

Un petit GPS avec ARDUINO

Par Nulentout : Jeudi 8 mai 2025.

Chère lectrice, cher lecteur, si vous vous êtes égarés sur le gros didacticiel (1) nommé 230 Expériences "amusantes" avec Arduino, dans les conclusions et les regrets en fin de document était exprimé mon regret de ne pas avoir réussi à utiliser les petits modules GPS que l'on peut facilement s'approvisionner sur la toile pour nos cartes électroniques favorites. J'avoue que parfois ce genre d'échec me titille au point de me pousser à m'acharner un peu au "chevet du patient", sans pour autant que la situation ne tourne à l'obsession. Et bien pour ce petit module, avant de le mettre définitivement à la réforme, j'ai décidé dans une ultime opération de sauvetage, d'appliquer la *méthode de la force brute* !



Fig.1

Comme strictement aucune des trois bibliothèques testées n'a donné de résultat, toutes ont refusé de dialoguer avec la ligne de type **SCI** du composant, j'ai décidé d'analyser en détails la documentation technique de ces modules, et "d'extraire par mes propres procédures" les informations sérielles fournies par ces dispositifs. Ce qui m'a encouragé à poursuivre mes efforts, c'est d'avoir appris que ces unités dialoguent en codes ASCII à 9600baud et respectent un protocole nommé **NMEA** qui au final ne semblait pas trop compliqué à décortiquer. S'il est certain que d'avoir été obligé de créer l'intégralité du code sans le secours d'une bibliothèque, au final la satisfaction d'y être arrivé n'est que plus importante. C'est en 2019 que j'avais commandé trois variantes de modules GPS pour Arduino. Tous durant mes longues tentatives se sont montrés équivalents.

Restant fidèle à un principe qui consiste à refuser toute frustration pour celles et ceux qui *désireraient tenter l'expérience avec le minimum d'investissement*, c'est à dire uniquement l'achat du module GPS, et bien qu'il s'agisse d'une application complète qui aboutit à un appareil autonome et portatif comme visible sur la Fig.2, ce tutoriel sera accompagné de plusieurs expériences ne nécessitant que le **Moniteur vidéo de l'IDE**. Naturellement, vous pouvez vous contenter de téléverser les démonstrateurs et vous faire directement plaisir en vous contentant du résultat. Toutefois, pour celles et ceux qui désirent en savoir plus et bien assimiler l'aspect technique et logiciel, le didacticiel va entièrement détailler l'approche technique et le développement. Il s'agit

fondamentalement d'une suite "naturelle" au didacticiel sur les 230 expériences.



Codeur
incrémental
rotatif

Fig.2

(1) Le didacticiel sur les 230 Expériences "amusantes" avec Arduino est disponible sur : <https://www.robot-maker.com/ouvrages/wp-admin/post.php?post=12510&action=edit>

01) Un fifrelin indispensable de théorie sur le protocole NMEA 0183.

L'internet fourmille de sites sur lesquels on trouvera tout ce qu'il faut savoir sur les protocoles définis par l'association **National Marine Electronics Association** basée au Maryland. **La norme NMEA 0183 décrit la communication entre équipements marins dont les systèmes GPS.** Méfiez-vous, j'ai constaté sur certains sites que la vitesse de transmission sur la SCI était de 4800baud. Ce n'est plus vrai, actuellement, tout au moins pour les petits modules GPS, elle est du double. Pour vous épargner initialement d'avoir à consulter les nombreux documents traitant du sujet sur l'Internet, on va passer en revue le contenu des trames de caractères retournés par un GPS qui respecte la norme 0183. Nous allons le faire sur "du concret" par le biais de démonstrateurs.

➤ **Experience_01 : Visualisation des signaux GPS NMEA 0183.**

D'une simplicité maximale, ce démonstrateur se contente de surveiller la ligne **SCI** du petit module **NEO-6M**, et d'afficher sans aucun formatage les caractères prélevés sur la ligne série secondaire. Pour simplifier au maximum le logiciel, on réserve la voie série d'Arduino au dialogue avec le **Moniteur de l'IDE** initialisé comme pour tous les autres démonstrateurs à 57600 baud. Puis par logiciel on émule sur **D2** et **D3** (*Pris dans l'ordre des broches disponibles*) une deuxième ligne série qui va dialoguer avec le récepteur GPS à 9600bauds. Dans ce but on se simplifie la vie et l'on fait appel à la bibliothèque **<SoftwareSerial.h>** native dans le compilateur, il suffit de la déclarer. Sur la Fig.3 on a mis en évidence les "paquets" de trames qui se succèdent. Sur cette copie d'écran le GPS vient d'être mis sous tension, il n'a pas encore capté suffisamment de satellites pour extraire des données, raison pour laquelle des informations qui devrait se trouver entre les séparateurs ',' ne sont pas présentes. Il vous suffit de téléverser le démonstrateur **Expérience_01.ino** et d'observer les données sur le Moniteur que vous avez initialisé à 57600baud. Dès que vous voyez s'afficher des trames du type de celle

Version logicielle du 12/01/2025

```
$GPRMC,V,,,,,,,,,N*53
$GPVTG,,,,,,,,,N*30
$GPGGA,,,,,0,00,99.99,,,,,*48
$GPGSA,A,1,,,,,,,,,99.99,99.99,99.99*30
$GPGSV,1,1,01,02,,,09*73
$GPGLL,,,,,V,N*64
$GPRMC,V,,,,,,,,,N*53
$GPVTG,,,,,,,,,N*30
$GPGGA,,,,,0,00,99.99,,,,,*48
$GPGSA,A,1,,,,,,,,,99.99,99.99,99.99*30
$GPGSV,1,1,00*79
$GPGLL,,,,,V,N*64
$GPRMC,V,,,,,,,,,N*53
$GPVTG,,,,,,,,,N*30
$GPGGA,,,,,0,00,99.99,,,,,*48
$GPGSA,A,1,,,,,,,,,99.99,99.99,99.99*30
$GPGSV,1,1,00*79
$GPGLL,,,,,V,N*64
$GPRMC,V,,,,,,,,,N*53
$GPVTG,,,,,,,,,N*30
```

Fig.3

```
$GPGLL,4427.32733,N,0013
$GPRMC,090423.00,A,4427.
$GPVTG,,T,,M,0.162,N,0.3
$GPGGA,090423.00,4427.32
$GPGSA,A,3,24,14,23,13,1
$GPGSV,3,1,12,05,16,192,
$GPGSV,3,2,12,14,32,050,
$GPGSV,3,3,12,22,52,062,
$GPGLL,4427.32769,N,0013
$GPRMC,090424.00,A,4427.
$GPVTG,,T,,M,0.292,N,0.5
$GPGGA,090424.00,4427.32
$GPGSA,A,3,24,14,23,13,1
$GPGSV,3,1,12,05,15,192,
$GPGSV,3,2,12,14,32,050,
$GPGSV,3,3,12,22,52,062,
$GPGLL,4427.32812,N,0013
$GPRMC,090425.00,A,4427.
$GPVTG,,T,,M,0.510,N,0.9
$GPGGA,090425.00,4427.32
$GPGSA,A,3,23,13,1
$GPGSV,3,1,12,05,15,192,
$GPGSV,3,2,12,14,32,050,
$GPGSV,3,3,12,22,52,062,
$GPGLL,4427.32859,N,0013
$GPRMC,090426.00,A,4427.
```

Fig.4

de la Fig.1, c'est que le matériel est opérationnel. Pouvant fonctionner à partir de +3,3Vcc jusqu'à +5Vcc personnellement j'ai opté pour la sortie

+5Vcc régulée d'Arduino car sur des cartes de type NANO la tension de 3,3V est un peu limite. La première observation montre que chaque trame commence par un préambule mis en évidence en jaune. Le caractère '\$' est par convention le marqueur du début d'une trame. Les deux caractères qui suivent "GP" précisent que le dispositif qui dialogue sur la SCI est un GPS. Les trois lettres qui suivent caractérisent la nature du contenu de la trame. Chaque trame contient une suite d'informations diverses séparées par les délimiteurs ',' et se termine en principe par le trio "HH" où '*' est le marqueur de fin de la trame et HH un **CRC** de contrôle exprimé en **HEXADÉCIMAL**. Enfin, on peut constater que des séquences de six trames se succèdent et se répètent plus ou moins une fois par seconde environ. Lorsque plusieurs satellites ont été captés et leurs informations traitées, la LED bleue du module **NEO-6M** se met à clignoter à la cadence de délivrance des groupes et, comme montré sur la Fig.4, le nombre de séquences peut passer à huit groupes avant de retrouver un nouveau cycle d'affichage. Avant d'analyser le contenu de chaque groupe, et de chaque trame, commençons par détailler ce que représente le **CRC** et la façon dont il est calculé sur un GPS.

> Le CRC de contrôle de la réception.

Toute transmission d'information, que ce soit par voie filaire où par ondes hertziennes peut être affectée par un ou des parasites. Dans le cas d'une transmission de type binaire, codé ici en HEXADÉCIMAL, un ou plusieurs BITS peuvent être perdus ou inversés par présence d'un parasite. Aussi, le dispositif terminal qui va traiter l'information doit s'assurer que l'intégralité de cette dernière est correcte. C'est le but du **CRC**, qui est l'abréviation de **Cyclic Redundancy Code**. On peut affirmer que les **CRC** sont utilisés depuis le début des transmissions de donnée binaires, qu'elles se fassent en série ou en parallèle. Les checksums (*Sommes de contrôle*) consistent à effectuer avec chaque caractère envoyé un calcul plus ou moins compliqué, et à terminer la transmission par le résultat de ce dernier. Par exemple on effectue la somme OCTET par OCTET et l'on envoie le résultat. Sur le système de réception on va effectuer un traitement identique, et on compare notre résultat avec celui du **CRC**. Si les deux sont différents, la donnée reçue a été polluée et n'est pas fiable. Alors on traite l'erreur. Pour la norme NMEA 0183, le calcul du **CRC** est élémentaire. On se contente de faire un **OU EXCLUSIF** entre tous les caractères ASCII compris entre les délimiteurs '\$' et '*' bornes non comprises. Naturellement, lors de nos expériences on va traiter le sujet, mais à ce stade c'est prématuré, car on ne sait pas encore extraire et filtrer les données.

> Experience_02A : Mise en évidence d'un sérieux problème.

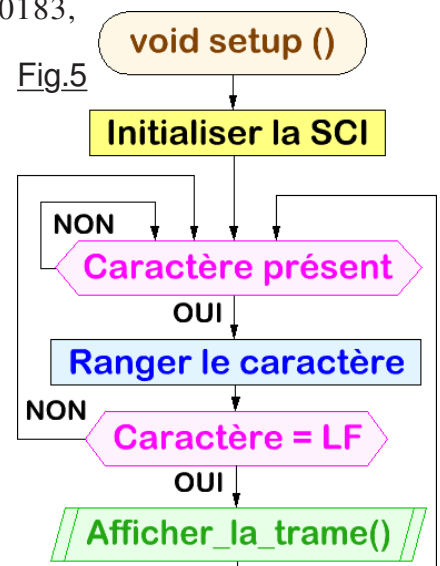
Initialement, j'ai naïvement pensé que la mise en mémoire des différentes trames serait relativement facile, l'organigramme de la Fig.5 traduisant l'algorithme idoine. Si l'on se contente de la définition d'une trame telle que la présente la norme NMEA 0183,

Version logicielle du 13/01/2025

```
Trame NMEA : $
Trame NMEA : GPRM
Trame NMEA : C,100
Trame NMEA : 459.0
Trame NMEA : 0,A,44
Trame NMEA : 27.32
Trame NMEA : 867,N,
Trame NMEA : 00131.3
Trame NMEA : 1441,E,0.387,
Trame NMEA : ,130125,,A*7C
Trame NMEA :
```

on en déduit qu'une trame commence par '\$' et se termine par le **CRC** suivi de **CR** et de **LF**. **CR** pour "Carriage Return" et **LF** pour "Line Feed". Si tel était le cas, on commence à remplir la mémoire tampon dès qu'un '\$' est reçu jusqu'à rencontrer le **LF** de fin de la trame. Alors on affiche le contenu de la mémoire tampon et on va alors lire et enregistrer la trame suivante. C'est exactement ce que fait **Expérience_02A.ino** que je vous

Fig.5



```
Trame NMEA : $GPVTG,,T,,M,0.387,N,0.71
Trame NMEA : 6,K,A*2F
```

Fig.6

```
Trame NMEA : $GPGGA,100459.00,4427.
Trame NMEA : 32867,N,00131.31
Trame NMEA : 441,E,1,08,1.18,
Trame NMEA : 170.1,M,48.1,M,
Trame NMEA : ,*52
Trame NMEA : $GPGSA,A,3,10,19,17,24
Trame NMEA : ,14,13,12,15,,,,
Trame NMEA : ,2.43,1.18,2.12*
Trame NMEA : 07
```

```
Trame NMEA : $GPGSV,3,1,11,10,20,311,
Trame NMEA : 28,12,44,217,25,
Trame NMEA : 13,31,137,17,14,
Trame NMEA : 08,049,13*70
```

invite à téléverser. Le résultat de cette stratégie est représenté sur la Fig.6 qui

visiblement montre que la trame semble contenir des passages inopinés à la ligne. Pourtant, lorsque l'on téléverse le démonstrateur **Expérience_02B.ino** on constate sur la Fig.7 qu'il n'y a pas de caractère [10] ni de caractère [13] après '\$'. C'est pour moi totalement contradictoire et incompréhensible. Si quelqu'un trouve la réponse ... je suis preneur.

CONCLUSION : Si on se réfère à **P02B** on n'a pas de passage à la ligne après '\$'. Si on considère **P02A** des passages à la ligne fantôme sont enregistré dans la mémoire tampon. Il va falloir "fonctionner en dégradé", c'est à dire filtrer ces **CR** et **LF fantôme** pour arriver à des contenus de la mémoire tampon corrects contenant une trame complète commençant par le délimiteur '\$' et s'achevant par l'annonceur du **CRC** '*'. C'est le but du prochain démonstrateur. Ces débuts ont été dans ce

développement logiciel particulièrement indigestes !

Version logicielle du 13/01/2025

```
$ G P R M C , 1 1 0 2 0 1 . 0 0 , A ,
[36] [71] [80] [82] [77] [67] [44] [49] [49] [48] [50] [48] [49] [46] [48] [48] [44] [65] [44]
[36] [71] [80] [86] [84] [71] [44] [44] [84] [44] [44] [77] [44] [48] [46] [49] [51] [50] [44]
```

Fig.7

➤ **Experience_03 : Mise en mémoire tampon des données.**

Pour être capable d'effectuer un traitement logiciel sur des données, il importe de les avoir à disposition et dans leur ensemble sous un "format propre". Hors, sur la voie série secondaires elles arrivent par paquets à une cadence rapide. Ensuite entre deux séquences on dispose d'un délai d'environ une demi-seconde pour les traiter. Il faut au préalable les logger dans une mémoire tampon. C'est exactement ce que fait le démonstrateur **Expérience_03.ino** qui procède en deux étapes. Il commence par enregistrer une trame complète en effectuant le filtrage des **CR** et **LF fantômes** dans la mémoire tampon prévue pour 82 caractères qui est la taille maximale dans la norme NMEA 0183. Puis il procède à l'affichage du contenu du tampon aboutissant au résultat de la Fig.8 qui cette fois est cohérent.

```
Trame NMEA : $GPRMC,143824.00,A,4427.34014,N,00131.31589,E,0.854,,130125,,,A*77
Trame NMEA : $GPVTG,,T,,M,0.854,N,1.581,K,A*27
Trame NMEA : $GPGGA,143824.00,4427.34014,N,00131.31589,E,1,06,3.78,177.7,M,48.1,M,,*57
Trame NMEA : $GPGSA,A,3,31,26,28,25,29,11,,,,,,,,5.24,3.78,3.64*0C
Trame NMEA : $GPGSV,3,1,11,04,00,340,,11,16,040,25,12,18,091,,18,21,168,*7B
Trame NMEA : $GPGSV,3,2,11,20,07,064,,25,57,083,13,26,24,290,27,28,65,276,33*75
Trame NMEA : $GPGSV,3,3,11,29,87,091,38,31,42,305,14,32,15,226,*4A
Trame NMEA : $GPGLL,4427.34014,N,00131.31589,E,143824.00,A,A*63
Trame NMEA : $GPRMC,143825.00,A,4427.34019,N,00131.31589,E,0.846,,130125,,,A*78
Trame NMEA : $GPVTG,,T,,M,0.846,N,1.566,K,A*2D
Trame NMEA : $GPGGA,143825.00,4427.34019,N,00131.31589,E,1,06,3.78,177.7,M,48.1,M,,*5B
Trame NMEA : $GPGSA,A,3,31,26,28,25,29,11,,,,,,,,5.25,3.78,3.64*0D
Trame NMEA : $GPGSV,3,1,11,04,00,340,,11,16,040,25,12,18,091,,18,21,168,*7B
Trame NMEA : $GPGSV,3,2,11,20,07,064,,25,57,083,13,26,24,290,27,28,65,276,33*75
Trame NMEA : $GPGSV,3,3,11,29,87,091,38,31,42,305,11,32,15,226,*4F
Trame NMEA : $GPGLL,4427.34019,N,00131.31589,E,143825.00,A,A*6F
Trame NMEA : $GPRMC,143826.00,A,4427.34028,N,00131.31585,E,0.855,,130125,,,A*77
Trame NMEA : $GPVTG,,T,,M,0.855,N,1.584,K,A*23
Trame NMEA : $GPGGA,143826.00,4427.34028,N,00131.31585,E,1,05,6.35,177.8,M,48.1,M,,*56
Trame NMEA : $GPGSA,A,3,26,28,25,29,11,,,,,,,,8.56,6.35,5.75*0C
Trame NMEA : $GPGSV,3,1,11,04,00,340,,11,16,040,25,12,18,091,,18,21,168,*7B
Trame NMEA : $GPGSV,3,2,11,20,07,064,,25,56,083,12,26,24,290,28,28,65,276,33*7A
Trame NMEA : $GPGSV,3,3,11,29,87,091,38,31,42,305,09,32,15,226,*46
Trame NMEA : $GPGLL,4427.34028,N,00131.31585,E,143826.00,A,A*62
```

Fig.8

➤ **Experience_04 : Premier programme d'exploitation.**

Maintenant que l'on peut stocker une trame complète et épurée des **CR** et **LF fantômes** on va pouvoir commencer à écrire du programme qui effectue un traitement informatique. Bien que la notion de **CRC** n'est pas celle qui nous motive le plus dans l'exploitation d'un GPS, comme en page 3 on a abordé le sujet, autant évacuer cette notion. Le traitement effectué par le démonstrateur **Expérience_04.ino** consiste à filtrer les trames de type **GPGGA**, à les afficher, et à calculer le **CRC affiché entre crochets** en fin de ligne. Quand **P04** est téléversé dans Arduino, on peut vérifier sur la Fig.8 que le résultat du calcul de vérification affiché dans la colonne violette est bien égal à la valeur du **CRC** retournée par le module GPS dans la colonne jaune. En particulier, sur la deuxième et la troisième ligne le **5D** confirme bien que cette valeur est en **HEXADÉCIMAL**. Naturellement, pour que nous puissions comparer facilement le résultat du **OU EXCLUSIF** représenté par l'opérateur '^' en langage C++ le croquis exprime la valeur également en base 16 par l'instruction **Serial.print(CRC,HEX)**. Pour calculer cette "checksums" on utilise à chaque chargement d'un caractère **OCTET** reçu sur la ligne série l'instruction **CRC = CRC ^ byte(OCTET)**. La valeur de la variable **CRC** de type **byte** est forcée à zéro avant chaque extraction d'une trame. Conformément à la norme NMEA 0183 seuls les caractères compris entre les bornes '\$' et '*' sont pris en compte.

```
Trame NMEA : $GPGGA,085306.00,4427.33439,N,00131.31662,E,1,07,1.27,170.4,M,48.1,M,,*50 >>>> CRC = [50]
Trame NMEA : $GPGGA,085307.00,4427.33453,N,00131.31644,E,1,07,1.27,169.8,M,48.1,M,,*5D >>>> CRC = [5D]
Trame NMEA : $GPGGA,085308.00,4427.33421,N,00131.31625,E,1,06,1.27,169.4,M,48.1,M,,*5D >>>> CRC = [5D]
Trame NMEA : $GPGGA,085309.00,4427.33364,N,00131.31604,E,1,06,1.27,169.1,M,48.1,M,,
Trame NMEA : $GPGGA,085310.00,4427.33341,N,00131.31592,E,1,06,1.27,168.8,M,48.1,M,,
Trame NMEA : $GPGGA,085311.00,4427.33290,N,00131.31569,E,1,06,1.27,168.4,M,48.1,M,,
```



Fig.9

HEXA

➤ Le préambule du protocole de la norme NMEA 0183.

Comme c'était annoncé en début de ce didacticiel, nous allons pas à pas décortiquer la structure des trames normalisées, progression imagée par des démonstrateurs adaptés. Nous avons vu avec **P1** que chaque trame commence par un préambule de cinq lettres qui suivent l'annonceur '\$'. Ce préambule commence toujours par **GP** qui est la signature du système qui dialogue sur la SCI, en l'occurrence **un GPS** dans notre cas. Le tableau de la Fig.10 précise la nature des informations logées dans la trame en fonction de son préambule défini sur trois lettres. On remarque que pour filtrer les trames de type **GPGGA**, ce sont les seules qui présentent une deuxième lettre du préambule égale à 'G'. C'est précisément cette particularité qui est exploitée par **P04** pour filtrer ce type de trames. Donc en fonction de ce que l'on désire afficher, nous effectuerons la sélection de la trame à exploiter par une technique analogue à celle adoptée ici.

Préambule	Nature des informations
GGA	Fix et Date. Fig.10
GLL	Coordonnées géographiques.
GSA	DOP et satellites actifs.
GSV	Satellites visibles.
RMC	Données minimales recommandées.
VTG	Direction, CAP et Vitesse.

➤ *Experience_05 : L'extraction par fracturation.*

Mais non ! Il ne s'agit pas ici de la fracturation hydraulique pour récupérer les combustibles fossiles en polluant l'environnement. On va avec le démonstrateur **Expérience_5.ino** extraire des séquences de données dans la trame de type **GGA** que nous savons isoler en mémoire tampon. Le programme va devoir extraire les données chiffrées qui nous intéressent et les transformer en équivalent numérique pour les afficher "en clair". Considérons la Fig.11 qui est extraite de la Fig.9 et décortiquons les données qui y sont listées. Dans la colonne en rose pastel, on trouve l'heure de **\$GPGGA,085311.00,** 4427.33290,N,00131.31569,E,1,06,1.27,168.4,M,48.1,M,,*53 Fig.11

réception de ces informations. Dans la colonne bleu clair et la colonne vert clair sont précisées les coordonnées géographiques du récepteur **NEO-6M**. Dans la colonne jaune on retrouve le **CRC** qui est analysé par le croquis. La valeur orange précise le nombre de satellites pris en compte pour traiter les données. Enfin, dans la colonne ocre l'altitude du récepteur GPS. Dans un premier temps,

```

10H 38Min 24Sec TU.
>>>> Trame incorrecte.
10H 38Min 26Sec TU.
>>>> Trame incorrecte.
10H 38Min 28Sec TU.
>>>> Trame incorrecte.
10H 38Min 30Sec TU.
>>>> Trame incorrecte.
10H 38Min 32Sec TU.
>>>> Trame incorrecte.
10H 38Min 34Sec TU.
>>>> Trame incorrecte.
10H 38Min 36Sec TU.
>>>> Trame incorrecte.
10H 38Min 38Sec TU.
>>>> Trame incorrecte.
10H 38Min 40Sec TU.
>>>> Trame incorrecte.

