues lignes droites telles que observées lors de trajets en train : dans ce cas, 3 points A, B et C successifs sont alignés si l'angle (AB, AC) est inférieur à une valeur seuil θ_l ce qui se traduit par le calcul classique du produit scalaire $\frac{||\vec{AB}.\vec{AC}||}{||\vec{AB}||.||\vec{AC}||} > \cos(\theta_l) \simeq 1 - \frac{\theta_l^2}{2}$. Une autre implémentation plus grossière de ce type d'algorithme est proposée dans [6, p.140].

Nous proposons dans le tableau 7 une implémentation sous Matlab/Octave du calcul de l'angle entre 3 points successifs et l'élimination des points pour lesquels cet angle est inférieur à un seuil fourni en paramètre, ou dont la distance au point précédent est inférieur à une valeur minimum que nous avons arbitrairement sélectionné à 2×10^{-6} soit une vitesse de déplacement de 0,8 km/h à l'équateur. La figure 12 démontre que pour un choix d'angle seuil de l'ordre 0,5 degrés, un nombre substantiel de points est éliminé sans pour autant affecter l'aspect général de la trace. L'élimination des points considérés comme colinéaires permet d'éliminer de 20 à 50 % des points sans affecter notablement le rendu visuel de la trace (Fig. 12).

7 Utilisation des traces GPS dans GRASS

Cette présentation ne saurait se prétendre complète sans une mention au logiciel GRASS (*Geographic Resources Analysis Support System*), référence dans le domaine de la gestion géographique

```
function reduit=angle_gps(nom,ang_lim)
% prend nom_fichier[.dat] et l'angle seuil

eval(['load ',nom,'.dat']);
eval(['x=',nom,';']);
eval(['clear ',nom]);

% version Matlab
t=diff(x(:,1:2));
an=diff(angle(t(:,1)+i*t(:,2)));
a=find(abs(an)>ang_lim);
XX=x(a,1); YY=x(a,2);
reduit=[XX YY];
% fin version Matlab

% version C
for n=2:length(x)
    t(n-1,1)=x(n,1)-x(n-1,1);
    t(n-1,2)=x(n,2)-x(n-1,2);
    norm=sqrt(t(n-1,1)^2+t(n-1,2)^2);
    if (norm>2E-6) an(n-1)=acos(t(n-1,1)/norm); % 2E-6=0.8 km/h
    else an(n-1)=0;end;
end

k=1;
for n=1:length(an)-1
   if (abs((an(n+1)-an(n)))>ang_lim) r(k,:)=[x(n,1) x(n,2)];k=k+1;end;
end
% fin version C

length(x)
length(reduit)
length(r)
reduit=r;
```

TAB. 7 – Script Matlab/Octave nommé `angle_gps.m` calculant l'angle entre 3 points successifs et chargé d'éliminer les points pour lesquels cet angle est inférieur à un seuil fourni par l'utilisateur. Ce script prend en argument un nom de fichier sans son extension (supposée être `.dat`), contenant deux ou trois colonnes avec les latitudes, longitudes et altitudes respectivement (tel que issu de `deg2dec.m` par exemple), et un angle seuil en radians (une valeur de 0,01 radians semble raisonnable). Deux implémentations de ce code réalisant la même opération sont proposées : une en calcul matriciel typique de Matlab, l'autre avec des boucles typique d'un langage de type C.

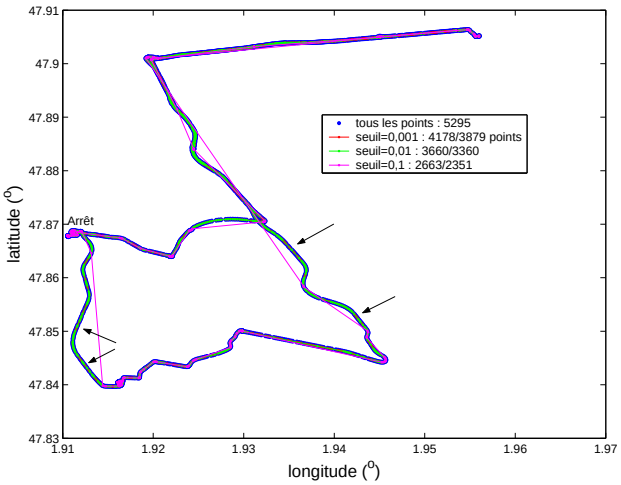

FIG. 12 – Traitement d'une trace obtenue à Orléans illustrant la réduction du nombre de point affichés en fonction de l'angle seuil entre 3 points adjacents. Pour un angle seuil de 0,001 et 0,01 radians, aucune différence perceptible au niveau du tracé n'est visible par rapport à la trace contenant tous les points. Avec un angle seuil de 0,1 radians, la perte d'information est nettement visible. Les valeurs indiquées dans la légende correspondent au nombre de points restant à tracer après traitement du fichier : le premier nombre correspond à l'élimination des points considéré comme colinéaires, et le second nombre après élimination des points considérés comme trop proches (norme inférieur à 2×10^{-6}). Ce second traitement élimine principalement des points autour de la zone annotée par "Arrêt" dans la figure, pendant laquelle le véhicule a été stationné. Les flèches attirent l'œil sur quelque différences entre les traces issues d'un seuillage à 0,01 radians et à 0,001 radians.

des données (*Geographical Information System, GIS*). GRASS suit la philosophie d'unix en se présentant comme un ensemble de petits modules puissants appelés séquentiellement lors du traitement d'informations. GRASS s'installe trivialement sous GNU/Debian (Sarge) par `apt-get install grass` qui fournit la version 6.0.0 à la date de rédation de cet article (début Août 2006). Notre introduction aux commandes de base de ce logiciel nous a été fournie par [4, pp.326-355] bien

que la version plus récente de GRASS utilisée ici semble rendre un certain nombre de commandes présentées dans cet ouvrage obsolète.

Nous allons nous intéresser à la présentation de données acquises lors de randonnées dans la vallée de Chamonix : nous désirons superposer les traces acquises avec des information de topographie du terrain. Nous devons donc résoudre deux problèmes :

1. lire nos propres données dans GRASS et les incorporer aux bases de données existantes
2. obtenir des informations topographiques de la région qui nous intéresse, les incorporer dans la base de données de GRASS et les afficher,
3. exporter les résultat en un format utilisable dans ce document.

Familiarisons nous dans un premier temps avec GRASS en résolvant le premier point. Nous avons vu plus haut comment, au moyen de scripts shell et de octave, convertir des fichiers NMEA en des listes de données sous la forme de 3 colonnes représentant la longitude, la latitude et l'altitude de chaque point enregistré (la dernière étape avant la génération d'un fichier KML par exemple, juste avant l'ajout de l'entête et le remplacement des espaces par des virgules). Nous partons par exemple d'un fichier nommé `060729_argentiere.dat` dont le début se présente sous la forme

```
6.92566666666667 45.9809566666667 1251.5
6.92558 45.9809533333333 1233.8
6.92557333333333 45.9809433333333 1231.2
6.92557333333333 45.9809266666667 1227.2
6.92555166666667 45.98091 1220.4
6.92552833333333 45.9808883333333 1212.8
...
```

et ainsi de suite pour un total de 34311 points.