```

on va simplifier et la finalité de **P05** consiste à extraire l'HEURE de réception des données et à l'afficher en clair sur le **Moniteur** de l'**IDE**. Par ailleurs, l'affichage de ces données ne sera effectué que si elles sont cohérentes, dans le cas contraire l'opérateur sera prévenu par un texte d'erreur. Ces lignes sont présentées à l'écran environ une fois par seconde qui est la cadence des "rafales" de données sur la SCI. Lorsque vous aurez constaté le bon fonctionnement du programme qui affiche l'heure en T.U. (*Une heure de moins en hiver et deux heures de moins en été par rapport à l'heure légale en France.*) vous changez la valeur du paramètre **PERTURBER** en **true** situé en tête de listage. L'affichage devient celui de la Fig.12, car pour l'un des retours de trame sur deux l'un des caractères du **CRC** dans ce dernier est faussé. On peut ainsi vérifier la fiabilité de la vérification des trames. On constate au passage qu'il s'écoule bien une seconde entre chaque réception d'un groupe de données. Dans une application digne de ce nom, l'heure devrait être traduite en heure légale, ce sera bien entendu le cas dans certains prochains démonstrateurs. Ceci étant précisé, les techniques pour extraire les autres données vont dériver

Fig.12

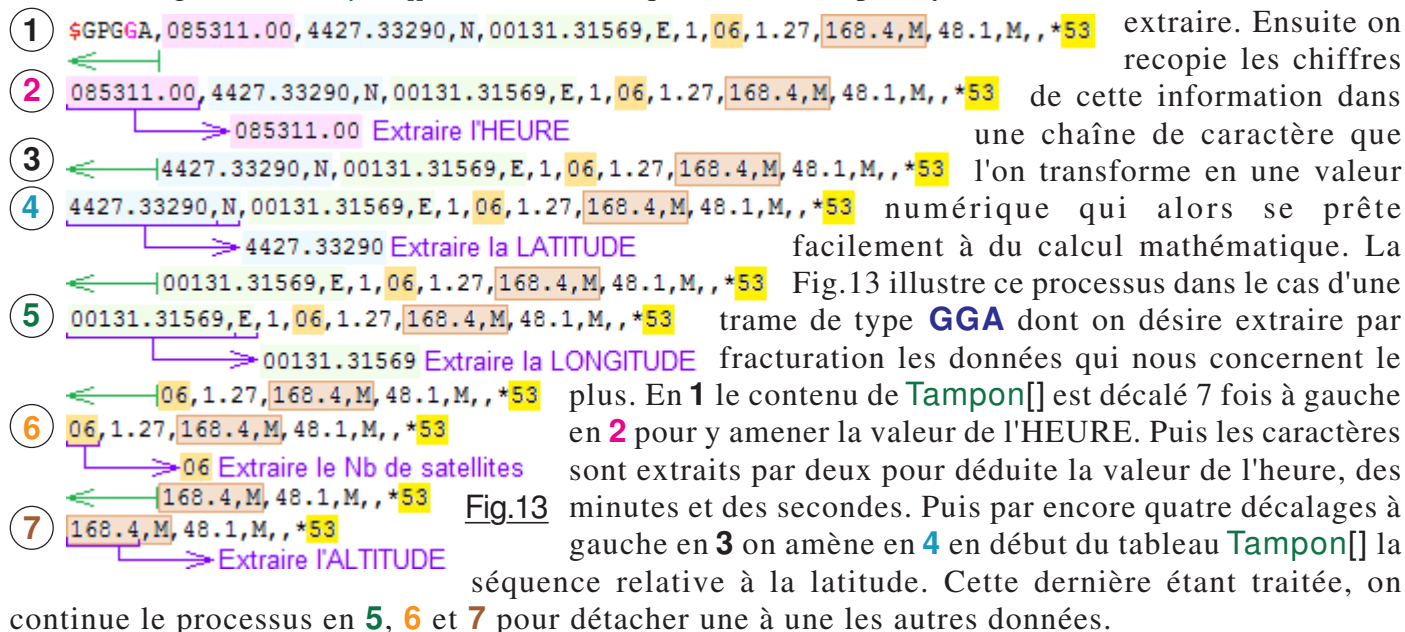
directement de ce qui vient d'être mis en place. Le chemin vers un système autonome est tout tracé ...



Ben Mòa môa je trouve que c'est quand même plus simple de regarder l'heure sur la pendule du salon que d'utiliser un appareil très compliqué de fracturation qui n'est pas hydraurucmachin !

➤ Changer de genre !

Banal et permanent en génie logiciel, on doit sans arrêt changer la nature des données. Le cas typique dans **P5** consiste à extraire des chaînes de caractères d'une mémoire tampon. Puis, transformer en nombres numériques ces éléments qui sont des suites de lettres et de chiffres. Ce petit chapitre s'adresse à celles et ceux qui ont du temps à consacrer au "comment ça marche". La technique de base qui sera utilisée chaque fois que l'on veut extraire une donnée numérique consiste à décaler à gauche **Tampon[]** autant de fois que nécessaire pour y amener le début de la donnée à



continue le processus en 5, 6 et 7 pour détacher une à une les autres données.

➤ **Expérience_06 : Décaler à gauche les valeurs numériques.**

Afin d'illustrer concrètement le propos du chapitre précédent, **Expérience_02.ino** se contente de décaler cinq fois à gauche le contenu de **Tampon[]** pour isoler par "latéralisation" les données numériques qui sont sensées nous concerner. Le résultat montré sur la Fig.14 est à comparer avec celui de la Fig.13 qui était un montage créé manuellement dans un logiciel de dessin élémentaire.

```
$GPGGA,120750.00,4427.32503,N,00131.31336,E,1,05,1.95,167.2,M,48.1,M,,*5F
120750.00,4427.32503,N,00131.31336,E,1,05,1.95,167.2,M,48.1,M,,*5F
4427.32503,N,00131.31336,E,1,05,1.95,167.2,M,48.1,M,,*5F
00131.31336,E,1,05,1.95,167.2,M,48.1,M,,*5F
05,1.95,167.2,M,48.1,M,,*5F
167.2,M,48.1,M,,*5F
```

Fig.14

02) Traitement numérique des données.

Suite logique aux préliminaires qui avaient pour but de faire un peu connaissance avec la norme NMEA 0183 on va dans ce chapitre aborder enfin l'extraction de toutes les données en vue plus tard de les présenter sur des afficheurs plus propices que le **Moniteur de l'IDE** à créer une appareil autonome. Pour ce développement logiciel on va persister à se servir de la trame de type **GGA**. L'idée de base consiste à reprendre la structure de **P06**, d'extraire toutes les données coloriées dans la Fig.13 puis de les afficher en clair sur la ligne série USB de dialogue avec le P.C.

➤ **Expérience_07 : Extraction de toutes les données de base.**

Bien qu'il soit peu probable que l'utilisateur utilisera le petit appareil dans l'hémisphère SUD au lieu d'afficher 'N' par défaut pour la latitude, on puisera l'information dans les trames GPS. Cette approche augmente la fiabilité et la perspective de vérifier que les données affichées soient cohérentes. Le démonstrateur **Expérience_07.ino** réalise une mise en page minimale pour l'affichage de l'heure et pour celui des coordonnées géographiques. Pour l'heure, ajoute si nécessaire le zéro en tête si Heure, Minutes ou Secondes sont inférieures à dix. Pour les coordonnées géographiques, comme le montre la copie d'écran de la Fig.15 elles sont proposées dans un format plus habituel de type Degrés/Minutes/Secondes que celui DD,MM.MMMMM de la donnée brute du GPS. Il faut prendre garde au fait que beaucoup de descriptions sur Internet de la NMEA 0183 font

référence à un format de type DD,MM.**MMMM** alors qu'actuellement il y a *cinq décimales pour la valeur des minutes* au lieu de quatre, et ce autant pour la longitude que pour la latitude. On se doute que dans **P7** cette constatation est prise en compte. Sur la Fig.15 la trame complète réceptionnée est affichée, avant les données traitées dans les zones colorées en rose. On peut alors comparer toutes ces informations. Il est temps d'aller faire un petit tour dans les autres types de préambules.

Fig.15

```
$GPGGA,094327.00,4427.33152,N,00131.31584,E,1,09,1.02,167.7,M,48.1,M,,*5C
09:43:27 TU >>> 44°27'20"N 1°31'19"E 167m (9)
$GPGGA,094328.00,4427.33154,N,00131.31571,E,1,08,1.09,167.5,M,48.1,M,,*57
09:43:28 TU >>> 44°27'20"N 1°31'19"E 167m (8)
$GPGGA,094329.00,4427.33150,N,00131.31560,E,1,08,1.09,167.5,M,48.1,M,,*52
09:43:29 TU >>> 44°27'20"N 1°31'19"E 167m (8)
```

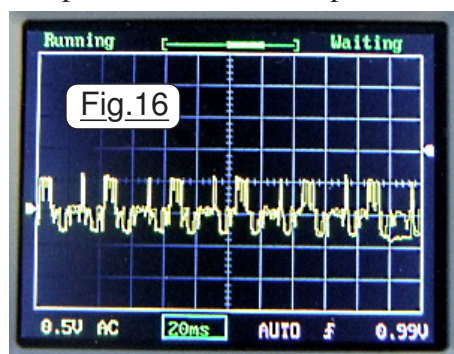
Nombre de satellites

ALTITUDE du GPS

REMARQUE : Durant le développement d'**Expérience_06.ino** je me suis heurté à un comportement vraiment étrange du programme alors que je n'avais ajouté que deux instructions banales de type **if** (**HEURE** < 10) **Serial.print('0')** qui ne peuvent faire diverger le programme. Pourtant, c'est ce qui s'est passé. J'ai donc subodoré une collision entre la **PILE** et le **TAS**. C'est la raison pour laquelle dans le démonstrateur **P06** j'ai ajouté le test de PILE dans la séquence d'initialisation, test qui sera "propagé" car sa présence n'engendre pas de gêne.

➤ Conclusions expérimentales.

A l'usage suite à de nombreux essais, je me suis rendu compte que le **+5V** la carte Arduino NANO étant alimentée par sa mini-prise USB était un peu faible et présentait un taux de "brouillage" non dérisoire. Sur l'oscillogramme de la Fig.16 on peut observer que la tension crête à crête de "l'herbe" fait pratiquement 1V, générées quand on branche le module GPS. Par contre, quand on alimente avec une pile de 9V rechargeable sur l'entrée **IN**, la perturbation est moindre car l'impédance interne est plus faible. On peut donc imaginer que le fonctionnement en version autonome



sera fiable. Si vous observez **Image 01.JPG** disponible dans le dossier **<IMAGES>** vous allez découvrir que je dispose de trois modules de type un peu différents bien que tous utilisent le même circuit intégré spécialisé. Il semblerait à l'usage, sans pour autant être formel, que plus les dimensions de la petite antenne de capture sont de tailles réduites, plus grand est le temps de synchronisation du GPS. C'est le modèle **B** dont l'antenne est un carré de 25mm de coté qui semble donner les meilleurs résultats. Il importe également d'envisager dès ce stade du développement l'intégration d'Arduino et du module **NEO-6M** dans un petit boîtier pour en

faire un appareil totalement autonome. Il faut donc vérifier que les deux éléments électroniques acceptent une promiscuité inévitable et trouver une orientation de l'antenne qui convienne. Une première étude aboutit à placer le circuit imprimé du GPS verticalement contre la carte Arduino qui est de type NANO pour des raisons d'encombrement. De ce fait, l'antenne se trouvera également dans un plan vertical. Il semble que cet attelage soit techniquement acceptable. Sur **Image 02.JPG** on observe que la LED rouge d'Arduino placé sur une plaquette à essai signale la réception d'un signal sur sa voie série. Noter que l'on ne peut pas confondre avec la **LED PW** qui signale la présence du **+5Vcc** car *sur le petit circuit imprimé cette LED "trop présente" à mon sens a été isolée*. À l'aide d'un cutter, j'ai coupé la piste qui la relie à sa résistance de limitation de courant. Sur la photographie d'**Image 03.JPG** saisie de l'autre coté du module **NEO-6M** on peut remarquer que dans l'encadré jaune la LED bleue de ce module est allumée. Dans la pratique elle s'illumine durant la transmission des données *lorsque suffisamment de satellites sont utilisables pour les calculs*.

➤ Expérience_08 : Extraction de la date.

Toujours dans l'exploration du contenu des données de la norme nous allons travailler un peu avec la trame de préambule **RMC** et passer l'affichage de l'heure de T.U. en heure légale française. Puisque le logiciel ne peut pas savoir si l'on est en heure d'hiver ou heure d'été, dans ce "Sketch" on va préciser cette information en tête de programme sous forme d'un

Fig.17

```

Version logicielle du 15/01/2025
PILE - TAS = 1458
$GPRMC,081556.00,A,4427.33046,N,00131.31886,E,0.095,,160125,,,A*72
09H 15min 56S le 16 Janvier 2025
$GPRMC,081557.00,A,4427.33049,N,00131.31891,E,0.024,,160125,,,A*70
09H 15min 57S le 16 Janvier 2025

```

paramètre **Hiver** de type booléen. Les résultats obtenus avec **Expérience_08.ino** sont montrés sur la Fig.17 qui est une copie d'écran de la fenêtre du **Moniteur** de l'**IDE** en avec la "présentation" du "Sketch" dans la zone jaune, et l'indication en zone bleue de l'espace situé entre la **PILE** et le **TAS**. Pour celle et ceux qui désirent en savoir plus sur ce thème, vous trouverez le petit fichier technique **Collision_entre_la_PILES_et_le_TAS.pdf** préservé dans le sous-dossier **<DOCUMENTS>**. Puis, le GPS étant synchronisé, se suivent à chaque seconde l'affichage du contenu de la mémoire **Tampon[]** contenant une trame de type **RMC** dans les zones vertes et l'extraction des données dans les zones roses. Contrairement au cas de la Fig.15, ici c'est l'heure légale en France qui est affichée.

➤ **Expérience_09 : Attendre des données cohérentes.**

A sa mise sous tension, le GPS n'a rien à analyser pour effectuer ses traitements, il lui faut attendre que suffisamment de satellites soient en visibilité pour en capturer les signaux et combiner ces derniers pour déterminer la position géographique du récepteur. En théorie, avec trois "balises" on peut se situer sur Terre. Toutefois, si vous consultez **Principe de base du GPS.pdf** disponible dans le dossier **<DOCUMENTS>**, vous allez constater que pour effectuer un traitement

de localisation le système attend d'avoir au moins quatre satellites fiables en visibilité. Riches de cette information on peut analyser l'historique d'un démarrage dont la Fig.18

```

Expérience_P09 du 16/01/2025
PILE - TAS = 1522
0000 Attente des satellites.
$GPGLGA,,,,,0,00,99.99,,,,,*48 (31)
0112 Attente des satellites.
$GPGLGA,,,,,0,00,99.99,,,,,*48 (31)
0113 Attente des satellites.
$GPGLGA,153534.60,,,,,0,00,99.99,,,,,*65 (40)
0114 Attente des satellites.
$GPGLGA,153535.00,,,,,0,00,99.99,,,,,*62 (40)
0115 Attente des satellites.
$GPGLGA,153536.00,,,,,0,00,99.99,,,,,*61 (40)
0352 Attente des satellites.
$GPGLGA,153930.00,,,,,0,04,1.90,,,,,*57 (39)
Reception de 05 satellites. (73)
Reception de 05 satellites. (73)
Reception de 05 satellites. (73)

```

①

Longueur

2A

2B

3

4

5

Fig.18 de caractères dans un groupe de données, lorsqu'elle est complète. **Si elle fait 73 caractères, alors les données sont utilisables** et toutes les trames sont documentées. En 2A et en

2B l'analyse se poursuit. Jusqu'à 2A les trames ne font que 31 caractères car de nombreux champs ne sont pas renseignés entre les séparateurs "virgules". Après 113 secondes de la mise sous tension, l'information de l'heure **TU** est disponible et la longueur des chaînes de caractère passe de 31 à 40. Puis en 3 l'attente se poursuit jusqu'à ce qu'en 4 le nombre de satellites utilisables dépasse le nombre minimal nécessaire de quatre. Et en 5 commence alors la phase d'exploitation possible des données. On constate dans cet exemple qu'il a fallu attendre 353 secondes pour obtenir des données utilisables soit 5 minutes et 52 secondes. Cette durée d'attente dépend directement de la configuration de la noria des satellites GPS. Par exemple sur la Fig.19 on ne doit attendre que 4 minutes et 40 secondes. L'attente est plus courte. D'une façon générale, la durée d'attente avant utilisation possible des données sera toujours de l'ordre de cinq à six minutes.

```

0279 Attente des satellites.
$GPGLGA,083733.00,,,,,0,03,2.97,,,,,*55 [7] (39)
0280 Attente des satellites.
$GPGLGA,083734.00,,,,,0,04,3.83,,,,,*51 [7] (39)
Reception de 04 satellites. (73)
Reception de 04 satellites. (73)
Reception de 04 satellites. (73)

```

Fig.19

03) Les données secondaires.

Disséminées dans les divers types de trames, bien que peu importantes aux yeux de l'utilisateur "lambda", il serait dommage dans un contexte d'expérimentation de ne pas ouvrir une petite parenthèse sur ces données complémentaires. On va commencer par celles contenues dans les trames de type **GGA** avec le démonstrateur **P10**. Pour que les données soient cohérentes dès leur affichage, à partir de ce démonstrateur on attendra systématiquement que le nombre des satellites utilisables soit au moins égal à quatre comme l'impose la normalisation NMEA 0183.

➤ **Experience_10 : Données secondaires de GGA.**

Outre la nature du positionnement, on peut obtenir dans cette trame une idée de la fiabilité de la géo-localisation. Dans ce démonstrateur nous allons nous limiter à ces deux informations secondaires. Pour extraire ces informations, nous devons au préalable les situer dans la trame de type **GGA** et les interpréter. Dans ce but la Fig.21 représente une telle trame extraite de la copie

```
Experience_P10 du 17/01/2025
PILE - TAS = 1368
$GPGGA,173120.00,4427.32958,N,00131.31495,E,1,06,1.62,155.5,M,48.1,M,,*59
Positionnement : Point calé
Dilution horizontale = 1.62 soit une precision Nominale
$GPGGA,173121.00,4427.32958,N,00131.31495,E,1,06,1.62,155.5,M,48.1,M,,*5D
Positionnement : Point calé
Dilution horizontale = 1.62 soit une precision Nominale
```

Fig.20

d'écran de la Fig.20 obtenue avec le démonstrateur **Expérience_10.ino** lorsque l'appareil alimenté depuis plus de dix minutes était "synchronisé" correctement avec **six** satellites.

①	②	③	④	⑤	⑥	⑦	⑧	Fig.21
\$GPGGA,173120.00,4427.32958,N,00131.31495,E,1,06,1.62,155.5,M,48.1,M,,*59								

- **1** : Heure d'acquisition des données en T.U.
- **2** : Latitude du récepteur GPS.
- **3** : Longitude de localisation.
- **4** : Type de **Positionnement**.
- **5** : Nombre de satellites en poursuite.
- **6** : Qualification du "FIX". (Voir le tableau Fig.22)
- **7** : Altitude en Mètres au dessus du niveau moyen des Océans. (*Mean Sea Level*)
- **8** : Correction de la hauteur du géoïde en Mètres par rapport à l'ellipsoïde WGS84. (1)

Positionnement	Nature
0	Point non calé.
1	Point calé.
2	Mode différentiel.
6	Point estimé.

Dilution horizontale de la précision. Permet de connaître la fiabilité du calcul. Fig.22
1 : Optimale, **2 à 3** : Excellente, **5 à 6** : Bonne, Supérieure à **8** : Mesure non fiable

➤ **Experience_10 : Données secondaires de GGA.**

Finalement, le lendemain de la rédaction du chapitre précédent, j'ai décidé de modifier le démonstrateur **Expérience_10.ino** pour en exploiter également les données d'altitude. La Fig.23 montre une nouvelle copie d'écran de la fenêtre du **Moniteur** de l'**IDE**. Outre l'altitude et sa correction en hauteur, on peut remarquer que la date de la version a été mise à jour. *Je vous invite au passage à aller vous renseigner sur ce que signifie géoïde **WGS84** et la notion de **MSL**.*

Fig.23

```
Experience_P10 du 18/01/2025
PILE - TAS = 1328
$GPGGA,084419.00,4427.32878,N,00131.31604,E,1,07,1.08,158.6,M,48.1,M,,*55
Positionnement : Point calé
Dilution horizontale = 1.08 soit une precision Nominale
ALTITUDE : 158.6m Correction en hauteur : 48.1m
```

(1) : **WGS84** est un système de coordonnées définissant un modèle de la Terre. Il est défini par un ensemble de paramètres primaires et secondaires dont les caractéristiques de la forme de l'ellipsoïde terrestre, sa vitesse angulaire, et sa masse.

les données "en temps partagé". Il se trouve qu'avec **Expérience_12.ino** on arrive à des durées de traitement et d'affichage qui sont trop importantes. Arrivé en **3** on continue notre analyse logicielle alors que la suite des données, symbolisé par **5** est tronquée. La fin de la trame en cours est perdue, et se précipite en **4** le début de la trame suivante. Du coup, le programme travaille sur des données "imprévues" et tombe dans une boucle sans fin qui affiche des '8'.

CONCLUSION : Intercaler du traitement logiciel durant la capture d'un groupe de donnée n'est pas utilisable car les temps de traitement deviennent incompatibles avec la cadence des données sur la ligne série. Inévitablement si les durées de traitements deviennent significatives, on arrive à des pertes de données. Par ailleurs, la technique qui consiste à engendrer des décalages à gauche n'est plus envisageable "en temps partagé" car elle est trop couteuse "en cycles machine".


SOLUTION : La nouvelle approche va consister à enregistrer l'intégralité des données d'un groupe qui arrivent en "rafales". Puis, le total étant alors à notre disposition, effectuer les traitements et les affichages désirés. *Les techniques de décalage du tampon de données ne sont plus pertinentes.*

➤ **Expérience_13 : Saisie en rafale des données.**

Démonstrateur qui se charge de *stocker presque* l'intégralité du contenu d'un groupe de données. Presque, car deux difficultés réelles se présente pour le cas des trames de type **GSV**. Pour comprendre la complexité qu'engendre ce type de trames, commençons Fig.26 par décortiquer le contenu d'une trame de type GSV dont l'exemple présente le cas le plus complet :

1	2	3	4	5	6	7	Fig.26
\$	G	P	G	S	V	,3,1,10,11,19,043,22,12,22,088,19,18,16,169,18,20,05,068,11*7E	
			A B C D	A B C D	A B C D	A B C D	

- **1** : Nombre de trames de type GSV dans la rafale du groupe des données. (*Ici trois.*)
- **2** : Numéro de la trame dans le groupe de données. (*Ici c'est la première.*)
- **3** : Nombre de satellites visibles. (*SV*).
- **4 à 7** : **Données relatives à un satellite** :

Cette séquence se répète jusqu'à quatre satellites par trames.		A = N° d'identification du satellite.
		B = Élévation en degrés du satellite.
		C = Azimut en degrés du satellite.
		D = Intensité de réception du signal du satellite.