GRASS nécessite dans un premier temps la définition de la zone sur laquelle nous allons travailler. En l'absence de base de données *a priori* sur lesquelles travailler, nous définissons une nouvelle zone vierge :

1. pour une première utilisation, créer un répertoire dédié aux données utilisées par GRASS : dans mon cas `/home/jmfriedt/grass`
2. lancer GRASS par `grass60`
3. créer une nouvelle carte sur laquelle nous allons travailler : `Create New Location` que j'appellerai `jmfriedt` avec un Mapset du même nom. Cette carte n'existant pas, GRASS nous demande s'il faut la créer : nous désirons travailler avec des informations de Latitude-Longitude (B) avec des données au format WGS84 (le format dans lequel les données GPS sont fournies) : nous remplissons ainsi le champ `datum name`, sans transformation additionnelle. Ce mode de représentation des coordonnées spécfie aussi l'ellispoïde associé (`ellipsoid name`) et il ne nous reste plus qu'à créer un espace de travail incluant toute la région de l'Europe qui nous intéresse. La France par exemple s'étend de 5 degrés ouest à 8 degrés est et de 42 à 52 degrés nord. Nous utilisons ces paramètres dans le choix des dimensions initiales de la carte. La validation des paramètres se fait par ESC suivi de ENTER.
4. une fois le carte nommée `jmfriedt` comme nom de LOCATION et de MAPSET crée, nous la validons une fois de plus par la paire ESC puis ENTER.
5. nous observons alors le lancement d'une interface Tk et obtenons accès à une ligne de commande GRASS :
   ```
   Welcome to GRASS 6.0.0 (2005)
   ...
   GRASS 6.0.0 (jmfriedt) :/home/jmfriedt/grass >
   ```

Ayant maintenant défini un terrain de jeu vierge, notre première tâche consiste à importer des données au format ASCII présentées sous forme de colonnes contenant respectivement la longitude, la latitude et un commentaire (ce commentaire dans notre cas est l'altitude). La commande

pour atteindre ce but est `v.in.ascii` qui, comme son nom l'indique, convertit un fichier ASCII en vecteur de points. Un certain nombre d'arguments sont fournis tels que le nom du fichier, le nom du vecteur de sortie et le caractére de séparation des colonnes. Dans notre cas la commande se résume en :
`v.in.ascii input=060729_argentiere.dat output=waypoint format=point fs=space`
qui va générer le vecteur nommé `waypoint` à partir du fichier `060729_argentiere.dat` décrit auparavant. Noter que toutes les commandes shell sont accessibles depuis GRASS. Lors de la conversion du fichier, GRASS nous informe que le format de fichier est bien reconnu (`Maximum number of columns : 3` et qu'il a bien lu le bon nombre de points (`Number of points : 34311`) tel que nous le confirme `wc -l 060729_argentiere.dat`. Afin de tracer ces points, il nous faut définir un terminal de sortie qui sera dans un premier temps l'affichage graphique : `d.mon start=x0`. Finalement, nous traçons ce vecteur avec un certain nombre de paramètres explicites : `d.vect map=waypoint icon=basic/diamond size=8 color=green fcolor=red`. Nous pouvons zoomer en une région de la carte par `d.zoom` qui nécessite de bien lire les instructions pour en comprendre le bon fonctionnement (bouton de gauche pour le premier point, bouton du milieu pour le second point, bouton de droite pour reculer). Le résultat présenté sur la figure 13 n'a à peu prés aucun intérêt par rapport à la sortie que fournirait `gnuplot` avec le même fichier. Nous allons cependant voir plus bas tout l'intérêt de ce petit exercice par la superposition de nos traces aux informations topographiques du terrain.

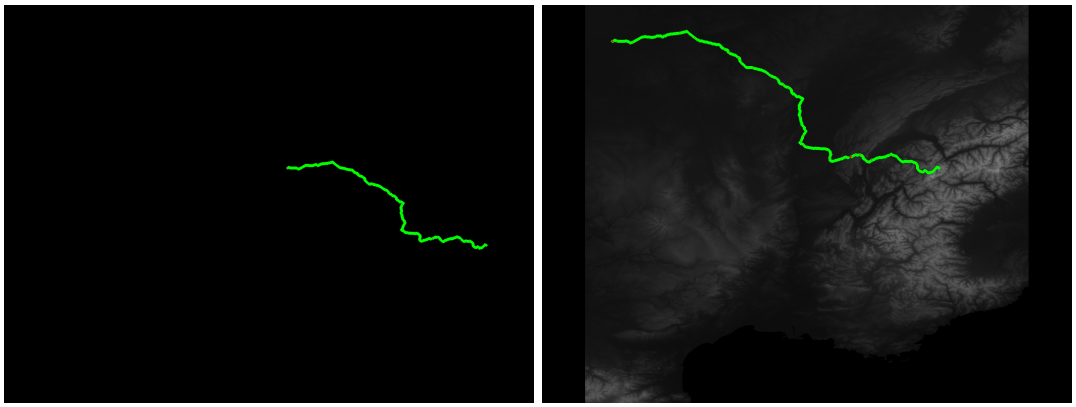

FIG. 13 – Gauche : tracé d'un trajet Chamonix-Orléans au moyen de GRASS, démontrant la capacité à lire un fichier de points au format ASCII. Droite : les mêmes données, superposées aux informations topographiques de la région des Alpes.

Nous avons déjà mentionné que les données topographiques pour l'Europe ne sont pas fournies gratuitement par les agences nationales d'études géographiques. Aussi devons nous faire appel aux bases de données américaines GLOBE (*Global Land One-kilometer Base Elevation*) disponibles à `http://www.ngdc.noaa.gov/mgg/topo/globeget.html`. Il s'agit de fichiers relativement volumineux – 129,6 MB après décompression – avec une résolution latérale de l'ordre du kilomètre. La NOAA qui distribue ces données propose aussi un tutoriel pour importer ces données dans GRASS : nous suivrons les étapes décrites à `http://www.ngdc.noaa.gov/mgg/topo/report/s11/s11Gvi.html`. De façon résumée,

1. nous récupérons le fichier `grasglob.tar` à `ftp://ftp.ngdc.noaa.gov/GLOBE_DEM/data/elev/grass/grasstar` qui contient les scripts nécessaires à l'importation de ces bases de données et le désarchivons dans notre répertoire de travail, créant ainsi une arborescence `grass/users`,
2. nous téléchargeons les bases de données topographiques qui nous intéressent (la base de donnée incluant les Alpes est nommée G10G) à `http://www.ngdc.noaa.gov/mgg/topo/gltiles.html` et en sélectionnant la région couvrant de 0 à 50 degrés nord et 0 à 90 degés est. Le fichier obtenu est `g10g.gz`

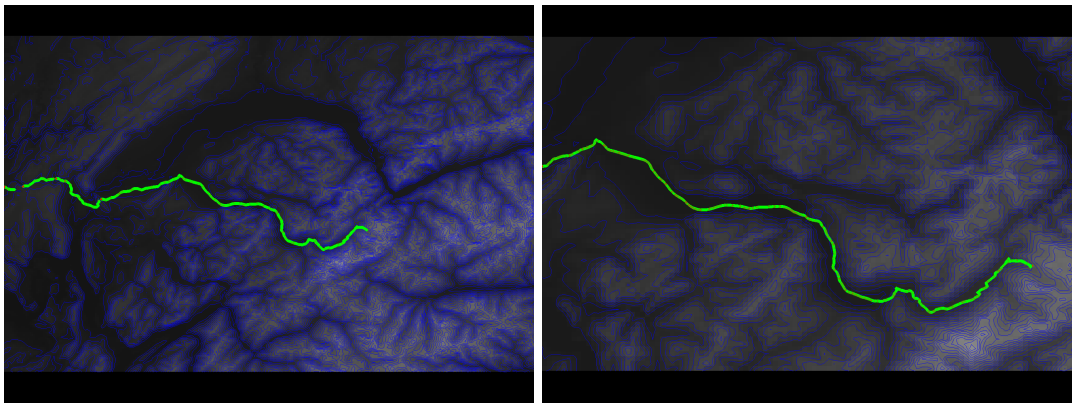

FIG. 14 – Superposition d'une information topographique en tons de gris et sous forme de lignes de niveau espacées de 200 m avec les points GPS enregistrés lors d'un départ de la vallée de Chamonix en direction de Orléans. Le zoom à droite permet de bien distinguer les différentes lignes de niveau mais met aussi en évidence la résolution spatiale médiocre des informations topographiques.