On peut avoir jusqu'à quatre trames de type **GSV** dans une transmission. Ainsi la limitation pour le domaine grand public est de 16 satellites. **ATTENTION**, beaucoup de sites sur Internet stipulent que cette limitation est de trois trames GSV par groupe et de 12 satellites.

Première difficulté : N'enregistrer que les données pertinentes. Le démonstrateur **P13** effectue ce filtrage. Mémoriser l'intégralité des données d'une trame consiste à déclarer autant de tableaux que de types de trames dans un groupe. La Fig.27 va nous permettre d'optimiser la taille de ces tableaux. Un premier problème se présente : En fonction de la configuration des satellites, on peut avoir jusqu'à quatre trames de type GSV. Pour toutes les mémoriser il faudrait déclarer quatre tableaux de 59 cellules soit 236 octets de moins pour le programme et de moins entre la PILE et le TAS. *Comme les trames de type GSV ne contiennent que des informations relatives à la configuration des satellites, et qu'il s'agit de données secondaires, on ne va déclarer qu'un seul tableau, quitte à utiliser quatre groupes de données GPS successifs pour collecteur l'intégralité de ces informations.*

STRATÉGIE : On commence par attendre le début de la première trame d'un groupe. L'expérience montre que les trames sont toujours dans l'ordre de celles de la copie d'écran de la Fig.27 le caractère '\$' signalant le début d'une trame. On saute alors "GP" qui n'apporte aucune information utile. Puis, on se synchronise sur le groupe en testant le préambule **GLL**. Comme l'ordre des trames est connu, pour celles qui suivent on se contente de "sauter" le préambule et de mémoriser dans le tampon dédié. Préambule entièrement filtré commencent alors les données contenant de l'information. Tous les caractères qui suivent sont mémorisés jusqu'à rencontrer le marqueur de fin des données '*' qui ne sera pas mémorisé. On saute alors les deux caractères du CRC, le **CR** et le **LF** et le processus va recommencer pour la trame suivante. On peut remarquer que pour les trames de type **GSA** le 'A' qui suit le préambule et le '3' n'apportent pas vraiment d'information pertinente, car ce sera toujours le

\$GPGLL,4427.33423,N,00131.31717,E,140408.00,A,A*60 (40)
 \$GPRMC,140409.00,A,4427.33440,N,00131.31723,E,0.359,,190125,,,A*7B (56)
 \$GPVTG,054.7,T,034.4,M,0.359,N,0.665,K,A*29 (33)
 \$GPGGA,140409.00,4427.33440,N,00131.31723,E,1,08,1.14,164.2,M,48.1,M,,*56 (63)
 \$GPGSA,A,3,25,32,12,11,28,29,31,26,14,20,33,05,1.73,1.14,1.30*03 (50) (@)
 \$GPGSV,3,1,10,11,19,043,22,12,22,088,19,18,16,169,18,20,05,068,11*7E (58)
 \$GPGSV,3,2,10,25,05,077,,26,20,287,27,28,08,286,,29,86,175,38*2F (54)
 \$GPGSV,3,3,10,31,39,307,27,32,18,229,18*79 (32)

Fig.27

(@) : 'A' et '3' en tête' sont sautés.

cas. Ils ne sont donc pas mémorisés. Reste à optimiser la taille des tampons mémoires spécifiques à chaque type de trame. Dans ce but nous allons nous servir de la Fig.27 qui représente une copie d'écran d'un groupe contenant trois trames de type GSV. (Image 04.JPG prouve qu'il peut y en avoir jusqu'à quatre.) Les caractères ont été coloriés de la façon suivante : En rouge et en marron les préambules non mémorisés. En orange les CRC non mémorisés. Des paquets de dix sont repérés en bleu clair et en violet. Enfin les paquets non égaux à dix en rose. Il est alors facile pour chaque ligne de totaliser ce qui est bleu clair, violet et rose. Le résultat est entre parenthèses en italiques et en gris. Comme vous pouvez le constater dans [Expérience_13.ino](#) les tailles déclarées sont augmentées d'une unité pour le logement de l'octet '\0' qui marque la fin des chaînes de caractères.

➤ Temps disponible pour le traitement et l'affichage.

Comme pour chaque démonstrateur, la Fig.28 présente une copie d'écran de la fenêtre contextuelle du **Moniteur de l'IDE** et commence par la présentation du programme en zone jaune. Contrairement à ce qui se faisait avant, ici il n'est plus nécessaire d'attendre que quatre

Experience_P12 du 21/01/2025
 PILE - TAS = 1260
 Duree de la capture = 909mS
 GLL = ,,,,V,N
 RMC = ,V,,,,,,N
 VTG = ,,,,N
 GGA = ,,,,0,00,99.99,
 GSA = ,,,,99.99,99.99,99.99
 GSV 1/1 = 1,1,00

Duree de la capture = 961mS
 GLL = 4427.32542,N,00131.31822,E,164430.00,A,A
 RMC = 164431.00,A,4427.32544,N,00131.31820,E,0.042,,200125,,,A
 VTG = ,T,,M,0.042,N,0.078,K,A
 GGA = 164431.00,4427.32544,N,00131.31820,E,1,04,2.74,178.0,M,48.1,M,,
 GSA = 18,16,31,,,,,,5.82,2.74,5.14
 GSV 1/3 = 3,1,10,05,11,034,16,08,00,268,,10,04,162,,16,58,306,32

Duree de la capture = 914mS
 GLL = 4427.32544,N,00131.31820,E,164431.00,A,A
 RMC = 164432.00,A,4427.32549,N,00131.31822,E,0.144,,200125,,,A
 VTG = ,T,,M,0.144,N,0.267,K,A
 GGA = 164432.00,4427.32549,N,00131.31822,E,1,04,2.74,178.0,M,48.1,M,,
 GSA = 18,16,31,,,,,,5.82,2.74,5.14
 GSV 2/3 = 3,2,10,18,66,070,35,26,82,216,38,27,31,273,22,28,15,193,12

Duree de la capture = 900mS
 GLL = 4427.32549,N,00131.31822,E,164432.00,A,A
 RMC = 164433.00,A,4427.32554,N,00131.31817,E,0.156,,200125,,,A
 VTG = ,T,,M,0.156,N,0.288,K,A
 GGA = 164433.00,4427.32554,N,00131.31817,E,1,04,2.74,177.9,M,48.1,M,,
 GSA = 18,16,31,,,,,,5.82,2.74,5.13
 GSV 3/3 = 3,3,10,29,22,072,21,31,32,212,32

satellites au moins soient au dessus de l'horizon, l'affichage reste cohérent dès la mise sous tension comme on peut le voir dans la zone bleu clair correspondant à la mise sous tension du système. Ensuite, les groupes qui suivent sont issus du fonctionnement d'[Expérience_13.ino](#) lorsque le nombre de satellites visibles (SV) est de dix conduisant à trois trames de type GSV dans les groupes de données. Le dispositif est donc sous tension depuis plusieurs minutes. Comme chaque trame de type GSV ne peut accueillir que quatre descriptions satellitaires au maximum, pour les dix unités il faut deux trames complètes et la dernière qui n'en compte que deux. Pour mettre en évidences leurs descripteurs, sur la Fig.28 les identificateurs sont coloriés en orange. Vous pouvez constater dans les "surlignages" en vert pastel qu'il faut trois groupes de données pour lister les dix satellites. Le croquis **P13** chronomètre finement le temps nécessaire pour capturer l'intégralité de chaque groupe et le précise en tête d'affichage dans les zones coloriées en rose clair. On en déduit qu'il reste à peine un dixième de seconde pour effectuer les traitements et l'affichage. Le µP de la carte NANO est rapide, mais il faudra aussi que l'affichage ne traine pas trop !

➤ **Experience_14 : Données secondaires de GSV.**

Avec **P12** on désirait extraire les données secondaires relatives à la configuration des satellites avec les trames de type **GSV** complétant ainsi les données des trames de préambule GSA. Avec les préambules de type **GSV**, comme détaillé dans la Fig.26, on peut obtenir l'élévation et l'azimut des divers satellites en visibilité ainsi que la puissance de leur signal en réception. On peut alors se faire une idée de la configuration spatiale telle que celle montrée en Fig.6 de la page 4 de **Principe de base du GPS.pfd** disponible dans le sous-dossier <DOCUMENTS> et je vous invite fortement à le consulter. Dans **P12** les affichages ont été réduits au maximum pour gagner du temps, au risque de ne pas être "parlants". *Cette concession à la lisibilité n'était pas suffisante* et le démonstrateur divergeait. Le démonstrateur **Expérience_14.ino** respecte la nouvelle approche, avec pour bénéfice un affichage des données bien plus explicite.

Deuxième difficulté : L'intensité de réception du signal n'est pas toujours présente.

C'est précisément ce constat qui impose d'extraire les données satellitaires par simple exploitation d'indices qui se résume à une technique de décalage d'un pointeur dans le tampon pour explorer entièrement la trame de type **GSV**. Nous savons qu'une telle trame peut contenir entre une et quatre définitions de caractéristiques de satellites. Aléatoirement l'intensité de réception

Combinatoire des signaux "nuls"

4		3		2		1	
1111	4	111	3	11	2	1	1
1110	3	110	2	10	1	0	0
1101	3	101	2	01	1		
1011	3	011	2	00	0		
0111	3	100	1				
1100	2	010	1				
0110	2	001	1				
0011	2	000	0				
1010	2						
0101	2						
1001	2						
1000	1						
0100	1						
0010	1						
0001	1						
0000	0						

Fig.29

Nb satellites dans la trame GSV

Nombre de non "nuls"

du signal peut ne pas être insérée dans la chaîne de caractère. Pour un même nombre de satellites la taille de la trame et la position des données ne sera alors pas identique. Le tableau de la Fig.29 précise la combinatoire possible. On constate qu'en fonction du nombre de satellites dans la trame et des informations de niveau de réception du signal on devrait envisager trente cas possibles. C'est bien trop pour utiliser des vérifications avec des instructions de type **if**. Aussi, on va utiliser la technique simple qui consiste à puiser les informations par simple décalage d'un INDEX dans la chaîne de caractère de **Tampon_GSV** et ce tant qu'il reste des octets de données. Pour savoir s'il reste des informations après avoir exploré les données d'un satellite, on regarde si après le délimiteur ',' il y a un chiffre. La copie d'écran du **Moniteur de l'IDE** qui représente un démarrage du GPS avec **Expérience_14.ino** est trop grande pour s'intercaler facilement dans la mise-en page de ce texte. C'est la raison pour laquelle elle est préservée dans **Image 05.JPG** du sous-dossier <IMAGES>. En standard en **1** le logiciel se présente. Puis en **2** une attente de six à sept minutes va s'écouler avant que quatre satellites au moins soient en hauteur de capture. En **3**, **4**, **5** et **6** on commence à dérouler des séquences dont pour deux cas les signaux sont trop faibles et leur intensité pas affichée. En **7** c'est le cas particulier d'un signal trop faible situé en fin de trame. En **8** un cas banal où c'est le premier satellite dont le signal est trop faible. Cet exemple est typique d'un cas où il y a *quatre trames de type GSV*. En **9** les quatre satellites de la trame sont en configuration favorable. Enfin en **10** on rencontre le cas classique d'une dernière trame incomplète, avec ici un seul descriptif. On notera au passage que les satellites sont listés par ordre croissant de leur identificateur.

➤ **Experience_15 : Données du type RMC.**

Non encore abordées, elles comportent des informations importantes pour la navigation qui n'ont rien de secondaires. Outre les coordonnées sphériques du mobile, on trouve dans la trame de type **RMC** le cap géographique de ce dernier ainsi que sa vitesse sol. En prime, la déclinaison magnétique locale est précisée information primordiale si l'on navigue à l'aide d'un compas magnétique, ce dernier étant utilisé en aviation pour vérifier les informations de la centrale inertielle sur les "liners" ou recalculer le conservateur de cap à bord d'un petit aéronef.



Méfiez-vous, car petit à petit avec des démonstrateurs réputés amusants le Nulentout va vous faire bosser sur tous les types de trames du truc GPSmachin !

1
2
3
4
5
6
7
8
9
Fig.30

\$GPRMC100214.00,A,4427.32688,N,00131.31595,E,1.253,230125,,A

- **1** : Heure UTC du groupe de données.
- **2** : Alerte du logiciel de navigation. (*A = OK, V = warning (alerte)*)
- **3** : Latitude du mobile.
- **4** : Longitude du mobile.
- **5** : Vitesse sol en nœuds. (*Ici elle devrait être nulle !*)
- **6** : Cap géographique suivi par le mobile.
- **7** : Date du groupe de données.
- **8** : Déclinaison Magnétique.
- **9** : Non précisé sur Internet.

(Date, heure et positionnement sont déjà obtenue dans d'autres types de trames. il est donc inutile de les extraire d'un préambule de type RMC.)

Les données listées dans les trames de type **RMC** sont précisées dans l'encadré de la Fig.30 dont les informations sont glanées sur Internet. La ligne de cette Fig.30 est issue d'**Expérience_15.ino** qui se contente d'afficher le contenu du **Tampon_RMC[]** et n'effectue aucune extraction. Comme nous somme immobiles, on constate que la valeur de la vitesse affichée en **7** est aberrante. La valeur en **8** de la déclinaison magnétique n'est pas fournie, et les trois ',' au lieu de deux attendus nous permettent d'affirmer que le '**A**' en fin de chaîne est bien une donnée supplémentaire non précisée sur Internet. (*Je suppose que c'est identique au 5 des trames VTG.*)

➤ **Expérience_16 : Données du type VTG.**

Cette expérience partage avec celle de **P15** une sorte d'inutilité. En effet, elle ne concerne que le cap suivi par le mobile et sa vitesse sol. Du coup **Expérience_16.ino** ne fait que visualiser le contenu de **Tampon_RMC[]** et n'effectue aucune extraction. La petite salamandre ne s'est pas trompée. On aura passé en revue tous les types de trames. Ainsi pour générer une application autonome nous aurons tous les éléments pour sélectionner et extraire les données pertinentes.

1
2
3
4
5
Fig.31

\$GPVTG,T,,M,1.286,N,2.382,K,A

- **1** : Cap géographique en Degrés. (*T pour True track made good.*)
- **2** : Cap magnétique en Degrés.
- **3** : Vitesse sol en nœuds. (*Ici elle devrait être nulle !*)
- **4** : Vitesse sol en Kilomètres heure. (*Ici elle devrait être nulle !*)
- **5** : **A** > Mode Autonome. **D** > Mode Différentiel. **E** > Mode Estimé. **S** > Mode Simulé.
N > Donnée non valide.

NOTE = Un nœud est égal à un mille marin par heure, soit **1,852** Km/H. Nous avons bien le rapport **2.382 = 1.286 x 1.852**

➤ **Expérience_17 : Exercice de révision.**

Avant d'envisager le développement d'une application autonome, nous allons tester la possibilité d'extraire et d'afficher toutes les données et de vérifier que même si l'affichage traine un peu, le temps pour tout traiter sera suffisant. Le démonstrateur **Expérience_17.ino** constitue un rassemblement des techniques élaborées dans les manipulations précédentes. Il réalise une synthèse de toutes les données pertinentes et les présente en clair sans économiser les textes pour pénaliser au maximum la durée des traitements et des affichages. En outre, on liste l'intégralité des trames d'un groupe avant d'en extraire les données pertinentes pour charger au maximum le travail effectué par le microcontrôleur. Tant que le nombre de satellites en configurations favorables n'atteint pas quatre, le programme se contente d'afficher toutes les trames suivi du texte "**Attente de 4 satellites.**" Dès que les traitements sont possibles, le programme passe à des affichages du genre de celui de la Fig.32 qu'il me semble utile de commenter. Normalement, à la mise sous tension le logiciel se présente en **1** suivi en **2** du contenu des données du groupe mémorisées. En **3** à la mise sous tension on devrait avoir pendant cinq à six minutes l'affichage du texte "**Attente de 4 satellites.**", mais ici on a fait un RESET alors que le GPS est sous tension depuis un bon moment. C'est la raison pour laquelle le démonstrateur affiche directement les données qui sont cohérentes.

Experience_P17 du 24/01/2025

PILE - TAS = 759

①

Les trois trames qui contiennent des données pertinentes ont leur préambule encadré en vert. Les trames **GLL** et

GSA pourront être ignorées. Quand à **VTG**, il faut réaliser des essais en mouvement pour voir si l'on peut effectivement extraire des informations de vitesse sol et de cap. Ce ne sera envisageable qu'avec un appareil

autonome. Comme c'était le cas pour le croquis **P15** on ne

liste les caractéristiques satellitaires qu'en étalant la saisie et l'affichage sur plusieurs trames. Par exemple

```
----- Groupe suivant -----
GLL = 4427.32792,N,00131.31272,E,100243.00,A,A
RMC = 100244.00,A,4427.32828,N,00131.31276,E,1.070,,240125,,,A
VTG = ,T,,M,1.070,N,1.982,K,A
GGA = 100244.00,4427.32828,N,00131.31276,E,1,07,1.16,150.2,M,48.1,M,,
GSA = 32,2,15,19,23,10,,,,,2.09,1.16,1.74
GSV 3/3 = 3,3,11,24,80,044,40,25,24,237,18,32,11,318,18
```

⑥

②

④

⑤

③

***** Données du groupe *****

11H 02min 44S le 24 Janvier 2025

Latitude = 44°27'20"N Longitude = 001°31'19"E Altitude = 150m

Correction en hauteur de la géoïde = 48.1m

Positionnement : Point calé avec 07 satellites en poursuite et 11 visibles.

Dilution horizontale = 1.16 soit une précision Nominale.

Satellite 24 Hauteur 80 Azimuth 044 Signal 40

Satellite 25 Hauteur 24 Azimuth 237 Signal 18

Satellite 32 Hauteur 11 Azimuth 318 Signal 18

Fig.32

ici c'est la trame n°3 sur 3 qui est prise en compte. Comme c'est la dernière et qu'il y a 11 satellites visibles en 5, le dernier groupe ne contient que 11 - 8 = 3 satellites. En 6 le 1 signale que le point est calé et le 07 précise qu'il y a sept satellites en poursuite, c'est à dire donnant des informations fiables pour les calculs. Il ne faut donc pas confondre le nombre de satellites au-dessus de l'horizon local et le nombre de ceux qui sont aptes à fournir des informations fiables.

➤ Experience_18 : Optimisation du programme.

Tous les démonstrateurs qui précèdent avaient pour but de nous former à l'extraction des données d'un GPS sachant que j'ai abandonné l'idée d'utiliser une bibliothèque dédiée. L'inconvénient de cette approche c'est que nous avons été obligés d'apprendre le "langage des GPS" et de créer nos propres routines. L'avantage, c'est que l'on ne va utiliser que du code nécessaire. Il faut savoir qu'une application concrète autonome va devoir intégrer les routines d'affichages concernant le dispositif utilisé. Pour ce faire, il faut de la place dans le programme. Ensuite, on va désirer ajouter des fonctions. Pour que ce soit possible, il faut impérativement optimiser le code. Le but du démonstrateur **Expérience_18.ino** consiste à effectuer strictement le même traitement que celui de son prédécesseur, en optimisant la taille du programme et celle des données. Par ailleurs, afficher le contenu des trames est strictement inutile au point de vue opérationnel. Ce n'était justifié que pour nous aider à appréhender la norme NMEA 0183. Ce type d'information sera donc ignoré à partir de ce démonstrateur. La copie d'écran du **Moniteur de l'IDE** qui représente un fonctionnement stabilisé du GPS avec **P18** est trop grande pour s'intercaler facilement dans la mise-en page de ce texte. C'est la raison pour laquelle elle est préservée dans **Image 06.JPG** du sous-dossier **<IMAGES>** comme c'était déjà le cas pour **P14**. Dans la zone jaune, comme chaque fois le programme se présente. Puis succèdent les renseignements de base dans la zones bleues et en vert et rose les informations secondaires. Comme dans les manipulations précédentes, les caractéristiques satellitaires sont réparties sur plusieurs groupes, repérables dans les zones roses. **Il serait préférable de toutes les regrouper, quitte à las réunir dans des variables mémorisées**, mais ici il fallait rester dans des traitements similaires pour pouvoir comparer. On peut vérifier en tête de listage que le programme en version "légère" fait 688 octets de moins, les données dynamique se sont réduites de 236 octet et la place disponible entre la **PILE** et le **TAS** a augmenté de 236 octets. Ces bénéfices sont substantiels. Une autre façon de gagner de la place en code objet et en mémoire dynamique, c'est de "supprimer" les chaînes de caractères qui sont considérés comme des tableaux variables logés à la fois en zone programme et en données dynamiques. Il suffit de **loger les textes du dialogue Homme/Machine en mémoire non volatile EEPROM**. C'est évidemment ce qui sera fait pour les applications qui vont suivre. (À condition que l'EEPROM ne soit pas utilisée pour y loger des données qui doivent être conservée lors de la coupure d'énergie.) Nous allons enfin pouvoir passer à des applications et développer des systèmes totalement autonomes avec leur écran et leur clavier ...

04) Préparation de la première application autonome.

Autonome implique plusieurs contraintes par rapport à la solution élémentaire qui se résume à approvisionner une carte Arduino NANO ou UNO et un module GPS. C'est l'ordinateur qui fournit le clavier et l'écran de visualisation. Pour envisager un petit appareil portatif, il faut ajouter un dispositif de visualisation, un "clavier" pour dialoguer dans des menus et une source d'énergie :

- **Pour des raisons d'encombrement**, je m'impose une carte NANO qui tient infiniment moins de place et n'engendre, comme nous l'avons déjà vérifié, aucun problème de proximité. Par ailleurs, **sur la NANO on dispose de deux entrées analogiques de plus** détail qui n'est pas négligeable quand il faut répartir les ressources en Entrées/Sorties du microcontrôleur.
- Pour la source d'énergie une petite batterie de 9V de type 6F22 est parfaite. Mon choix se porte sur des technologie C,Li-ION rechargeable avec un bloc secteur USB, mais ce n'est pas impératif.
- Pour la visualisation, avec cette version on va écarter les affichages graphiques ce qui nous invite à sélectionner des modules peu onéreux de type LCD. Pour développer le programme je vais employer un modèle avec des luminophores actifs, mais pour des raisons d'autonomies de l'appareil je vous invite à choisir des modèles vraiment à cristaux liquides sans rétro éclairage.
- Pour le dialogue Homme/Machine, on va tenter dans cette application de n'utiliser qu'un seul codeur rotatif incrémental avec un bouton poussoir central **KY-040**.