3. nous décompressons ce fichier et le renommons `g10g.pc`
4. nous changeons son endianneess (opération nécessaire sur un processeur Intel sur lequel ce test a été fait) par `dd if=g10g.pc of=g10g.ux conv=swab` et plaçons le fichier `g10g.ux` résultant dans le sous répertoire `grass/user/cell` de notre répertoire de travail.

Au lancement de GRASS, nous n'allons plus créer une carte vide mais allons charger le monde décrit dans les fichiers de configurations fournis ci-dessus. Ainsi, nous lançon `grass60` qui cette fois nous propose `Location : grass` et `Mapsets : user`. Ayant sélectionné ces options, nous cliquons sur `Enter GRASS`. Une nouvelle fonction, `g.region`, permet de charger une carte topographique d'une région du monde. Nous lançons donc `g.region rast=g10g.ux` qu'il faut ajuster afin de gérer correctement les valeurs sur 16 bits : `r.mapcalc 'g10g=if(g10g.ux-32768,g10g.ux-65536,g10g.ux,g10g.ux)'`. Cette opération prend environ 4 minutes sur une Pentium II cadencé à 400 MHz et possédant 128 MB de RAM, pour finalement générer la variable `g10g` contenant les données topographiques qui nous intéressent. Noter qu'un peu plus de 200 MB d'espace libre sur le disque dur sont nécessaires puisque le contenu de la variable y est stocké. La liste des objets bitmaps disponibles, `g.list type=rast`, nous montre bien cette variable dont le contenu s'affiche par `d.rast g10g`, toujours après avoir défini le terminal de sortie `d.mon start=x0`. Au lieu d'une image colorée en fonction de l'altitude nous désirons assigner à cette carte un dégradé de gris : `r.colors map=g10g color=grey` puis pour réafficher `d.redraw`. L'arc des Alpes est alors visible dans le coin en haut à gauche.

Nous ajoutons maintenant nos données en suivant la séquence vue précédemment, et obtenenons les figures telles que présentées sur la Fig. 13 à droite. Afin d'exporter ces images au format PNG, nous avons remplacé le terminal graphique par la sortie dans un fichier par `d.mon start=PNG` après avoir défini quelques variables d'environnement sous GRASS :

```
export GRASS_WIDTH=1600
export GRASS_HEIGHT=1200
export GRASS_TRUECOLOR=TRUE
export GRASS_BACKGROUNDCOLOR=000000
export GRASS_PNGFILE=topographie.png
```

Le tracé s'obtient en retappant les commandes `d.vect` et `d.rast` et en achevant le tracé par `d.mon stop=PNG`. Une fois le fichier d'image obtenu, nous repassons au terminal graphique par `d.mon select=x0`.

Afin d'ajouter des courbes de niveaux au fond de carte représentant l'élévation par des tons de gris, nous allons transformer les informations topographiques spatiales en données vectorielles