➤ **Experience_19_Noyau : Le programme noyau.**

A ce stade rien ne nous interdit d'envisager plusieurs variantes qui ne diffèrent les unes des autres que par le type d'afficheur utilisé, présence ou non d'un codeur incrémental rotatif, utilisation d'un clavier à deux ou cinq touches etc. Il est évident que si on choisit un afficheur graphique les possibilités seront plus importantes que se limiter à un LCD avec deux lignes de texte. Toutes ces applications auront un ADN commun, et **on ne va pas passer son temps à l'extraire des croquis qui fonctionnent**. Trois domaines seront incontournables :

- La déclaration de la ligne SCI,
- La saisie et la mémorisation des groupes de données,
- L'attente de données cohérentes à la mise sous tension.
- Éventuellement la mesure de la tension de la batterie.

➤ **Affectations des Entrées/Sorties de l'ATmega 328.**

Bien que ce ne soit pas obligatoire, pour faciliter au maximum les manipulations lors de l'élaboration du programme, on va réserver **D0** et **D1** exclusivement au téléchargement du code objet par la ligne USB. L'entrée de mesure de la tension de la batterie se fait sur **A7** car c'est une broche qui ne peut servir que d'entrée analogique. Pour le codeur incrémental rotatif **KY-040** il faut impérativement utiliser **D2** et **D3** qui se servent des interruptions 0 et 1. Pour l'entrée de détection du bouton central on va prendre dans l'ordre **D4**. C'est possible car pour l'afficheur LCD avec sa bibliothèque LiquidCrystal.h on peut choisir à notre guise les broches de l'ATmega328. On va les

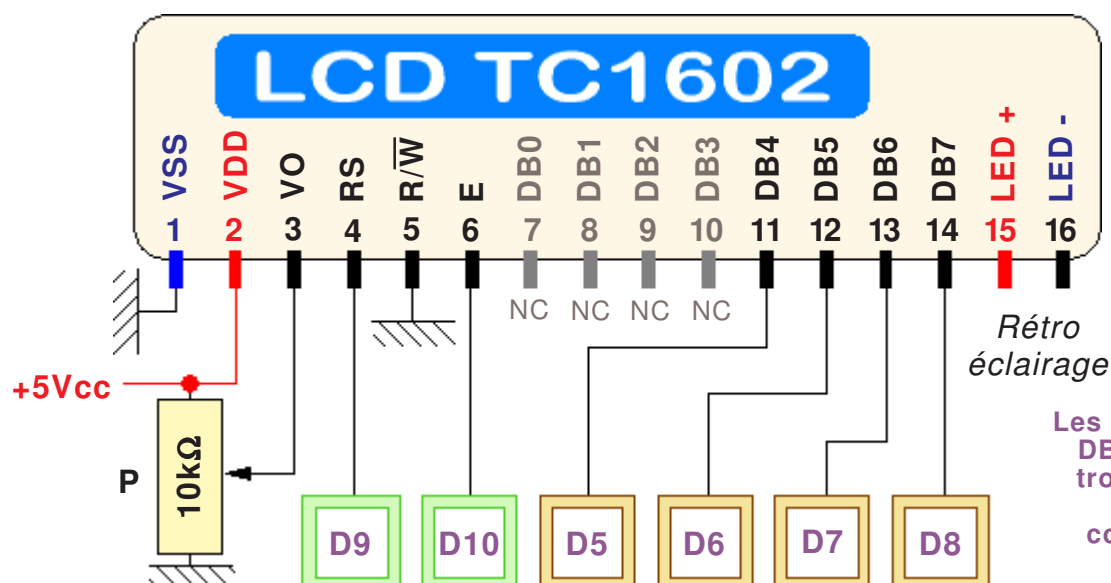


Fig.33

affecter dans l'ordre de **D5** à **D10**. Du coup, toujours dans l'ordre croissant on affecte la broche **D11** à la sortie **SCI** du module GPS. Il reste inutilisées les broches **D12** et **D13**, cette dernière pouvant éventuellement servir à piloter la LED d'Arduino. La Fig.33 détaille les branchements de l'afficheur LCD. Le potentiomètre **P** peut être un linéaire de **4,7KΩ**, **10KΩ**, **22KΩ**, sa valeur est sans importance. On peut même imposer le contraste maximal en réunissant la broche **VO** directement à **GND**.

➤ **Experience_20 : Implanter le LCD.**

Plutôt que de vous présenter directement le programme complet dont le listage sera long "comme un jour sans pain", il me semble plus judicieux de procéder par étapes. Ainsi chaque démonstrateur sera plus aisé à analyser dans ses spécificités. Par ailleurs, ainsi on valide le matériel au fur et à mesure des manipulations. Dans ce démonstrateur **Expérience_20.ino** on n'intègre matériellement et dans le croquis que l'afficheur LCD qui se contente d'afficher la **PAGE_1** c'est à dire la date et l'heure. Dès que l'activation de la **PAGE_1** en Fig.34 est possible la LED d'Arduino s'illumine le

Fig.34 temps du rafraichissement du petit écran.

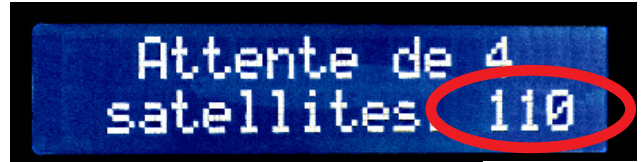


Fig.35

Durant l'attente des quatre satellites Fig.35 un compteur dans l'encadré rouge qui recycle à 255 s'incrémente pour nous informer du déroulement du programme. Penser à mettre à jour le booléen dans le **#define Hiver** situé en tête de listage pour que l'heure légale soit correcte. Pour comprendre certaines séquences du démonstrateur, je vous invite fortement à prendre connaissance des informations données en **Fiche n°3** et **Fiche n°4**. Les fiches au format A5 sont prévues pour être à portée de main quand on développe un programme en Arduino. Elles sont disponibles dans le fichier **FICHES A5.pdf** qui accompagne le didacticiel.

Programmation méthodique et structurée.

Ce didacticiel se veut plus qu'un simple recueil de petits (*Ou plus gros.*) programmes pour Arduino, mais propose un cheminement qui ambitionne de privilégier une approche *si possible méthodique et structurée* de la programmation. Il ne prétend surtout pas péremptoirement fournir l'exemple académique de ce qu'il faut faire pour bien programmer, mais désire ouvrir des pistes pour susciter encore plus l'envie de coder sur Arduino. Personnellement, au cours de mes années consacrées à la passion du C++ j'ai fini par dégager un certain nombre de critères conduisant à des programmes "propres". Je résume ces informations dans la **Fiche n°9** et la **Fiche n°10** et je m'impose en développement de les respecter au mieux. La **Fiche n°2** est précieuse pour optimiser le choix du type des variables.

La date indiquée pour #define Version.

Oups ... j'ai oublié de préciser un point important : *Dans tous les démonstrateurs, conformément aux conseils proposés dans la Fiche n°9 je précise la Date de sa dernière mise au point.* Il m'est arrivé lors de relectures épisodiques, de corriger parfois un petit détail sans modifier le Date. Donc ne pas s'en étonner, c'est peu fréquent mais possible.

La rigueur du langage PASCAL.

Oups BIS ... j'ajoute un deuxième point important : Dans le langage **PASCAL** que j'ai longtemps pratiqué et que je considère comme le plus élégant de tous ceux que j'ai expérimenté ce jour, conformément aux conseils proposés dans la **Fiche n°9** on doit placer toute procédure ou fonction avant l'instruction qui y fait appel ... y compris avant **void setup** et **void loop**. Hors, lors d'une expérience "négative", j'étais arrivé à la conclusion que le **C++** n'acceptait pas cet ordre d'écriture. Je viens de me rendre compte qu'il n'en est rien, j'avais du me fourvoyer. *Donc, dans les divers programmes on respectera ce conseil judicieux.*

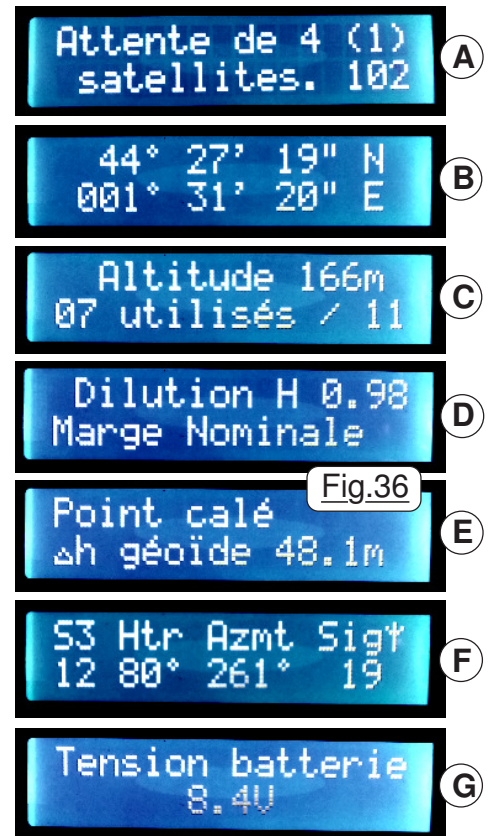
NOTE : Il me semble important de préciser que dans mes listages, les mots du langage C++ sont en marron, les identificateurs sont en vert et souvent les valeurs particulières sont en violet.

➤ **Experience_21 : Ajout du codeur incrémental KY_040.**

A partir du moment où l'on inclut dans l'appareil un potentiomètre, un codeur rotatif ou un clavier à deux, cinq touches ou plus, l'utilisation de l'ensemble "explose" en possibilités de fonctions opérationnelles. Le démonstrateur **Expérience_21.ino** apporte la preuve qu'il est possible d'exploiter avec un simple codeur rotatif, de façon conviviale, un appareil fournissant déjà pas mal de fonctions diverses. Les photographies de la Fig.36A à la Fig.36G présentent les différentes "pages-écran" disponibles avec **P21**. Inutile de présenter à nouveau l'écran de la Fig.34 qui n'a pas été modifié. Sur RESET c'est cette page qui sera affichée tant que le codeur incrémental rotatif ne sera pas actionné. Quand on le tourne dans le sens horaire, les pages défilent dans l'ordre des écrans représentés sur la Fig.36 et dans l'ordre inverses si le sens de rotation est en antihoraire. Par contre, la Fig.35 a muté en celle de la Fig.36A toujours avec le compteur en bas à droite. Entre parenthèses est précisé le nombre de satellites en poursuite, c'est à dire fournissant un signal fiable. Justification fondamentale d'un GPS, en Fig.36B ce sont les coordonnées sphériques du mobile qui sont affichées. En Fig.36C l'information de position est complétée par l'altitude au niveau moyen des océans. Sur la ligne du bas on précise le nombre de satellites en poursuite, et le nombre de ceux situés au-dessus de l'horizon local. En Fig.36D est précisé le facteur de dilution complété sur la ligne du bas par la fiabilité du positionnement. Un peu plus technique, l'écran de la Fig.36E indique le mode de détermination du positionnement ainsi que le décalage en hauteur du géoïde. Quand on active l'écran Fig.36F une LED bleue dédiée s'illumine. Elle signale que l'on est dans une fonction à plusieurs pages. Pour sortir de cette fonction il faut cliquer sur le bouton central du codeur incrémental. Ce mode présente les caractéristiques des satellites enregistrés dans le premier **Tampon_GSV** qui en contient quatre ordonnés de **S1** à **S4**. Sous ce numéro d'ordre est indiquée la référence du satellite en poursuite. Puis viennent les informations de l'élévation en **Hauteur** et d'**Azimut** et enfin la valeur de l'intensité de réception du **Signal**. Lorsque cette donnée n'est pas disponible la valeur est représentée par "***". (Collé à **Sig** le symbole d'une antenne.) Enfin en Fig.36G l'écran affiche la tension aux bornes de la batterie d'alimentation. Pour comprendre comment est exploité ce codeur incrémental **KY-040** la **Fiche n°6** en détaille les spécificités.

➤ **Caractères spéciaux sur un afficheur LCD.**

P articularité des afficheurs LCD qui utilisent pratiquement tous la même "racine électronique", leur police de caractère est limitée à celle de la **Fiche n°1**, mais nous offre la possibilité de créer jusqu'à huit caractères personnels sous forme d'une matrice de 8 x 5 PIXELs. Les instructions idoines sont précisées dans la **Fiche n°5**. En particulier vous pouvez observer que dans la police standard il n'y a pas les accentués ni les lettres avec tréma. On doit donc dans cette application se créer le 'é' et le 'ï'. Par ailleurs, je trouve que les lettres minuscules de la police standard avec jambage telles que le **g**, le **j**, le **p** et le **q** décalées vers le haut ne sont pas très esthétiques. C'est la raison pour laquelle, en plus du 'ö' et du symbole de la petite antenne, j'ai ajouté dans le logiciel la génération d'un 'g' personnel. Enfin, la lettre Delta pour Δh de la Fig.36E complète judicieusement notre collection matricielle. Cette information représente la correction de la hauteur du géoïde en mètres par rapport à l'ellipsoïde WGS84. Comme le terme "correction" contient bien trop de caractères, j'ai choisi de le remplacer par la lettre Δ qui traditionnellement en physique traduit un gradian ou la variation d'un paramètre dont on étudie le comportement. Je vous invite au passage de consulter la **Fiche n°14** qui précise la notion assez théorique de WGS84.



➤ **Experience_22 : Ajout de la détermination d'une loxodromie.**

Maintenant que le programme d'application n°1 est globalement élaboré, il importe de définir le schéma électronique complet et éventuellement d'ajouter d'autres fonctions qui rendront notre appareil encore plus séduisant. Dans l'état actuel, le code objet n'occupe que 40% de l'espace réservé au programme. Autant dire que nous avons encore beaucoup de place pour loger d'autres fonctions éventuelles. Encore faut-il trouver des idées pertinentes. Commençons par analyser le schéma électronique complet. Il est défini en [Fiche n°17](#). Pour comprendre en détails l'utilité de la résistance **R** consulter la [Fiche n°11](#) et la [Fiche n°12](#). Comme déjà précisé j'utilise une carte Arduino NANO pour des raisons d'encombrement dont les caractéristiques et le brochage sont

ATTENTION : Pas de VIN simultanément avec la liaison Mini-USB ou la carte électronique sera détruite.

précisés sur la la [Fiche n°7](#) et la [Fiche n°8](#). Il faudra impérativement placer l'inverseur sur **AR** lorsque la mini-fiche USB sera reliée à une prise de l'ordinateur pour téléverser une version modifiée du logiciel par exemple. (*Un programme est appelé à "vivre"*).

REMARQUE : La première idée qui vient à l'esprit pour vérifier les informations fournies par notre module GPS consiste à comparer la position géographique qu'il nous donne à celle que l'on peut aller voir sur Google Maps. On situe notre maison, puis notre "laboratoire". On effectue un zoom maximal. On pointe avec précision sur l'écran de l'ordinateur l'emplacement du module prototype. Et bien on constate une petite différence dans les secondes d'angle. Disposant d'un GPS commercial fiable, j'ai effectué la même manipulation. Constat identique. Se sont les deux GPS qui indiquent strictement la même position. *C'est donc la position donnée par Google Maps qui est en cause ... et c'est normal.* En effet, *le pointage sur l'écran de l'ordinateur est forcément approximatif car pixellisé à la fois par la carte fournie par Google Maps et par l'écran vidéo de l'ordinateur* d'où ces différences. Par contre, disposant de six modules pour Arduino, tous ont séjourné sur le montage expérimental et ont donné des résultats strictement identiques confirmant la fiabilité de ces petits modules du commerce.

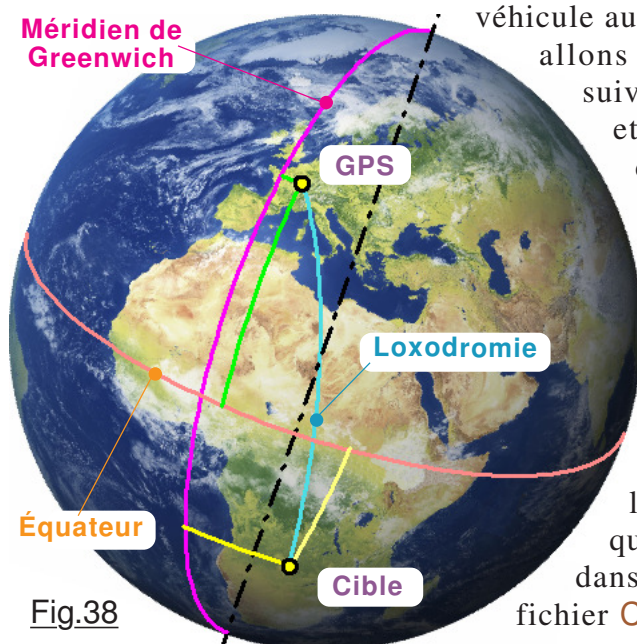
➤ **Affichage de la distance à "vol d'oiseau" d'un point CIBLE.**

Restons réalistes, ce n'est pas avec la puissance "dérisoire" d'un ATmega328 que l'on va pouvoir développer un GPS cartographique avec calcul d'itinéraire et autres fonctionnalités d'une puissance telle que l'on finit par croire que c'est élémentaire. Par contre, avec un appareil aussi rudimentaire, on peut définir un point géographique qui nous concerne, et le dispositif sera apte à nous indiquer en permanence la distance curviligne sur le géoïde qui nous en sépare. Si par exemple on se déplace en

véhicule automobile, nous allons constater que

suivant les impératifs routiers on s'en éloigne par moment, et puis plus notre route nous en approche, plus cette distance va diminuer pour finir par afficher zéro quand on aura atteint notre destination. Si ce n'est pas fabuleux, ce n'est déjà pas si mal pour un projet purement ludique. Pour obtenir le résultat montré sur la Fig.37 qui indique la distance entre PARIS et l'île de Paquès, les traitements mathématiques sont assez indigestes et nous obligent à faire un petit tour en *trigonométrie sphérique*. Aussi, pour ne pas encombrer inutilement le didacticiel sur la détermination d'une loxodromie représentée en Fig.38, pour celles et ceux qui désirent en savoir plus sur les traitements effectués dans **P22** je vous invite à consulter dans **<Documents>** le fichier **Cartographie et Loxodromie.pdf**.

Distance cible :
11942.73km Fig.37



➤ Google Maps pour évaluer une position géographique et une distance.

Que ce soit plus tard pour intégrer dans la petite machine des points d'intérêts ou pour engager une "campagne de tests" pour valider le logiciel avec une forte probabilité de vraisemblance, il nous faudra obtenir des coordonnées sphériques à la surface de la Terre et des distances mesurées sur cette dernière. Nous avons vu dans l'encadré page 19 que les valeurs obtenues seront approximatives. Peu importe, elles seront suffisamment précises pour satisfaire notre besoin. Le but de ce chapitre est de vous proposer une technique simple pour procéder à cette collecte d'informations.

Coordonnée géographique d'un point d'intérêt :

- 1) Activer **Google Maps**, (Voir la Fig.39)
- 2) En haut à gauche en **1** donner le nom de la Cible. Quand on valide le logiciel effectue un grand zoom et centre sa fenêtre sur la cible.
- 3) En bas à droite avec la puce **+** agrandir au maximum jusqu'à voir en détails la cible,
- 4) Avec le bouton **droit** de la souris cliquer exactement sur le point dont vous désirez les coordonnées géographiques,
- 5) S'ouvre juste en dessous du point visé une petite fenêtre contextuelle qui nous fournit les coordonnées géographiques de ce point.



Fig.40

Latitude	Pas de signe	N
Latitude	Signe négatif	S
Longitude	Pas de signe	E
Longitude	Signe négatif	W

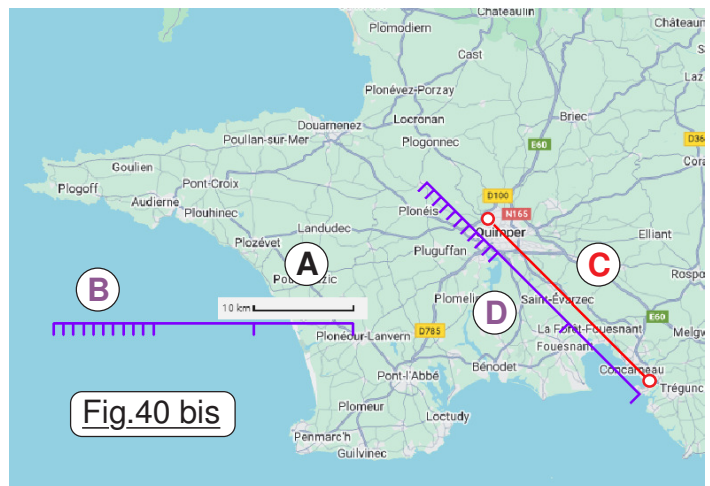
Le tableau de la Fig.40 indique les différents cas possibles et leur interprétation. Dans l'exemple de la Fig.39 nous avons :

- Latitude Cible = **45,85778N**
 - Longitude Cible = **1,23067E**
- Car les deux sont positifs.

Distance entre deux points : (Ce sera pour nous autre le GPS et un point d'intérêt.)

Pour exemple on suppose que vous habitez Concarneau. Tante Émilie habite Quimper. Vous désirez connaître à vol d'oiseau la distance qui sépare vos deux adresses :

- 1) Activer **Google Maps**, (Voir la Fig.40 bis)
- 2) Effectuer un agrandissement de la région.
- 3) Tracer en **C** un trait reliant les deux points.
- 4) En bas à droite en **A** est indiquée l'échelle.
- 5) Sur une bande de papier tracer une échelle mobile comme représenté en **B**.
- 6) Avec cette règle improvisée vous mesurez la distance en **D** soit environ 23km.



Comme précisé dans le document **Cartographie et Loxodromie.pdf** la distance mesurée par cette méthode sera un peu imprécise. Elle le sera d'autant plus que la surface représentée sur la carte sera importante puisque l'on construit à plat une réalité qui est sphérique. Toutefois, l'ordre de grandeur est suffisant pour vérifier le programme et nous avons en main tous les outils pour le faire.

➤ Vérification des calculs effectués dans Experience_22.

Pour pouvoir faire confiance à un logiciel quel qu'il soit, il importe d'effectuer une "campagne de tests" pertinente pour le valider avec une fiabilité raisonnable. L'une des étapes cruciales lors de telles vérification consiste à déterminer un jeu d'essais pertinent. Si vous avez consulté le document **Cartographie et Loxodromie.pdf** vous avez immédiatement compris que les traitements permettant de calculer la distance curviligne sur le géoïde qui sépare le GPS d'une cible imposent des essais établis avec attention. **Lors du développement, toutes les étapes intermédiaires ont été vérifiées pas à pas pour valider ligne à ligne chaque phase des calculs.** Dans le démonstrateur

Expérience_22.ino vous avez téléversé un programme supposé validé. Aussi, pour vous la "campagne de tests" va se résumer à quelques manipulations faciles de validation "globale".

```

79 char Latitude_cible[9] = "2718624S"; // Format DDmmmmmm. (Île de Paques) ①
80 char Longitude_cible[10] = "10943481W"; // Format DDDmmmmmm. (Île de Paques)
81
82 //char Latitude_cible[9] = "4074726N"; // Format DDmmmmmm. (New York) ②
83 //char Longitude_cible[10] = "07404785W"; // Format DDDmmmmmm. (New York)
84
85 //char Latitude_cible[9] = "4437051N"; // Format DDmmmmmm. (Clansaye) ③
86 //char Longitude_cible[10] = "00480745E"; // Format DDDmmmmmm. (Clansaye)
87
88 //char Latitude_cible[9] = "4737359N"; // Format DDmmmmmm. (DONZY) ④
89 //char Longitude_cible[10] = "00312634E"; // Format DDDmmmmmm. (DONZY)
90
91 //char Latitude_cible[9] = "4445450N"; // Format DDmmmmmm. (Prk Ludo Rolles) ⑤
92 //char Longitude_cible[10] = "00144350E"; // Format DDDmmmmmm. (Prk Ludo Rolles)
93
94 //char Latitude_cible[9] = "4445544N"; // Format DDmmmmmm. (ICI) ⑥
95 //char Longitude_cible[10] = "00152192E"; // Format DDDmmmmmm. (ICI)

```

Fig.41

Située vers le début du listage de **P22** on trouve la zone de la Fig.41 dans laquelle se trouvent des coordonnées géographiques "trouvées" dans **Google Maps** avec les techniques décrites dans le chapitre précédent. Dans les remarques telles que celle repérée dans l'encadré rouge on trouve le rappel du format des données fournies par **Google Maps**. Ces données ont été légèrement modifiées en supprimant le signe négatif éventuel et en remplaçant par **N**, **S**, **E** et **W**. En première utilisation de ce croquis la **Cible** active est l'île de Pâques car elle est éloignée au maximum de la France, située à l'ouest et dans l'hémisphère sud. Il est évident qu'en fonction de votre lieu d'habitation la valeur que vous prouverez sera différente. Il n'en reste pas moins vrai que les 13000Km déterminés par le démonstrateur correspondent assez bien à la distance que l'on évalue sur **Google Maps**.



Pour le deuxième test en **2**, on passe en remarques les deux lignes 79 et 80 et on valide la 82 et la 83. En choisissant comme **Cible** la ville de New York on sélectionne une destination lointaine située à l'Ouest et dans l'hémisphère Nord. Quand on téléverse la nouvelle mouture du programme on va alors obtenir une distance qui ressemble à 6100km. C'est bon signe, le logiciel retourne des ordres de grandeurs assez conformes aux prédictions mesurées avec **Google Maps**. Comme exemples suivants, j'ai choisie en **3** un charmant petit village dans la Drôme et une autre petite commune en **4** ancrée vers le centre de la France. Je ne vous en livre pas les valeurs de distance à trouver, laissant à votre sagacité le plaisir d'aller les déterminer par vous-même.

achever cette courte campagne d'essais se termine par deux tests particuliers et impératifs. Le premier en **5** consiste à appliquer la technique détaillées en Fig.40 c'est à dire sélectionner une **Cible** relativement *proche entre dix et vingt kilomètres de chez vous*. Ce sont les conditions où la mesure approximative effectuée sur l'écran vidéo sera la plus précise. Enfin, le test final en **6** consiste à donner comme coordonnées celles que vous trouvez dans **Google Maps** en pointant le plus finement possible l'emplacement de votre maison. En théorie Arduino devrait indiquer une distance nulle. Dans la pratique on aura une valeur comme **0.01km**. Dix mètres d'erreur avec un tout petit calculateur qui mouline des données issues de satellites situés à des centaines de kilomètres ... avouez que c'est presque de la magie ! Nos algorithmes étant validés, on va pouvoir passer à un premier programme d'application dans lequel on pourra facilement indiquer des points d'intérêt. L'idée de base consiste à pouvoir enregistrer les coordonnées de plusieurs points d'intérêts en mémoire EEPROM pour s'en servir à notre convenance. Comme le matériel est limité à un simple codeur incrémental, pour inscrire les coordonnées des **Cibles** en EEPROM on va utiliser pour le dialogue Homme/Machine le **Moniteur de l'IDE**. Par contre, une fois que ces données seront en EEPROM l'appareil sera autonome pour les utiliser. Il nous reste à en définir les protocoles ...

➤ Application_GPS_1.ino.

Cette application est la plus simple au niveau matériel puisqu'elle se résume à une carte Arduino NANO, Un module GPS du commerce, en afficheur de type LCD, un codeur incrémental rotatif et un accumulateur rechargeable de 9v. (Ou une pile de type 6F22 ou 6LR61.) La première étape pour élaborer ce petit projet consiste à optimiser l'usage des ressources de l'ATmega328.

Protocole de sélection à l'usage d'un point d'intérêt.

Compte tenu du matériel disponible, on va se servir du bouton poussoir central du codeur incrémental. On suppose naturellement que des adresses sont déjà disponibles en EEPROM. À la mise sous tension, si le bouton central du codeur rotatif est activé, l'écran va afficher une page du genre de celle représentée en Fig.43 qui indique le numéro d'ordre de la Cible ainsi que son libellé. En tournant le codeur on balaye les adresses disponibles. Quand on clique à nouveau sur le bouton poussoir central, on passe au fonctionnement du GPS qui calculera alors la longueur de la Loxodromie entre sa position actuelle et celle de la **Cible**.

(En réalité le protocole a été complété et l'on enchaîne avec le choix des options.)

Cible EEPROM 18:
Park Ludo Rolles

Fig.43

Organisation des données en mémoire non volatile EEPROM.

Pour les coordonnées du point d'intérêt il faut 17 octets. Pour le texte à l'écran il faut 16 octets. Pour une cible il faut 33 octets. On pourra donc loger $1024 / 33 = 31$ adresses au maximum. **On va se limiter à 30 points d'intérêt**, plus en accord avec des habitudes où les dizaines ont tendance à s'imposer. Surtout, avec 31 P.I. il ne resterait qu'un seul octet de disponible pour les informations non volatiles. Hors on va



Fig.44

Fig.45

Adresse	Contenu
1023	Nombre de P.I.
1022	P.I. sur RESET
1021	Heure d'hiver
1020	Miles Nautiques

vouloir mémoriser le nombre de P.I. disponibles, celui qui sera rechargé sur RESET, l'option Été/Hiver, les affichages en km ou en MN etc. L'organisation de l'EEPROM

montrée sur la Fig.44 consiste à placer les P.I. En partant de l'adresse 0000 et en "montant" dans l'ordre des cellules. Pour les données des variables initialisées lors du RESET on part de l'adresse la plus "haute" 1023 et on loge les valeurs vers le bas au fur et à mesure des besoins exprimés. (Voir le tableau de la Fig.45 établi à ce stade du développement du programme.)

Pour pouvoir facilement développer et vérifier le comportement du programme d'application, il importe d'avoir des données cohérentes en mémoire non volatile. Aussi, un petit outil nommé **Ecrire_en_EEPROM_1.ino** place en mémoire sept points d'intérêt, impose le n°6 au rechargement sur RESET, impose l'heure d'hiver ainsi que les unités de distance et de vitesse en Km et Km/H.

La Fig.45 bis présente le contenu de la mémoire l'EEPROM de l'ATmega328 lorsque **Ecrire_en_EEPROM_1.ino** a été téléversé **et exécuté au moins une fois**. Toute la zone qui contient des points d'intérêt est affichée en forma texte, car ce sont des éléments de type **char**. Chaque

point d'intérêt est mis en évidence par coloriage. Il est ainsi facile d'en déduire l'adresse de début et de fin pour effectuer les essais de validation. Les derniers octets sont de type **boolean** ou **byte** et sont présentés en Hexadécimal. Les \$FF correspondent à des "225", c'est à dire des cellules vierges qui n'ont encore jamais été écrites.

ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	4	4	4	5	5	4	4	N	0	0	1	5	2	1	9	2
0016	E	M	a	M	a	i	s	o	n	.						
0032		4	7	3	7	3	5	9	N	0	0	3	1	2	6	3
0048	4	E	D	o	n	z	y	.								
0064		4	4	3	7	0	5	1	N	0	0	4	8	0	7	
0080	4	5	E	C	l	a	n	s	a	y	e	s	.			
0096		4	3	8	3	7	6	3	N	0	0	4	3	8		
0112	3	7	8	E	N	I	M	E	S	.						
0128				4	8	8	5	3	6	5	N	0	0	2	3	
0144	5	0	2	6	E	P	A	R	I	S	(N	t	r		
0160	D	a	m	e)	9	0	0	0	0	0	0	N	0	0	0
0176	0	0	0	0	0	E	P	o	l	e		N	O	R	D	.
0192						9	0	0	0	0	0	0	0	3	9	0
0208	0	0	0	0	0	0	W	P	o	l	e		S	U	D	.
0224	S	U	D	.												
0992	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1008	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	01	06

Fig.45 bis

Le MENU qui s'ouvre sur RESET.



Lorsque l'appareil est mis sous tension, il recharge les données de base depuis l'EEPROM, c'est à dire le nombre de P.I. disponibles en EEPROM, celui qui sera rechargé automatiquement durant le RESET, si le GPS est utilisé en heure d'hiver ou d'été et si les distances et les vitesses sont en unités métriques ou en unités nautiques. Si durant le RESET le bouton central du codeur incrémental est cliqué, on ouvre le **Menu du RESET** à son relâcher. Le protocole de sélection des diverses options est résumé sur la Fig.46 qui précise les actions à effectuer sur le codeur rotatif.

L'écran 1 indique en **B** le nombre de P.I. disponibles et en **A** celui qui sera pris en compte dans les calculs. En **C** sur la ligne du bas est indiqué le libellé de ce P.I. En faisant tourner dans un sens ou dans l'autre le codeur incrémental, on fait "défiler" les divers points d'intérêt en permutation circulaire. Quand celui qui est désiré est affiché sur l'écran LCD, en cliquant sur le **BPC** on passe à la sélection de l'option suivante en 2 et 3 pour le choix de l'heure légale. Dans cette configuration la rotation du codeur fait alterner entre Été et Hiver. En cliquant sur le **BPC** on valide l'option et on enchaîne en 4 et 5 à la sélection des unités utilisées pour afficher les distances et le jour où elles seront disponibles si j'y parviens, les unités des vitesses. À ce stade cliquer une dernière fois sur le **BPC** fait quitter le **Menu du RESET** et débute l'exploitation avec le **Menu du GPS**.



Fig.46



Note : Le symbole  indique qu'il faut cliquer sur **BPC** le **B**outon **P**oussoir **C**entral du codeur incrémental. Le dessin  pour son compte précise qu'il faut tourner le dispositif rotatif.

Protocole du dialogue Homme/Machine avec le Moniteur de l'IDE.

ATTENTION : Lorsque l'appareil sera relié à une ligne USB de l'ordinateur, il sera impératif de couper l'alimentation locale de l'accumulateur 9v. Passons à l'analyse préalable : Avec un seul codeur rotatif comme périphérique pour assurer le dialogue Homme/Machine, proposer à l'appareil du texte est inenvisageable. Aussi, comme inscrire des P.I. dans la mémoire non volatile de l'ATmega328 reste une fonction marginale de l'utilisation du GPS, cette opération sera réalisée en reliant l'appareil à un quelconque ordinateur dont on utilisera le clavier très convivial.

Fig.47

Prompt
Commande :
Longitude :
Latitude :
Libellé :

Code	OPTIONS
?	Menu
L	Lister les P.I.
E	Effacer un P.I.
A	Ajouter un P.I.
D	Déplacer un P.I.
Sortie par ' Q '.	

Pour que l'usage de notre petit dispositif soit agréable, il importe de proposer un maximum de souplesse opérationnelle. Le **Moniteur de l'IDE** devra permettre d'effacer un P.I. d'en obtenir la liste et bien entendu d'en ajouter à convenance dans la limite des places disponibles. Dans la ligne de saisie du **Moniteur** on sera amené à choisir des actions et consigner des valeurs.

- Le prompt donné dans le tableau Fig.47 indique la nature de l'information attendue.

- Le **Menu du GPS** sera l'un des caractères de la Fig.48.

Pour faciliter le travail de l'utilisateur, la touche affectée au caractère de commande pourra librement être frappée en minuscule. Par exemple le ',' sera équivalent à '?'. Chaque commande ne contiendra qu'un seul caractère validé immédiatement. On entrera alors dans une phase de saisie d'une donnée et le **prompt** sera contextuel. (*Prompt : Caractère ou texte d'invitation à saisir une donnée.*)

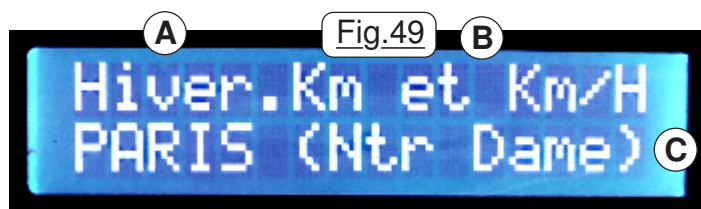
NOTE : Dans **Application_GPS_1.ino** la gestion des interruptions pour gérer le codeur incrémental a été entièrement revue par rapport à **P22** qui était à la fois trop compliquée et utilisait **delay()** qui ne fonctionnait pas lorsque l'interruption est active.

05) Première application autonome.

A ce stade du développement, on peut envisager la version d'**Application_GPS_1.ino** opérationnelle directement issue du démonstrateur **P22** auquel on a ajouté la fonction indispensable qui permet à l'aide du **Moniteur de l'IDE** de gérer les points d'intérêts sauvegardés en mémoire non volatile EEPROM. La mise en œuvre du dialogue Homme/Machine constitue la partie la plus technique ajoutée au programme et a été accomplie en plusieurs étapes.

➤ Réorganisation du Menu du GPS.

Dans le démonstrateur **Expérience_22.ino** les fonctions ont été intégrées dans l'ordre de leur développement. Les premières étaient naturellement les plus importantes. Puis ont suivi les "fonctions secondaires". Ensuite, le **Menu du RESET** ayant été ajouté au logiciel, il est apparu comme indispensable de compléter dans le **Menu du GPS** avec la fenêtre de la Fig.49 qui résume les données de base chargée depuis l'EEPROM lors d'un RESET. En **A** est précisée



Fonction

Date et l'heure.

Position.

Distance à la cible.

Affiche les options.

Tension batterie.

Altitude, Nb Tampon_GSV.

Dilution et fiabilité.

Type de positionnement.

Fig.50 Satellites.

si c'est l'heure d'été ou l'heure d'hiver qui est affichée. En **B** les unités de distance et de vitesse. Enfin en **C** sur la ligne du bas, la nature du point d'intérêt dont la loxodromie sera calculée. L'ordre des informations de la liste de toutes les fonctions du **Menu du GPS** a été repensé en plaçant dans l'ordre les fonctions les plus importantes au début et celles secondaires vers la fin. Le tableau de la Fig.50 résume cette nouvelle organisation. Par exemple l'heure ou la position et la distance à la cible sont bien plus opérationnelles que le type de positionnement, la dilution ou les caractéristiques de quatre satellites parmi "beaucoup". De même que les **valeurs des options** sont vitales pour que l'on sache quelle est la **Cible** sans avoir à invoquer le **Menu du RESET**.

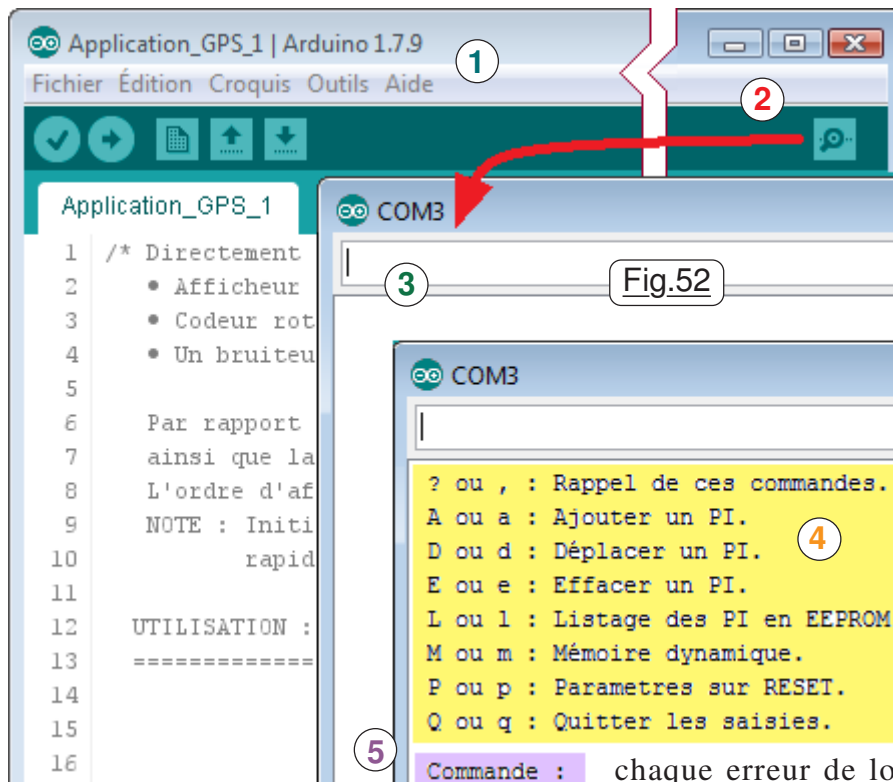
➤ Modification du Menu du RESET.

Après plusieurs expérimentations pour trouver un comportement agréable du programme, le protocole le plus convivial pour ajouter le dialogue Homme/Machine par l'entremise du **Moniteur de l'IDE** consiste à placer l'option en tête du **Menu du RESET**. L'approche explicitée en Fig.46 de la page 23 devient celle de la Fig.52 qui reprend le concept de rotation du codeur incrémental pour changer la valeur de l'item et d'un clic sur son bouton central pour valider l'option.

On commence par activer un croquis quelconque pour invoquer en **1** l'éditeur de texte de l'**IDE**. Puis, on clique sur le bouton central du codeur incrémental, et en le maintenant enfoncé, on clique sur la puce **2** qui ouvre la fenêtre contextuelle du **Moniteur** en **3**. Attendre que la



LED d'Arduino s'illumine pour libérer le bouton central du codeur rotatif et ouvrir sur l'afficheur LCD l'écran de la Fig.51 avec l'option **NON** par défaut. En tournant le codeur incrémental on fait alterner **OUI** et **NON**. Si on valide avec le bouton central étant en option **NON** on enchaîne sur la séquence de la Fig.46 en page 23. Au contraire, si la validation se fait avec **OUI** affiché, on enchaîne alors sur le **Menu du dialogue Homme/Machine**. S'affiche alors en **4** dans la fenêtre



contextuelle du **Moniteur** le rappel des commandes et en **5** le programme patiente pour une directive de votre part. La LED bleue va alors clignoter tant que l'on restera dans ce mode. Dans l'attente d'une **Commande** on doit fournir une lettre minuscule ou une lettre majuscule suivie immédiatement de la validation. Quand on tapera une commande, elle sera immédiatement exécutée. Si c'est l'ordre '**Q**' qui est imposé, le texte "**Fin du dialogue USB.**" s'affiche, un BIP sonore prévient l'opérateur, la LED bleue s'éteint et le logiciel passe directement au **MENU du GPS**. Un analyseur syntaxique vérifie toutes les sollicitations de l'opérateur et

adapté et d'une alerte sonore. Cette dernière impose le petit ajout de la **Fiche n°18** par rapport au schéma de la **Fiche n°17**. Comme l'ajout du bruiteur n'est pas impératif, je n'ai pas corrigé le contenu de la fiche 18, libre à vous de l'installer ou non en fonction de sa disponibilité.


➤ Tester le dialogue Homme/Machine.

Une petite prise en main me semble la bienvenue pour explorer les multiples facettes de la gestion des points d'intérêts. À ce stade de vos expérimentations vous avez déjà utilisé l'outil **Ecrire_en_EEPROM_1.ino** et téléversé **Application_GPS_1.ino**. Quand on compile le croquis, s'affiche en orange l'alerte de la Fig.53 qui sera commentée plus avant. Ne pas s'en préoccuper.

Le croquis utilise 21 576 octets (70%) de l'espace de stockage de programmes. Le maximum est de 30 720 octets. Les variables globales utilisent 1 654 octets (80%) de mémoire dynamique, ce qui laisse 394 octets pour les variables locales. Le maximum est de 2 048 octets.
La mémoire disponible faible, des problèmes de stabilité pourraient survenir.

Fig.53

Manipulations :

- 01) Ouvrir le **Moniteur** de l'**IDE** et imposer la vitesse de 19200bauds. (*Le 57600baud à été réduit à 19200baud car il était trop rapide et la fonction 'A' ne fonctionnait pas du tout correctement.*)
- 02) Maintenir enfoncé le bouton central du codeur incrémental. (**BPC.**)
- 03) Cliquer sur la puce  pour ouvrir à nouveau le **Moniteur** de l'**IDE**.
- 04) Attendre que la LED d'Arduino s'illumine et relâcher le **BPC**.
- 05) Tourner le codeur incrémental pour obtenir **OUI** sur l'afficheur LCD.
- 06) Cliquer sur le **BPC**, la liste des **Commandes** possibles s'affiche dans la fenêtre contextuelle.
- 07) Proposer '**x**' dans la fenêtre de saisie et valider. L'analyseur syntaxique ne reconnaît pas cette lettre comme étant valide et vous le signifie par un texte associé et un BIP pour attirer l'attention.
- 08) Valider à vide. Le programme affiche "**Longueur 0 ou > 1 !**" pour nous signifier qu'un texte de longueur nulle ou supérieure à un ne sera pas accepté. (*La loi c'est la loi !*)
- 09) Tenter le texte "**123456789abcdefghijklmnopqrstuvwxyz**". (*C'est pas la peine d'insister !*)
Observez au passage que le logiciel affiche entre crochet le texte qu'il a enregistré. Vous pouvez constater que sa longueur est limitée à 16 caractères.

REMARQUE : Pour ne pas avoir à frapper tous les caractères de l'exercice, il suffisait de faire un Copier dans le logiciel de présentation du didacticiel et un coller dans la ligne de saisie du **Moniteur**. La réciproque est vraie. On peut effectuer un Copier dans la fenêtre contextuelle du **Moniteur** suivi d'un coller dans n'importe quel traitement de texte.

Manipulations : (Suite)

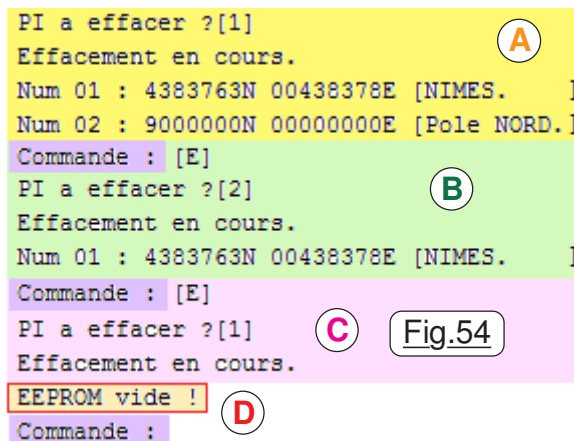
- 10) Proposer 'I' ou 'L' dans la fenêtre de saisie et valider. Lettre minuscules ou majuscules sont équivalentes. Les minuscules sont plus faciles à frapper.