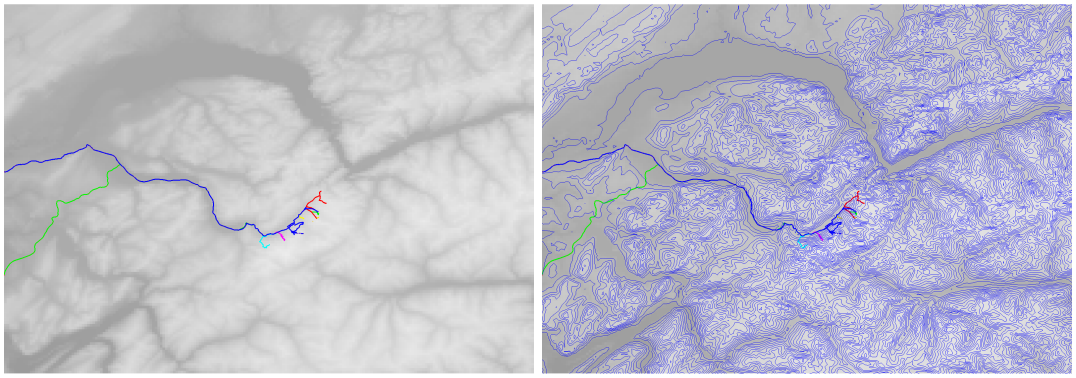

FIG. 15 – Gauche : superposition d'une information topographique dont les tons de gris évoluent avec le logarithme de l'élévation (et non linéairement comme auparavant sur la Fig. 14), et à droite ajout des lignes de niveau espacées de 200 m tel que décrit dans le texte. Les tracés GPS sont des randonnées dans la vallée de Chammonix : la coloration par une échelle logarithmique des tons de gris permet de bien distinguer les structures géographiques, notamment le lac Léman au nord.

contenant les coordonnées des lignes de niveau. La conversion d'une image bitmap dont l'intensité représente une altitude en un ensemble de courbes de niveau commençant au niveau de la mer (0) et especées de 200 m s'obtient par : `r.contour in=g10g out=cont200 min=0 step=200` dont le résultat est placé dans la variable `cont200`. Le résultat est affiché sur la carte par `d.vect cont200 color=blue` pour donner le résultat visible sur les Figs. 14 et 15.

8 Conclusion

Nous avons présenté un circuit électronique d'acquisition de trames GPS sur support de stockage de masse non volatile MMC. Ce circuit est caractérisé par sa capacité à conserver une grande quantité de données, un volume suffisamment réduit pour tenir dans une poche avec sa source d'énergie, et de nécessiter une tension suffisamment faible pour fonctionner sur 4 accumulateurs NiCd ou NiMH. Ce circuit a fonctionné sans échec pendant plus de 6 mois, y compris auprès d'usagers sans connaissances détaillées sur son principe de fonctionnement.

Nous nous sommes efforcés de diffuser par divers moyens les données ainsi accumulées : par fusion avec des données existantes telles que fournies par Google Maps, Google Earth et GRASS, ou en partageant avec des projets de cartographie libre tels que UPCT. L'ensemble des outils développés pour l'acquisition et la conversion des données a été décrit afin de permettre au lecteur de les adapter à ses propres applications.

Notre souhait serait désormais que ce circuit puisse servir de base au développement de tels projets aujourd'hui restreints à un public déjà équippé de récepteurs relativement onéreux. Les lecteurs intéressés par ce montage électronique sont encouragés à contacter les auteurs puisqu'une réalisation en quantités importantes doit permettre d'abaisser les coût d'achat des composants.

Références

[1] J.-M Friedt, É. Carry *Enregistrement de trames GPS – développement sur microcontrôleur 8051/8052 sous GNU/Linux*, GNU/Linux Magazine France **81** (Février 2006)

[2] Y. Guidon *Introduction à la compression de données : mise en évidence de l'entropie*, GNU/Linux Magazine France **74** (Juillet/Août 2005), pp.46-61

[3] *Interview de Michel Bondaz UPCT.ORG*, GNU/Linux Magazine France **73**

[4] S. Erle, R. Gibson & J. Walsh, *Mapping Hacks – Tips and tools for electronic cartography*, O'Reilly (2005)

[5] J.P. Snyder, *Flattening the Earth – Two thousand years of map projections*, The University of Chicago Press (1993), pp.178-183

[6] R. Gibson & S. Erle, *Google Maps Hacks – Tips and tools for geographic searching and remixing*, O'Reilly (2006)