La commande sera systématiquement transformée en majuscule. Ainsi on ne pourra pas confondre le 'I' avec le '1' par exemple. Par contre, quand on indiquera le libellé d'une Cible la "casse" sera respectée. De plus il ne faudra pas utiliser des accentués non acceptés par l'afficheur LCD.

- 11) À tout moment la **Commande** ',' ou son équivalent majuscule '?' listera un rappel des commandes possibles. Tester ',' puis '?'. Quand on a oublié les lettres valides, "on se pose une question" et le caractère qui vient à l'esprit est le point d'interrogation. Toutefois la virgule nous évite d'avoir à solliciter la touche majuscule.
- 12) On va tester le mode silencieux. Frapper 's'. Le programme affiche "BIPs NON".
- 13) Avec 'n' ou toute autre lettre non valide tester l'erreur. "Commande incorrecte." est affiché dans le plus grand silence. Ce mode est bien utile quand on ne veut pas déranger un entourage éventuel.
- 14) Indiquer une deuxième fois 's'. Le "BIPs OUI" confirme que le mutisme n'est plus d'actualité.
- 15) Enchaîner les deux commandes 'L' suivie de 'e'. Le programme affiche "PI a effacer ?".
- 16) Commencer par effacer la dernière, la n°7. Le programme affiche "Effacement en cours" et attire notre attention avec un BIP sonore. Avant de réitérer le prompt, il s'écoule plusieurs secondes. En effet, effacer un PI qui se trouve vers le début engendre le décalage de l'intégralité des emplacements qui suivent, qu'ils soient occupés ou non. Le listage de l'EEPROM qui suit l'effacement du PI confirme bien que la Cible qui était en emplacement n°7 n'est plus présente en mémoire non volatile et qu'il n'en reste plus que six.

On peut à convenance effacer plusieurs PI et il peut en rester moins que l'ordre de celui qui était à recharger par défaut sur un RESET. Pour palier ce risque, *dès que l'on efface un PI, celui qui sera rechargé par défaut sera le n°1.*

- 17) Frapper 'q'. Bip sonore, texte "Fin du dialogue USB." et présentation de la date et de l'heure sur l'afficheur LCD. La LED bleue est éteinte.
- 18) Tourner le codeur incrémental de trois pas dans le sens horaire pour faire afficher les paramètres des options. C'est bien la cible "Ma Maison." qui est rechargée sur un RESET. À partir d'ici, c'est donc le premier PI qui sera rechargé par défaut.
- 19) Pour confirmer que c'est bien l'emplacement n°1 qui est placé en Cible par défaut provoquez un RESET avec le petit bouton de la carte NANO et reprendre les manipulations de (18).
- 20) Recommencer les actions pour à nouveau revenir en **Menu du dialogue Homme/Machine**. Si vous avez encore une difficulté, reprendre en (02).
- 21) Commandes 'L' suivie de la consigne 'e'. On a vu qu'effacer la dernière fonctionne correctement. Cette fois on va désigner la première. *Scongregneugneu ! le coup d'avant c'est la Terre qui a perdu son pôle SUD. Franchement je supporte. Mais cette fois c'est "Ma Maison" qui n'existe plus !!!*
- 22) Le programme a confirmé qu'il efface correctement les PI des deux extrémités. Pour vérifier qu'il est aussi fiable avec les intermédiaires effacer le PI n°2 et le PI n°3 qui à ce stade sera celui de PARIS. OK, le logiciel fonctionne comme on le désire. On va tester l'analyseur syntaxique.
- 23) Commandes 'e' et '0' comme cible. (Zéro et pas O majuscule.)



PI a effacer ?[1] **A**
Effacement en cours.
Num 01 : 4383763N 00438378E [NIMES.]
Num 02 : 9000000N 00000000E [Pole NORD.]
Commande : [E]
PI a effacer ?[2] **B**
Effacement en cours.
Num 01 : 4383763N 00438378E [NIMES.]
Commande : [E]
PI a effacer ?[1] **C** **Fig.54**
Effacement en cours.
EEPROM vide ! **D**
Commande :

- 24) Tenter 'e' puis '-3' comme cible et Tenter 'e' suivi de '9'. Il ne sera pas possible d'effacer des PI qui n'existent pas.



Ben Môa môa quand je vois qu'il n'y a plus les pôles, ni PARIS, ni NIMES, ni Clansayes, ni Donzy, ni Ma maison. Tout à disparu. Il ne reste plus rien. Cette Commande 'e' c'est une arme de destruction massive !

- 25) Effacer maintenant les trois derniers P.I. Les manipulations sont montrées sur la Fig.54 qui en **A** ne contient plus que deux PI. L'effacement en **B** ne laisse plus qu'une adresse. Enfin en **C** on élimine la dernière position, le logiciel attire notre attention avec un BIP sonore et en **D** précise que maintenant l'EEPROM est vide.
- 26) Tenter d'effacer un PI avec 'e'. BIP et "EEPROM vide !"
- 27) Tester 'L' : Même sanction. (*On ne peut plus lister ou effacer des PI.*)

Il est naturel de se poser la question :

- **Quelle sera la cible rechargée par défaut sur RESET ?**

La réponse est facile : Tester par vous-même :


- 28) Provoquer un RESET : Un BIP d'alerte nous prévient que quelque chose n'est pas correct.

ATTENTION : Si sur RESET le programme génère un BIP sonore, c'est qu'il n'y a plus de PI disponible en EEPROM. Celui rechargé par défaut est aléatoire. Si on tente d'utiliser le **Menu du RESET** pour choisir un PI au rechargement, le comportement sera incohérent.

CONCLUSION : Toujours disposer au moins d'un PI en EEPROM. Donc ne jamais les effacer tous lors du **Menu du dialogue Homme/Machine** sur USB. (*Ou en recréer un.*)

➤ Dialogue Homme/Machine : Ajout de PI.

Ajouter à convenance un ou des PI est indispensable, car il n'est pas question de faire appel à un programme tel que `Ecrire_en_EEPROM_1.ino` chaque fois qu'une nouvelle cible devient pertinente dans notre usage du petit appareil. On se doute que la Commande '**A**' doit obéir à certaines règles de bonne conduite :

- La **Latitude** sera de type DDmmmmmm**N** ou **S**,  **D** pour les degrés. } Obligatoirement des chiffres.
- La **Longitude** sera de type DDmmmmmm**E** ou **W**, **m** pour les minutes. }
- Le libellé pourra contenir 16 caractères quelconques au maximum. *Éviter toutefois les caractères qui ne seront pas listés correctement par l'afficheur LCD* comme les accentués par exemple. Si on indique moins de 16 caractères, la donnée sera complétée par des espaces.

Manipulations : (Suite)

- 29) Effectuer un redémarrage pour vous retrouver en saisie USB.

30) Commande '**a**'. **Latitude ? 4074726N.**

31) **Longitude ? 07404785W.**

32) **Nom de la cible ? New York.**

33) Consigne '**a**'. **Latitude ? 2718624S.**

34) **Longitude ? 10943481W.**

35) **Nom de la cible ? Ile de Paques.**

36) Tester '**a**'. **Latitude ? 4737359N.**

37) **Longitude ? 00312634E.**

38) **Nom de la cible ? DONZY.**

Penser à procéder par Copier / Coller.

- 39) Maintenant que les formats vous sont familiers, vous pouvez vous amuser à ajouter les coordonnées de votre domicile, quitter, imposer de recharger ce dernier sur RESET et vérifier que pour l'indication de la distance (*Loxodromie*) le GPS indique bien zéro. C'est pas mal de manipulations à enchaîner, aussi prenez votre temps, ce sera un bon exercice de révisions.

Manipulations : (Suite)

- 40) On va tester l'analyseur syntaxique. Tenter '**a**'. **Latitude ? 1234567n.** Pas de problème car le '**n**' a été changé en '**N**' durant la saisie.

41) **Longitude ? 12345678S** : BIP et **Valeur incorrecte !** car **S** n'est pas valide en longitude.

42) Tester '**a**'. **Latitude ? 123456n** : BIP et **Valeur incorrecte !** car il manque un chiffre.

43) Continuer '**a**'. **Latitude ? 12345678n** : BIP et **Valeur incorrecte !** car cette fois il y a trop de chiffres, du coup le '**N**' n'est pas au bon endroit et l'analyser syntaxique se fâche.

44) Proposer '**a**'. **Latitude ? hhhhhhhn** : **Pas de BIP et la valeur est acceptée !**

CONCLUSION : L'analyseur syntaxique vérifie la longueur de la coordonnée, les indicateurs '**N**', '**S**', '**E**' et '**W**'. *Il ne vérifie pas la cohérence de la valeur numérique.*

➤ Dialogue Homme/Machine : Saturation de l'EEPROM.

L'analyseur syntaxique vérifie la longueur des données de position et leur cohérence. Nous avons vu qu'il ne vérifie pas les caractères qui logiquement devraient être des chiffres. Il vérifie encore moins la vraisemblance des positions indiquées. Par exemple il acceptera sans siller une latitude de 99 degrés ce qui naturellement est totalement illogique de même qu'une longitude de 999 degrés. Effectuer des vérifications sur les chiffres imposerait un gros travail logiciel et la place pour loger le code n'est plus disponible, on va y revenir. L'expérience acquise lors des innombrables vérifications effectuées en cours de développement m'a montré que ce n'est pas en donnant les informations de position que l'on se trompe, d'autant plus qu'elles sont faciles à relire. C'est principalement lors de leur détermination sur [Google Maps](#) que le risque de se tromper est significatif. Il reste à vérifier le comportement du programme si on tente d'ajouter un PI et que les trente postes sont déjà utilisés. *(Ce qui est peu probable ou alors c'est que vous voyagez beaucoup !)*

Manipulations : (Suite)

- 45) Franchement, se "cogner" trente adresses est tellement indigeste que j'ai de loin préféré improviser un petit logiciel qui sature l'EEPROM à ma place, quitte à avoir des doublons, des triplons et des trucplons. Téléverser l'outil [Emplir_EEPROM.ino](#) dont l'identificateur pas très "parlant" est choisi pour qu'il soit placé en tête dans l'explorateur de Windows.
- 46) Téléverser immédiatement [Application_GPS_1.ino](#) pour pouvoir manipuler.
- 47) RESET et **BPC** pour revenir dans le [Menu du dialogue Homme/Machine](#) sur USB.
- 48) Commende '**L**' : On dispose bien de 30 PI avec certains en triple ou en quadruple. Ils sont tous cohérents et rien n'interdit de les collectionner. *(Pas très logique mais toléré.)*
- 49) Provoquer un RESET et cette fois refuser le dialogue USB. Tourner le bouton du codeur incrémental dans les deux sens. Il est effectivement possible de charger n'importe lequel d'entres eux au moment d'un RESET. On peut aussi faire tourner précipitamment le codeur pour effectuer rapidement des sauts dans un sens ou dans l'autre.
- 50) Revenir dans le [Menu du dialogue Homme/Machine](#) sur USB.
- 51) Tenter la Commande '**a**' : BIP et [EEPROM pleine !](#) Il n'est plus possible d'ajouter un PI, il faudra impérativement libérer un ou des emplacements avec '**e**'.
- 52) À titre d'exercice repérer le contenu du n°16 par exemple et l'effacer. C'est un peu long.
- 53) Continuer en effaçant le premier. C'est encore bien plus long car cette fois c'est toute l'EEPROM qu'il faut traduire.
- 54) Insister en effaçant la dernière. Cette fois c'est très rapide car on ne déplace que 33 octets.

➤ Dialogue Homme/Machine : OUPS ... j'ai oublié la commande 'D'.

Emporté par l'expérimentation d'Effacer, Ajouter, Lister et observer le comportement de l'analyseur syntaxique, j'allais oublier la Commande '**D**' qui pourtant est bien indiquée lors de l'affichage par la consigne '**?**'. Cette fonction permet de permuter deux PI pour en changer l'ordre dans la mémoire EEPROM. On peut inverser les positions dans les deux sens.

Manipulations : (Suite)

- 55) Commençons d'échanger le premier et le dernier : '**d**' puis '**27**' et enfin '**1**' ... ça fonctionne.
- 56) Dans l'autre sens et des PI du "centre" : '**d**' puis '**5**' et enfin '**22**' ... ça roule également !
- 57) Tester '**d**' puis '**0**' ... BIP et [Nombre incorrect](#) comme punition.
- 58) Tenter '**d**' puis '**3**' et enfin '**29**' ... même punition.

L'analyseur syntaxique interdit toute tentative d'inverser deux emplacements si l'un d'eux n'existe pas. Reste à tester les incongruités logiques :

- 59) Armez-vous de courage et effacer tous les P.I. *sauf le premier*. Pour minimiser les temps d'effacement indiquer chaque fois le dernier PI.
- 60) Quand il ne reste plus qu'un seul PI, soumettre '**d**' puis '**1**' et enfin '**1**' ... BIP et le texte d'erreur [Echange 1 avec 1 ???](#) qui montre qu'un échange "doublon" sera refusé.
- 61) Effacer le dernier PI. Puis '**d**' qui nous gratifie de BIP et [EEPROM vide !](#)

L'analyseur syntaxique ne se laisse pas faire. La fonction de permutation entre deux PI présente un comportement sain. Commande '**a**' pour ne pas laisser l'EEPROM vide ...

On vient de vérifier que même si l'EEPROM est très encombrée, gérer les PI reste parfaitement cohérent. Pratiquement toutes les fonctions sont surveillées par l'analyseur syntaxique. On peut considérer que le programme est fiable et validé. Il reste encore à aborder le problème de la collision de la **PILE** avec le **TAS**.

➤ Collision de PILE !

Grâce à cette expérience qui va changer votre vie, (*C'est une boutade !*) vous allez non pas assister à une collision entre deux véhicules automobiles, de deux aéronefs, de deux particules dans un cyclotron apériodique, de deux discussions qui dégénèrent, mais à une **Collision de PILE !** On peut affirmer sans risquer de se tromper, que si à un moment donné votre programme se met à "diverger" alors que l'on a ajouté une "bricole" qui ne devrait pas le perturber, c'est qu'il y a une forte probabilité de collision entre la **PILE** avec le **TAS**. Pour rencontrer ce phénomène le mieux est de l'expérimenter :

Manipulations : (Suite)

- 62) Téléverser le démonstrateur **Experience_23.ino** copie conforme de **Application_GPS_1.ino** qui à ce stade ne comporte que des remarques en plus.
- 63) Pour comprendre en détails ce que signifie **Collision de PILE** commencer par consulter dans le dossier <Documents> le fichier **Collision_entre_la_PILE_et_le_TAS.pdf**, on peut maintenant passer à la suite : Activer le **Menu du dialogue Homme/Machine** sur USB.
- 64) Proposer la commande '**m**'. Le programme précise sa version. Et surtout il nous indique que la place qui reste dans la mémoire dynamique fait 298 octets. Compte tenu des campagnes de test effectuées on a validé sérieusement le programme et l'on peut considérer sérieusement qu'il est stable.

NOTE importante : le compilateur du C++ d'Arduino surveille un peu l'occupation des espaces mémoire de l'ATmega 328. Quand il commence à avoir un doute, il nous prévient avec le message d'alerte de la Fig.53 en fin de compilation. Une longue expérience de programmation en langage machine m'a amené à la conclusion que 300 octets de mémoire dynamique sont largement suffisants pour faire fonctionner un microcontrôleur sauf cas particulier. Du reste, si vous téléchargez d'autres publications de ma part sur Internet, vous constaterez que j'ai déjà mis en ligne des programmes qui utilisent 100% de l'espace dédié au programme et 98% de celui qu'il nomme "mémoire dynamique". Ces logiciels fonctionnent parfaitement. Donc je crois pouvoir affirmer que le compilateur d'Arduino est "frileux" et qu'il anticipe un peu exagérément. Du reste, ce message d'alerte apparaît bien avant l'état actuel du programme, et j'ai continué à ajouter des fonctions sans vergogne. Il se trouve, qu'actuellement on commence à approcher de la limite. Ajoutons encore un peu trop de code, et le compilateur va avoir raison.

- 65) Modifier le programme pour ajouter une fonction qui aurait été bien utile : La commande '**G**' qui permet d'effacer d'un coup tous les PI sauf le premier :

- Transformer la ligne 545 et la ligne 553 en remarques.
- Valider la ligne 445 ainsi que la ligne 565.

La taille du programme passe de 21592 à 21806 octets et surtout l'espace occupé en mémoire dynamique monte à 84%. On va voir que maintenant le programme "diverge".

- 66) Téléverser une deuxième fois **Experience_23.ino**.
- 67) Activer le **Menu du dialogue Homme/Machine**. On observe en Fig.55 **A** la présence de la nouvelle fonction.
- 68) Tester '**L**', '**?**' ... tout semble normal.
- 69) Proposer '**m**', le logiciel se présente et annonce en **B** que l'espace en mémoire dynamique ne fait plus que 193 octets. Avec certains programmes ce serait suffisant mais pas ici car le code empile pas mal de données temporaires.
- 70) Lorsque la **Commande** est en attente, frappez un texte de longueur exagérée tel que trente fois '**a**' par exemple. L'accusé de réception en **C** est correct, mais seuls cinq caractères ont été retenus alors qu'il devrait y en avoir seize. Ce n'est pas normal et l'ajout de **A** ne peut pas avoir cet effet.

Fig.55

```
? ou , : Rappel de ces commandes.
A ou a : Ajouter un PI.
D ou d : Déplacer un PI.
E ou e : Effacer un PI.
G ou g : Gommer tous les PI - 1.
L ou l : Listage des PI en EEPROM.
M ou m : Mémoire dynamique.
Q ou q : Quitter les saisies.
S ou s : Silencieux.
Commande : [M]
Version du 11/02/2025
PILE - TAS = 193

Commande : [AAAAA]
Longueur 0 ou > 1 !
Commande : [A]
Latitude ? [N]
Valeur incorrecte !
```

71) Essayer en **D** d'ajouter un PI. Impossible, le programme ne récupère plus tous les caractères frappés dans la ligne de saisie. Hors nous n'avons fait qu'ajouter une fonction de plus qui en aucun cas ne devrait avoir d'effet sur celles qui fonctionnaient parfaitement avant. On ne peut soupçonner alors qu'un effet de collision de PILE.

Conclusion : La conséquence prédite par le message d'alerte du compilateur n'est pas systématique. Toutefois, quand cette ligne orange est affichée, il vaut mieux faire "le test de collision de PILE" et surtout *si brusquement le logiciel ne se comporte plus "comme avant", soupçonner une collision de la PILE avec le TAS.*

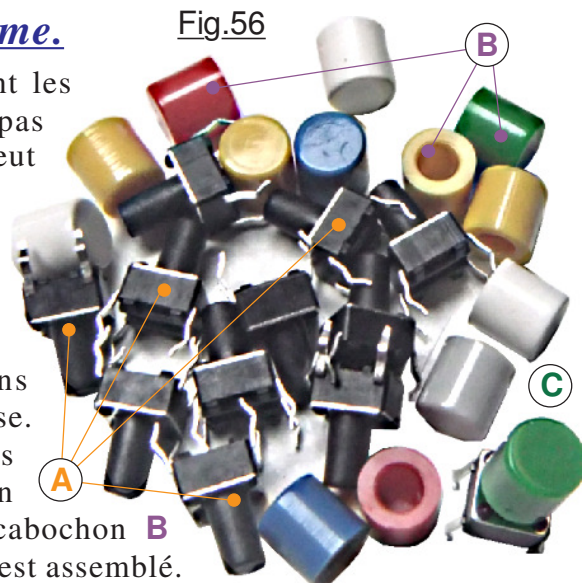
Remède : Libérer de la place en occupation de l'espace réservé aux données temporaires et locales. Ce n'est pas forcément évident, *mais l'un des moyens les plus efficaces pour y arriver consiste à placer les textes du dialogue H/M en mémoire non volatile EEPROM.* (Voir l'application n°2.)

Considérons que le programme **Application_GPS_1.ino** est aboutit et remplit assez bien les objectifs fondamentaux de ce type d'appareil. On a appris à extraire les informations arrivant "des étoiles" et tel qu'il est, notre petit instrument est déjà bien sympathique. Il reste toutefois un regret à formuler. Dans cette application on n'utilise pas la possibilité potentielle d'extraire du GPS notre vitesse par rapport au sol ainsi que notre cap. Pour développer ces fonctions il faudrait pouvoir les vérifier, c'est à dire embarquer notre module dans un véhicule automobile et observer son comportement. C'est faisable, encore faut-il qu'il soit autonome. Hors je ne dispose plus dans "mes stocks" de composants électroniques d'un afficheur vraiment de type LCD c'est à dire qui n'a pas besoin de rétro-éclairage. Celui des expérimentations est actif et consomme beaucoup trop pour envisager de l'alimenter avec un petit accumulateur 9V. Aussi je repousse ces études à une version utilisant un afficheur graphique infiniment moins gourmand en énergie que mon module prétendu LCD.

06) Deuxième et 3ième application autonome.

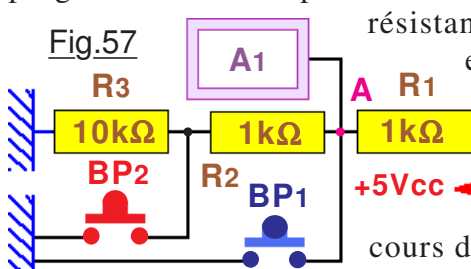
C'est pratiquement sa sœur jumelle avec quasiment les mêmes ingrédients. Comme tout le monde n'a pas forcément un codeur incrémental à sa disposition, on peut envisager de le remplacer par un petit clavier car des touches de toutes les tailles sont disponibles dans le commerce en ligne. Pour ma part je fais souvent appel à celles de la Fig.56 que l'on trouve par lot de cent. Outre leur petite taille, elles sont munies de cabochons de toutes les couleurs offrant un bon éventail de combinaisons envisageables. On peut ainsi choisir les teintes à notre guise. Sur cette photographie en **A** se trouve le corps des ces petits boutons poussoir. La partie active qui s'enfonce quand on clique porte un tenon conique sur lequel s'emboîte le cabochon **B** disponible en sept couleurs. En **C** l'un de ces composants est assemblé.

Ils se positionnent facilement sur une grille au pas standard. Autant dire l'idéal pour se concocter un petit clavier parfaitement adapté à notre besoin. Dans un premier temps, on va partir sur l'idée d'un petit *clavier à deux touches pour des raisons d'encombrement* et chercher une utilisation conviviale.



➤ **Expérience_24 : Clavier à deux boutons poussoir.**

Dans ce chapitre nous allons prendre en modèle un clavier élémentaire géré par **A1** par exemple. La Fig.57 reproduit le schéma électrique du dispositif de la Fig.58 (Voir également la photographie d'IMAGE 07.JPG) faisant partie intégrante des modules de mise au point de mes programmes. Non représenté sur le schéma, le circuit comporte en plus une LED jaune **L** avec sa



résistance de limitation de courant de **10kΩ**. Si le "strap" **S** est en place, elle indique la présence du **+5Vcc**. Si on enlève **S** du petit connecteur HE14 on dispose d'un témoin logique prêt à l'emploi.

La première source de parasitage dans la lecture d'une entrée analogique vient avec le "bruit alimentation". Considérons la Fig.59 qui montre l'évolution de la tension alimentation au cours de temps. En théorie la ligne USB du P.C. ou d'un

adaptateur secteur fournissent un **+5V** bien filtré et continu représenté par le trait rouge horizontal. Dans la réalité, à cette tension continue se superposent des variations très rapides que les techniciens nomment "herbe" ou "bruit blanc". Sur le dessin de la Fig.59 cette perturbation est

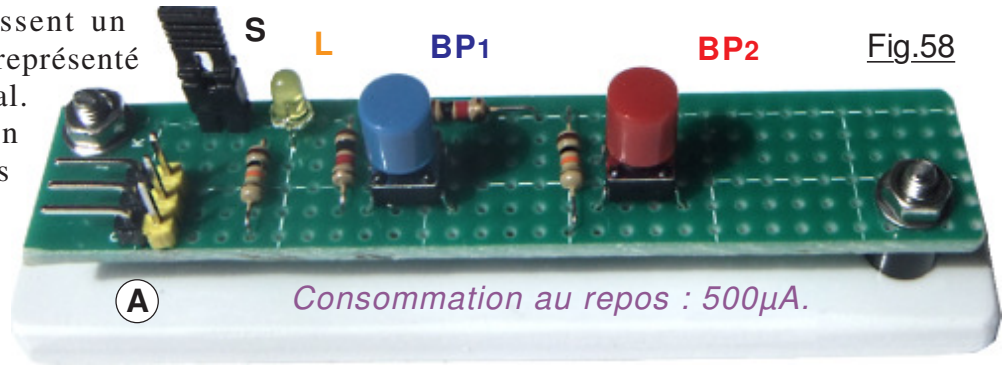
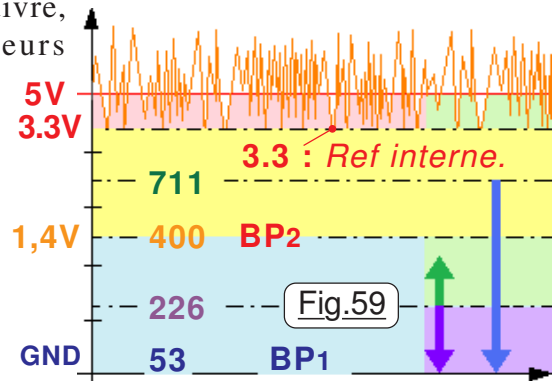


Fig.58

considérablement exagérée. (Ou alors l'adaptateur secteur doit être mis à la poubelle !) Même si l'alimentation est parfaite, le **+5V** du bloc secteur fournit la polarisation par la ligne électrique dans laquelle se trouve la résistance **R1** de protection de **1kΩ**. Elle est indispensable, car sans elle si l'opérateur clique sur le **BP1** on réalise un court-circuit franc entre le **+5V** et le **GND**. Cette ligne dans un environnement électromagnétique très encombré capte les parasites hertziens et sera forcément affectée par moment de petites impulsions parasites. Plus la résistance sera de valeur faible, moins ces phénomènes seront présents. À vide, en **A** de la Fig.57 la tension dépasse légèrement le seuil de comparaison interne de **3,3V** sur **A1**. Pour s'adapter à cette référence interne imposée par l'instruction `analogReference(INTERNAL)`; une résistance de **10kΩ** est ajoutée entre la broche qui devait aller au **+5V** et le **+Vcc**. Tout se passe maintenant comme si **R1** faisait **10kΩ**. La tension sur **A1** chute à environ **1,4V** et la numérisation par le **CAN** donne la valeur de **400** lorsque l'on clique sur **BP2**. Cliquer sur **BP1** a pour effet de faire un court-circuit entre **A1** et **GND**. Donc quand on cliquera sur le bouton poussoir **BP1**, **A** se retrouvera au potentiel nul. *Pour minimiser l'influence des parasites captés par la ligne des touches, il suffit de prendre comme seuil de décision la moitié de la tension présente entre celle du repos et celle de l'enfoncement de la touche concernée.* Ce qui compte pour le programme, ce n'est pas la tension mesurée, mais le résultat de la numérisation du **CAN**. Les valeurs seront directement concernées par celles des résistances **R1**, **R2** et **R3** qui ont une tolérance de fabrication. (Probablement dans les 5%.) Du coup sur votre prototype elles ne seront pas forcément les mêmes. Pour optimiser le code des études qui vont suivre,

Expérience_24.ino est conçu pour mesurer ces valeurs

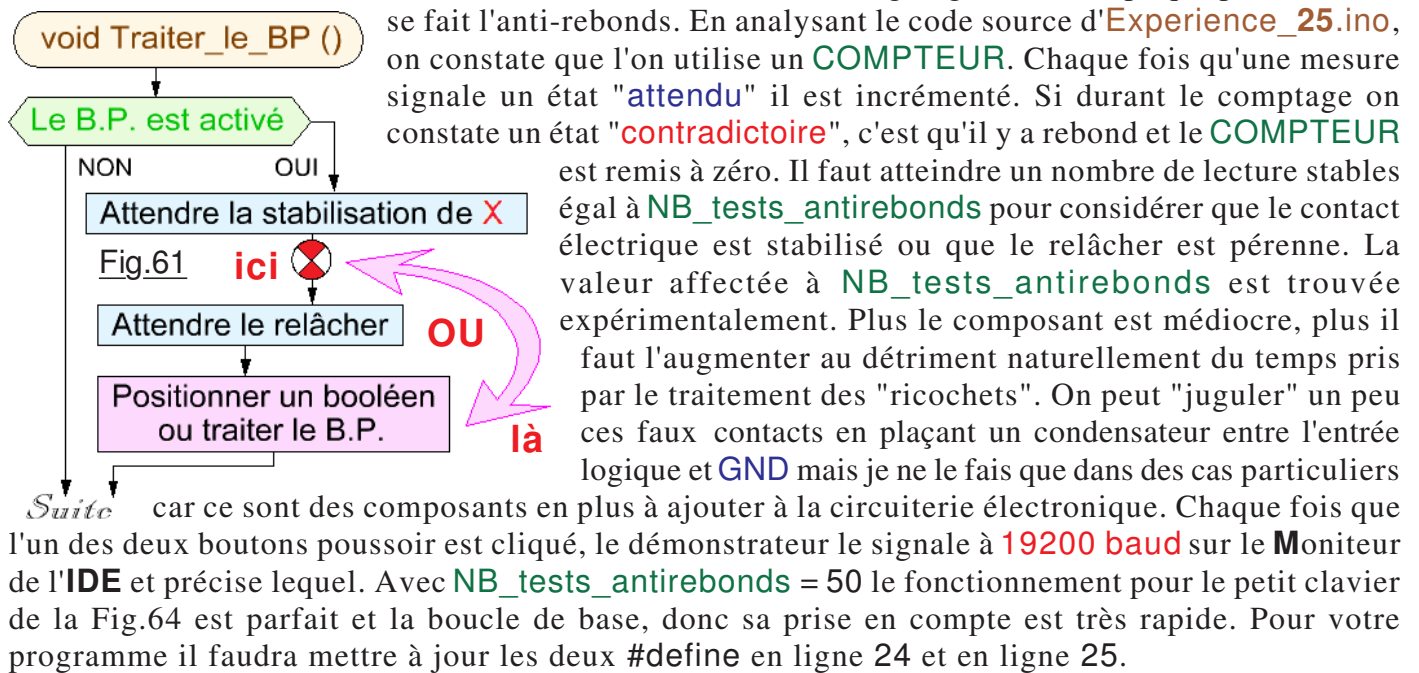
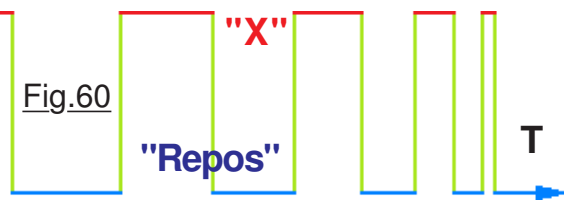
numérisées et les afficher en permanence à **19200 baud** sur le **Moniteur de l'IDE**. Sur mon exemplaire la valeur à vide est de **1023**. Cliquer sur **BP1** : La numérisation qui devrait en théorie donner zéro tourne aux environs de **53**. Agir ensuite sur **BP2** engendre une numérisation de l'ordre de **400**. Ces valeurs étant notées, nous allons pouvoir optimiser les programmes d'application. Sur la Fig.59 ont été reportées les valeurs numérisées avec le prototype. *Pour optimiser le filtrage des parasites on détermine des seuils de décision situé "à la moyenne" des valeurs mesurées.* Le premier seuil de décision vaut respectivement $400 + 1023 / 2 = 711$. Toute numérisation inférieure à cette valeur sera significative de l'activation de l'un des deux boutons poussoir. (Voir la flèche bleue). Le deuxième seuil vaut $53 + 400 / 2 = 226$ valeurs portées sur la Fig.59 situant la "frontière" entre **BP1** et **BP2**. Toute valeur plus grande que **226** désignera **BP2**, (Voir la flèche verte) toute valeur inférieure **BP1**. (Voir la flèche violette) Ces valeurs optimisées par mesurage seront à préciser par des `#define` dans les logiciels qui vont suivre.



➤ Expérience_25 : Assurer l'anti-rebonds.

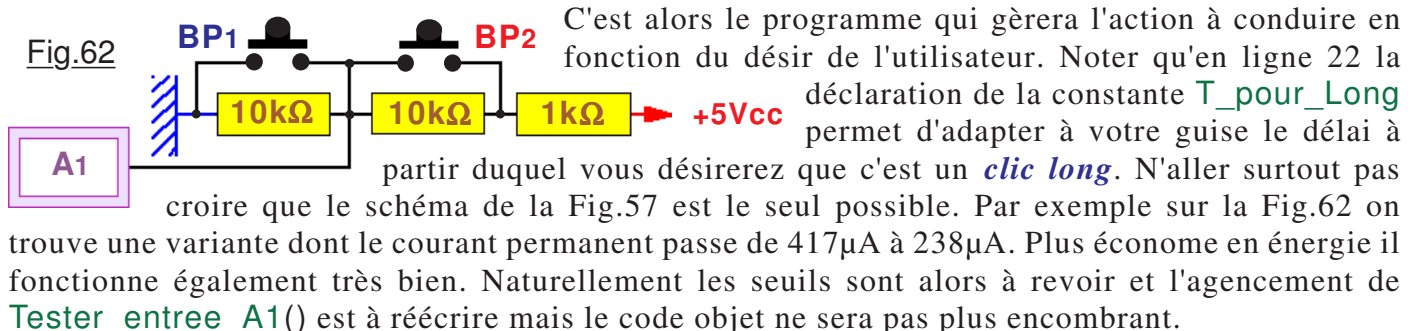
Avec **P24** on a réglé le problème des parasites hertziens mais il reste à éliminer les rebonds du contact électrique. Quand on bascule la mécanique d'un inverseur ou celle d'un bouton poussoir, on engendre le mouvement de pièces métalliques qui viennent en contact. Électriquement on génère un état "**X**" sur une entrée analogique ou binaire. Et bien cet état est inutilisable directement. En effet, *lorsque deux pièces mécaniques sont "bousculées" l'une contre l'autre, il y a des rebonds mécaniques* et l'évolution de l'état logique sur l'entrée concernée ressemble au signal de la Fig.60 qui correspond à un élément de très bonne qualité, car ici le contact ne rebondit que

quatre fois lors de l'activation. Les transitoires verts sont représentés par des "durées nulles" car ils se font très rapidement. *Si on ne tient pas compte de ce phénomène, alors le logiciel va croire que l'action sur le composant a été déclenchée quatre fois* et les traiter en conséquence. Dans la pratique, un interrupteur de qualité va présenter entre dix et vingt "ricochets". Un composant médiocre peut en générer jusqu'à deux cents voir plus ! Aussi, il faut systématiquement faire de l'*anti-rebonds*. Parmi les techniques ordinaires, on peut adopter celle de la Fig.61 dans laquelle le traitement peut être effectué juste après le relâcher du bouton poussoir en *là* ou dès que le clic est stabilisé en *ici*. L'approche "*ici*" est en général moins pénalisante en taille de programme, et peut surtout faire gagner un peu de temps en exécution si le composant met du temps à se stabiliser, car ce délai sera en partie "masqué" par la durée du traitement. La stratégie "*là*" impose la gestion d'un booléen mais autorise l'appel multiple à la séquence de surveillance d'une utilisation du clavier. (**P25** utilise *là*.) Cet organigramme n'explique pas comment



➤ **Expérience_26 : Quatre actions différentes avec seulement deux B.P.**

L'expérience montre que très souvent une petite application n'a pas besoin d'un grand nombre de touches sur le clavier servant au dialogue Homme/Machine. Il m'est arrivé souvent de n'installer que deux boutons poussoir comme montré sur la Fig.57 alors qu'il y avait quatre fonctions à gérer dans l'application. C'est tout à fait possible en faisant appel à la notion de *clic court* ou de *clic long*. Cette technique est applicable quel que soit le nombre de touches du clavier. Il suffit de chronométrer *la durée* du clic ... et chronométrer on sait faire ! Dès qu'elle *dépasse un seuil* déterminé expérimentalement, *le clic sera considéré comme long*. Pour mettre en œuvre **Expérience_26.ino** il n'y a strictement rien à modifier électriquement. On téléverse le "Sketch" et on se contente de cliquer sur **BP1** ou sur **BP2**. Comme pour **P25** le démonstrateur le signale à **19200 baud** sur le Moniteur de l'IDE, précise lequel est activé et en plus indique si c'est un *clic court* ou un *clic long*.



➤ **Experience_27 pour préparer la deuxième application.**

Par rapport à **Application_GPS_1.ino** elle ne fait que remplacer le codeur incrémental par un petit clavier personnel à deux touches. Il n'est toujours plus possible d'ajouter de nouvelles fonctions par manque de place entre la **PILE** et le **TAS**.

Globalement l'usage de l'appareil sera identique. Quand on tournait le codeur incrémental dans un sens ou dans l'autre, on changeait de PAGE écran sur le GPS ou on modifiait les options. Maintenant, le bouton rouge correspond à la rotation horaire, et la touche bleue fait revenir en arrière. Pour sortir d'une fonction ou valider une option, le **BPC** est remplacé par un **clic long** sur l'une quelconque des deux touches. Apprenons à utiliser le clavier à deux touches :

Manipulations :

- 01) Activer l'**IDE** et téléverser **Experience_27.ino**.
- 02) Ouvrir le **Moniteur** et vérifier que sa vitesse est de 19200bauds.
- 03) Faire un **RESET**. Attendre que l'appareil se mette à afficher la date et l'heure.

ATTENTION : Le clavier n'est pris en compte que lorsque l'affichage d'une PAGE sur l'écran LCD est achevée. Il peut s'écouler jusqu'à une seconde pour que le clic soit pris en compte. **Donc, attendre que la LED D13 d'Arduino s'allume. Relâcher immédiatement pour obtenir un clic court. Attendre que la LED bleue s'allume pour signaler un clic long.**

- 04) Faire un **clic court** sur le BP **rouge** : L'écran affiche la position géographique.
- 05) Continuer par un **clic court** sur le BP **rouge** : L'écran affiche la distance à la cible.
- 06) Poursuivre avec un **clic court** sur le BP **bleu** : L'écran revient en arrière sur les coordonnées.

 Touche rouge on avance dans le menu, Touche bleue on y recule.

- 07) Enchaîner des **clics courts** sur le BP **rouge** jusqu'à invoquer la PAGE qui affiche les caractéristiques de quatre satellites. La LED bleue s'allume précisant ce mode particulier. On peut cliquer rapidement sur les touches, elles sont immédiatement prises en compte. Les **clics courts** présentent les caractéristiques des satellites en permutation circulaire. La seule façon de sortir de ce mode qui éteindra la LED bleue consiste à effectuer un **clic long** sur l'un quelconque des deux BP qui fait revenir en première page du **Menu du GPS**.

➤ **Application_GPS_2.ino.**

Outre le remplacement du codeur incrémental par un petit clavier personnalisé, dans cette version du GPS on va changer radicalement de stratégie et effectuer une approche assez différente. En effet, avec les démonstrateurs précédents ou **Application_GPS_1.ino** les ressources de l'ATmega328 saturaient et il n'était plus possible d'ajouter des fonctions. Hors c'est tout à fait envisageable si l'on réalise des économies substantielles d'occupation mémoire et en particulier entre la **PILE** et le **TAS** car c'est actuellement la pierre d'achoppement. Pour contourner cette limite, dans la nouvelle mouture nous allons :

- **Placer les textes du dialogue Homme/Machine en mémoire non volatile EEPROM.** Dans ce but il faut y libérer de la place. À l'usage on constate que trente PI c'est vraiment inutile, avec dix on couvrira largement nos besoins ordinaires. Gain de place : $20 \times 33 = 660$ octets. Largement de quoi y loger tout les textes que l'on désire figer.
- Quand on efface un PI, décaler l'intégralité des emplacements est très pénalisant en temps de traitement. On ne déplacera plus que le nécessaire. Du coup la durée devient assez courte et il n'est plus nécessaire d'attirer l'attention de l'opérateur avec un BIP sonore.
- On va empêcher 'e' d'effacer tous les PI, il devra en rester au moins un. De ce fait le code objet s'allège de 26 octets et surtout il n'y a plus besoin d'un BIP sur **RESET** dont l'interprétation n'était pas évidente. (*BIP lorsqu'il n'y avait pas de PI à charger par défaut.*)
- L'espace mémoire disponible étant redevenu important, on pourra ajouter la fonction 'g' pour **Gommer (Effacer)** tous les PI sauf le premier, 'c' pour **Corriger** un PI et éventuellement prévoir la PAGE d'affichage de la vitesse et du cap s'il est possible de la développer correctement.

➤ Passage des textes du dialogue USB en EEPROM.

Par nature, les textes d'interfaçage entre l'ordinateur et l'humain ne changent pas. Ce sont intrinsèquement des données constantes. Hors pour le compilateur une **string** est un tableau de variables de type **char** que l'on peut manipuler à notre guise. Comme ces chaînes de caractères doivent être initialisées, elles sont déjà présentes dans l'espace réservé au programme. Mais en tant que tableau de variables, l'ATme328 subit une double peine : On doit leur réserver un emplacement en mémoire dynamique pour pouvoir les modifier à notre guise. Par exemple la chaîne "Bonjour." va consommer neuf octets dans la zone programme et ajouter également neuf octets en mémoire dynamique, donc diminuer d'autant l'espace libre entre la **PILE** et le **TAS**. (Neuf octets car il faut ajouter un octet au texte pour la sentinelle de fin de chaîne '\0'.)

NOTE : Généralement, lorsque je développe un projet, je loge systématiquement les tableaux de constantes en "haut" de la mémoire EEPROM. Puis, les variables à sauvegarder pour reprise du programme juste avant les constantes. Enfin, dans la place qui reste entre l'adresse \$0000 et le début des variables, je loge un maximum des textes servant au dialogue Homme/Machine. Ici, les 330 octets pour définir les PI continueront à loger à partir de l'adresse \$0000 pour ne par avoir à reprendre les routines concernées. Juste après on va gaver l'EEPROM avec les textes. Il n'y a pas de tableaux de constantes, juste quatre octets de variables pour les options.

L'implantation des données en mémoire non volatile est détaillée dans [Fiche n°19](#) et [Fiche n°20](#). On observe que l'EEPROM est entièrement utilisée. Il ne reste plus qu'un seul octet de libre représenté par '*' dans la zone bleue de la [Fiche n°20](#). Autant dire que pour la rentabilité du matériel on ne peut pas faire mieux. Le gain de mémoire obtenu est tel que non seulement il a été possible d'améliorer le programme, d'ajouter deux fonctions et le compilateur n'affiche toujours pas le message d'erreur de la Fig.53. On pourra ajouter du code si l'exploitation de la vitesse et du cap s'avère possible ...

➤ Fonction 'g' : Gommer tous les PI sauf le premier.

Quand nous avons désiré effacer les 30 PI, l'opération s'était avérée particulièrement indigeste. Dans la nouvelle mouture du logiciel, il n'y en a plus que neuf à effacer, et la manipulation reste encore un peu galère. Aussi, comme il reste de la place à revendre, autant ajouter une petite fonction "de luxe" pour faire tout le travail :

Manipulations : (Suite)

- 08) Maintenir l'un des deux BP cliqué et activer le **Moniteur**.
- 09) Lorsque la LED d'Arduino s'illumine, relâcher la touche.
- 10) Des **clics courts** sur le BP **rouge** font alterner entre la valeur **OUI** et la valeur **NON**.
- 11) Lorsque **OUI** est présent, engendrer un **clic long** sur l'une des deux touches. Le **Menu du RESET** montré en Fig.63 s'affiche dans la fenêtre contextuelle du **Moniteur**. Nous y trouvons deux nouvelles fonctions en **A** et en **B**.
- 12) Proposer **'g'**. Comme l'action peut s'avérer destructrice, un BIP sonore attire notre attention et le logiciel demande confirmation. Seule la lettre **'o'** ou **'O'** sera considérée comme une approbation. Confirmer avec **'o'**. Le programme affiche la liste des PI, et il n'en reste plus qu'un.
- 13) Réitérer la commande **'o'**. Le programme ne vérifie pas le nombre des PI. Il efface tous ceux qui sont après le premier même s'ils n'existent pas. Exactement comme le faisait la commande **'e'**, **le PI qui sera chargé par défaut sur RESET sera le n°1**.
- 14) Tester derechef la commande **'e'**. Le programme génère un BIP d'erreur, affiche le texte " **Il ne reste qu'un seul P.I.**" et refuse maintenant d'effacer le dernier PI présent en EEPROM. Ainsi sur RESET on sera certain de charger des données cohérentes.
- 15) Petite exploration de la mémoire avec **'m'**. On peut vérifier sur la Fig.64 que l'espace disponible entre la **PILE** et le **TAS** pour les données dynamique fait maintenant 877 octets. Il n'y a strictement aucun risque de **collision de pile**.

Fig.63

? ou , : Rappel de ces commandes.
 A ou a : Ajouter un PI.
C ou c : Corriger un PI. (A)
 D ou d : Déplacer un PI.
 E ou e : Effacer un PI. (B)
G ou g : Gommer tous les PI - 1.
 L ou l : Listage des PI en EEPROM.
 M ou m : Mémoire dynamique.
 Q ou q : Quitter les saisies.
 S ou s : Silencieux.

Fig.64

Commande : [M]
 Version du 13/02/2025
PILE - TAS = 877

➤ **Fonction 'c' : Corriger le contenu d'un PI.**

Autant la commande 'g' ne sera pas utilisée souvent et constitue un petit plus, autant pouvoir corriger l'une des trois entités d'un PI qui contient une erreur peut s'avérer bien pratique. Pour vérifier que l'on peut modifier n'importe quel PI présent, nous allons réinscrire en EEPROM les exemples initiaux pour avoir un "tronc commun".

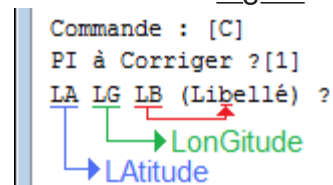
Manipulations : (Suite)

- 16) Téléverser une deuxième fois l'outil **Ecrire_en_EEPROM_1.ino** et l'exécuter au moins une fois en activant le **Moniteur** de l'**IDE**.
- 17) Téléverser ensuite **Application_GPS_2.ino** pour retrouver le programme d'exploitation.
- 18) Effectuer les manipulations pour activer le "dialogue H/M sur USB".
- 19) Commande '**L**' pour vérifier la présence des sept PI.
- 20) Nous commençons par corriger le premier : '**c**' suivi de '**1**'.
- 21) Le programme désire alors savoir quel est des trois items celui que l'on désire modifier. Comme montré sur la Fig.65 on doit le préciser par deux lettres, **LA**, **LG** ou **LB**. C'est la deuxième lettre qui sera analysée par le programme pour vérifier si elle est correcte et orienter le traitement. Tester "**If**" par exemple. Comme la deuxième lettre n'est pas légale on est tancé d'un message d'erreur "**Item incorrect.**" suivi d'un BIP d'alerte.
- Fig.65**

```
Commande : [C]
PI à Corriger ?[1]
LA LG LB (Libellé) ?
      |  |  |
      |  |  +--> LonGitude
      |  +-----> LAtitude
      +-----> LAtitude
```

The diagram illustrates the command prompt interface. It shows the command 'Commande : [C]' followed by 'PI à Corriger ?[1]'. Below this, the prompt 'LA LG LB (Libellé) ?' is shown. Three arrows originate from the letters: a blue arrow from 'LA' points to 'LAtitude', a green arrow from 'LG' points to 'LonGitude', and a red arrow from 'LB' points to 'Libellé'.

Fig.65



NOTE : comme on va le voir dans les manipulations qui suivent, pour simplifier le programme, la première lettre n'est pas vérifiée ni la longueur du texte.

- 22) Tester '**c**' suivi de '**1**' puis de "**xa**" : Bien que le premier caractère ne soit pas correct, pour le logiciel la consigne est claire et il nous demande la valeur de la Latitude. Proposer "**1234567s**" par exemple. (*Penser éventuellement à du Copier/Coller comme déjà mentionné.*)
- 23) Maintenant tester '**c**' suivi de '**7**' puis de "**xgmmmmmmmmmmmmmm**" par exemple. Le deuxième caractère étant légal, la Longitude est requise. Proposer "**12345678w**".
- 24) On continue avec '**c**' suivi de '**5**' puis de "**xbssssssss**" par exemple. Commande acceptée le programme va modifier le nom du PI : "**Bonjour les amis.**". On remarque qu'avec le point final on dépasse 16 caractères et il est ignoré. En revanche la casse est conservée pour les noms.
- 25) Dernier test correct : '**c**' suivi de '**2**' puis de "**xg**" par exemple avec "**12345678w**".

Ces manipulations montrent que l'on peut modifier l'item que l'on désire sur n'importe quel PI. Ce sera bien commode si on s'est trompé sur une cible, car nous ne sommes plus obligés de tout retaper.

- 26) Pour finir, '**c**' suivi de '**9**' : L'analyseur syntaxique surveille la logique de nos consigne.

➤ Une fonction ' p ' un peu tardive.

A ce stade du développement, une idée d'amélioration du dialogue H/M a émergé des manipulations qui précèdent dans le didacticiel. En effet, quand on ajoute un PI en EEPROM, c'est que probablement il va devenir celui par défaut. Alors, pouvoir le désigner dans le dialogue USB par la fonction 'p' a été ajouté. L'espace entre la PILE et le TAS a un peu diminué à 859 octets qui sont plus que suffisants pour garantir une bonne stabilité du programme. Ces deux modifications sont commentées sur la Fig.66 et ne coutent que 124 octets ce qui n'est vraiment pas onéreux.

Manipulations : (Suite)

- 27) Avec 'p' désigner le PI n°3 par exemple suivi de 'L'. (Voir la Fig.66)
- 28) Commande 'm' pour constater que maintenant l'espace entre la PILE et le TAS fait 859 octets.
- 29) Indiquer 'p', PI n°1 suivi de 'L' on vérifie que l'on peut bien désigner le premier.
- 30) Recommencer et désigner le dernier.
- 31) Tester 'p' puis désigner le PI n°9. On se doutait que l'analyseur syntaxique n'accepterait pas.
- 32) Proposer 'L' pour vérifier que rien n'a changé.
- 31) Commande 'g' et accepter. On constate qu'automatiquement le PI n°1 qui reste est maintenant celui qui sera chargé par défaut sur RESET.
- 32) Pour finir on va *ajouter le pôle SUD* pour préparer les prochaines manipulations :
- Num 01 : 9000000N 00000000E [Pole NORD.] < RESET

Num 02 : 9000000S 07300000W [Pole SUD.]
- Page 35**

```

? ou , : Ce MENU.
A ou a : Ajouter un PI.
C ou c : Corriger un PI.
D ou d : Déplacer un PI.
E ou e : Effacer un PI.
G ou g : Gommer tous les PI - 1.
L ou l : Listage des PI en EEPROM.
M ou m : Mémoire dynamique.
P ou p : PI sur RESET.
Q ou q : Quitter les saisies.
S ou s : Silencieux.
Commande : [M]
Version du 14/02/2025
PILE - TAS = 859
Commande : [P]
PI sur RESET ?[3]
Commande : [L]
Num 01 : 0000000N [NORD.]
Num 02 : 4737359 [Y.]
Num 03 : 4437053 [nsayes.]
Num 04 : 438376 [MES.]
Num 05 : 48853 [ble NORD.]
Num 06 : 12345 [a Maison.]
Num 07 : 9000 [Pole SUD.]

```

Fig.66

L'affichage du rappel des commandes a été modifié. Pour gagner de la place en EEPROM le texte "**Rappel de ces commandes.**" s'est simplifié et en **A** devient "**Ce MENU.**". En **B** est ajoutée la présence de la nouvelle commande '**P**'. Quand on utilise cette commande, en

C le logiciel demande quel sera le PI qui sera chargé par défaut sur RESET. À partir de cette dernière version du logiciel lors du listage des PI présents en mémoire non volatile EEPROM,

B celui qui est désigné au rechargement est pointé par "<RESET" repéré en **D** sur la copie d'écran. Sans que ce soit une révolution, pour la bagatelle de 124 octets de programme en plus, l'utilisation du petit appareil est encore plus agréable.

> Un point singulier.

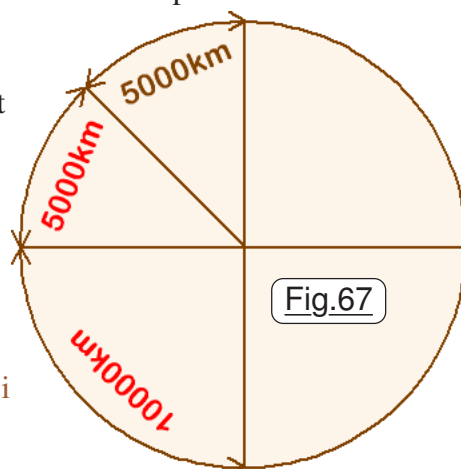
Pluriel aurait été plus circonstancié, car dans ce dernier chapitre nous allons aborder le cas particulier des deux pôles. Ils sont particuliers à plus d'un titre comme nous l'avons abordé dans le document, [Cartographie et loxodromie.pdf](#) et dans ce chapitre nous allons voir la conséquence du fait que les pôles sont à la convergence de tous les méridiens. De ce fait, pour calculer

D la valeur de la loxodromie, en théorie, que l'on donne 0,15,-73 pour la valeur de la longitude, la distance calculée doit en principe rester identique. C'est précisément ce que l'on va vérifier.

Manipulations : (Suite)

- 33) Avec '**q**' passer dans le **Menu du GPS**.
- 34) Cliquer deux fois sur le BP rouge. La distance cible fait 5062,34km ce qui est cohérent. En effet, le périmètre terrestre avoisine 40000km. Le GPS étant aux environs de la Latitude +45°, La distance qui nous sépare du pôle Nord est de $40000 / 8$ soit 5000km. (Voir la Fig.67)
- 35) Revenir dans le dialogue H/M sur la ligne USB. (Vous devez savoir faire maintenant.)
- 36) Modifier la longitude avec '**c**', '**1**', "**lg**" et "**12100000w**".
- 37) Consigner '**q**' suivi de deux clics sur le BP rouge. On retrouve la distance de 5062,34km.
- 38) Reprendre le dialogue H/M sur la ligne USB. (La routine à ce stade.)
- 39) Imposer '**p**' désigner le PI n°2 et vérifier par '**L**'.
- 40) On recommence avec Consigner '**q**' suivi de deux clics sur le BP rouge. Cette fois la distance fait 14941,18km. C'est logique, elle avoisine $40000 / 8 * 3$ soit 15000km. La précision des calculs de loxodromie sur des méridiens sont confirmés.
- 41) Provoquons un dernier retour dans le dialogue H/M sur USB.
- 42) Dernière modification de la longitude avec '**c**', '**2**', "**lg**" et indiquer vers l'est cette fois : "**16500000e**".
- 43) Encore '**q**' suivi de deux clics sur le BP rouge. On retrouve la distance de 14941,18km. **OUF ... tout fonctionne !**

Pour le plaisir, remarquez au passage que le pôle Nord et le seul endroit sur Terre où quelle que soit la direction dans laquelle vous vous retournez, vous regardez plein SUD. Du coup, pour partir vers une destination précise ... la boussole qui de plus pointe à la verticale, n'est strictement plus utilisable.



Mis à par le fait que la vitesse et le cap n'ont pas été explorés, on peut considérer qu'en l'état le programme est mature et "complet". Pour celles et ceux qui ont la version "codeur incrémental", je vous ai concocté [Application_GPS_3.ino](#) pour remplacer la gestion du clavier par celle du capteur rotatif sachant que si l'on n'a pas trop l'habitude, c'est inévitablement un travail un peu laborieux et toutes et tous n'aurez pas forcément le temps de disponible pour vous y coller. On peut passer à ce stade à une autre version utilisant un écran graphique.

07) Quatrième application autonome.

Intégrant un petit afficheur OLED bien moins gourmand en énergie que mon module LCD avec rétro-éclairage, il sera alors possible d'utiliser un banc d'essai en autonomie dans un véhicule automobile et tester la possibilité d'extraire des données satellites la vitesse sol et le cap actuel. Les afficheurs OLED existent en taille et en définition très variables. Je vais donc pour cette application utiliser celui de 1,3 pouce de diagonale monochrome disponible dans mes composants prévus pour expérimentation. Ce modèle existe en blanc ou en bleu. Au final, il me semble plus rapide de reprendre [Application_GPS_3.ino](#) et de remplacer une à une les routines d'affichage que de tout reconstruire à partir d'[Experience_19_Noyau.ino](#). Ce travail préparatoire sera effectué dans le démonstrateur [Expérience_28.ino](#) avant de tester de nouvelles fonctions.

➤ **Expérience_28 : Préparation de l'Application autonome n°3.**

Côté matériel, l'architecture électronique s'oriente vers le schéma proposé dans la [Fiche n°21](#). Avant d'envisager la liberté, le système étant autonome et "transportable" pour pouvoir être expérimenté dans un véhicule automobile en déplacement, il faut réécrire les fonctions pour présenter les informations sur l'afficheur OLED. Le

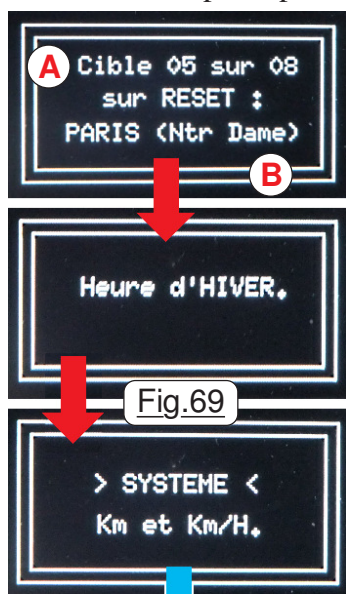


Fig.69

Menu du GPS

dialogue Homme/Machine sur la ligne USB reste strictement inchangé et l'ordre de saisie des options sur RESET reste également le même. Seules les présentations à l'écran sont légèrement modifiées. **ATTENTION** : Sur la Fig.68 les mosaïques sont blanches et l'ordre Vcc et GND est montré dans l'encadré rouge.

En fonction du fournisseur ces deux broches peuvent être permutées. Donc bien observer la sérigraphie pour la mise en œuvre correcte. Étant sur la page-écran de la Fig.68, si on clique sur le **BPC** on enchaîne sur le dialogue via le **Moniteur de l'IDE** sa sortie activant le **Menu du GPS**. Si l'option est sur **NON** on bascule dans les options de la Fig.69 du **Menu du RESET**. En **A** le programme précise qu'il y a actuellement huit PI en EEPROM et que c'est la 5 qui est indexée pour le chargement par défaut. En **B** est listé le nom de ce point d'intérêt. Tourner le codeur incrémental change l'indexation et liste en permutation circulaire les PI disponibles.

Cliquer sur le **BPC** valide l'option et fait passer au choix de l'heure d'été ou d'hiver. Tourner le bouton change l'option, le **BPC** fait alors passer aux unités de distance et de vitesse. La validation de l'option active le **Menu du GPS**.



Fig.68

➤ **Le menu de l'Application autonome n°3.**

Comme avec le type d'afficheur utilisé on peut présenter facilement jusqu'à quatre lignes de texte, nous allons en profiter pour réorganiser les fonctions de base et diminuer le nombre des pages-écran simplifiant ainsi l'utilisation de l'appareil. Les nouveaux écrans sont montrés dans l'ordre des photographies de la Fig.71 avec en Fig.70 l'attente d'un nombre suffisant de satellites.



Fig.70

En **C** on retrouve un compteur qui recycle à 255 et qui permet à l'opérateur de vérifier que la boucle de base n'est pas inerte. Par ailleurs, toujours dans ce but, en mode GPS et présentation des données, la LED Arduino s'allume durant l'affichage sur OLED. L'ordre des pages-écran en Fig.71 est dicté par l'importance que l'utilisateur accorde à leur contenu. Il est

évident que d'une personne à une autre le point de vue pourra être totalement évident. Il vous sera très facile de le modifier. Il suffit dans le **switch (Page_Ecran)** de la procédure **void Affiche_la_PAGE_Ecran()** d'intervertir les **indices** de **Affiche_la_PAGE_N()**; dans les diverses lignes des instructions **case**. Pour ma part, c'est la page **B** qui sera probablement la plus utilisée. Toutefois, si j'ai mis en premier l'affichage **A** c'est qu'il saute plus aux yeux et permet du premier coup d'œil de constater que le nombre suffisant de satellites a été capté pour avoir des données cohérentes. L'affichage en **C** est provisoire. Cette troisième page sera consacrée à la vitesse sol et au CAP si j'arrive à extraire correctement ces données lorsque le GPS sera en mouvement.

Comme la page-écran **C** n'est pas encore validée, pour sa mise au point le contenu de la trame de type **VTG** est listée sur les deux lignes du haut. Les deux lignes du bas occuperont toute la page si ces séquences peuvent fonctionner. Comme en laboratoire le GPS est immobile, ces données ne sont pas cohérentes. Toutefois, dans une optique de mise au point du programme, les données sont extraites en position correcte dans **Tampon_VTG[n]** et coloriées pour faire ressortir la logique de l'extraction et de l'affichage. Si l'option choisie utilise les unités marines, le logiciel en tient compte tant en sélection des données que dans l'affichage des unités. Dans la page **D** sont résumées les options actuelles et dans la ligne du bas est

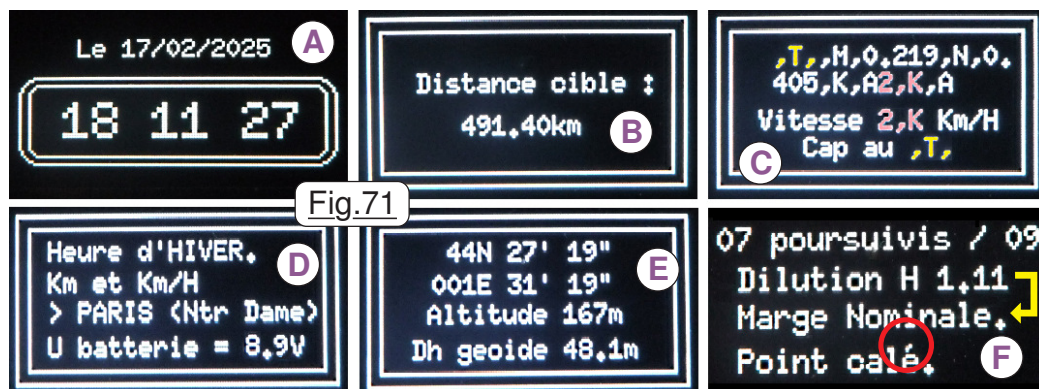


Fig.71

précisée la tension en entrée **Vin**. Dans la troisième ligne est précisé le nom de la **Cible**. En **E** sont précisées les coordonnées géographique, l'altitude et la correction en hauteur **Δh** par rapport au géoïde de référence. La lettre **Δ** est remplacée par **D** pour ne pas avoir à la construire avec trois segments de droite ce qui alourdirait trop le code objet. Enfin en **F** sont résumés les facteurs de fiabilité. Sur la ligne du haut on lit qu'actuellement neuf satellites sont en visibilité mais que seulement sept sont en poursuite et aptes à servir aux calculs. Du facteur de dilution en est déduit le type de précision officiel, **Nominale** étant le meilleur. Enfin en ligne du bas on trouve le type de positionnement. Dans l'encadré rouge on remarque que le caractère 'e' est bien accentué. Ce type d'attribut n'existe pas dans la police de caractères OLED. Aussi il est ajouté sous la forme de deux PIXELs moins gourmands en code objet que si l'on avait utilisé le tracé de lignes.

➤ La liberté dans une boîte de chaussures !

Boîtier non encore réalisé, car le choix définitif de l'afficheur qui sera utilisé n'est pas encore décidé, pour tester en mobile le programme il faut bien organiser le banc d'essais pour qu'il soit transportable et autonome. La Fig.72 présente "le laboratoire embarqué" avec tous les éléments

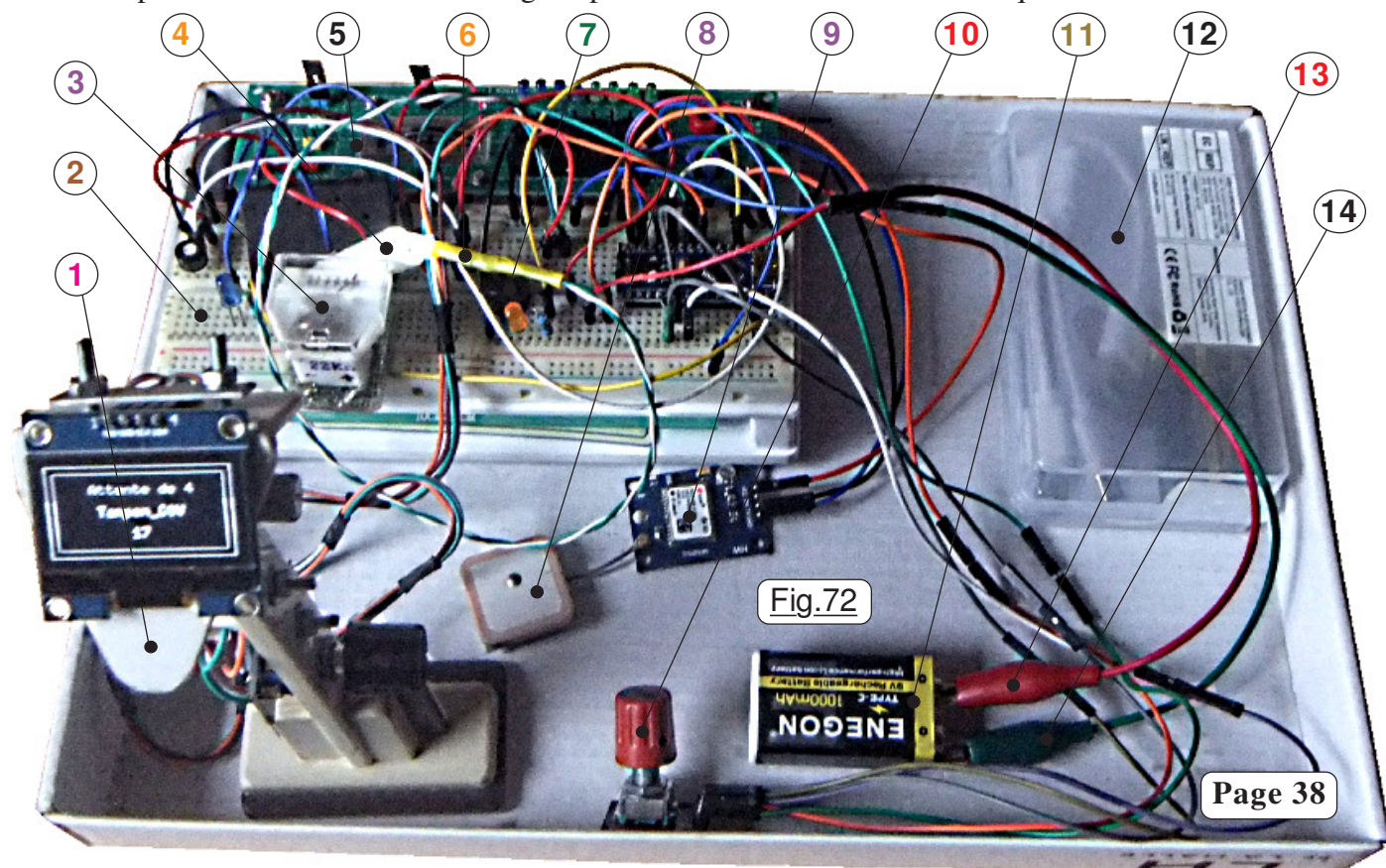
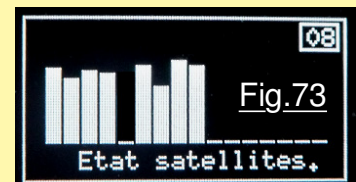


Fig.72

vitaux réunis dans le couvercle relativement rigide d'une boîte de chaussure qui fait merveille. Pincettes crocodiles **13** et **14** ainsi que plaque à essais **2** sont les deux mamelles de la mise au point du matériel et du logiciel. Ici en débranchant **13** on coupe l'alimentation par l'accumulateur rechargeable **11** ce qui permet alors de pouvoir brancher la ligne USB à l'ordinateur de développement. En **1** on trouve une colonne qui supporte OLED bien au dessus des fils de liaison. En situation mobile ce support un peu haut est couché. En **3** un petit galvanomètre surveille le **+5Vcc** d'alimentation de la carte NANO. Très lumineuse en **6** se trouve une petite LED blanche d'éclairage local chaque fois que je dois modifier un branchement. Pour ne pas qu'elle n'éblouisse, en temps normal elle est insérée dans le petit "abat-jour" **4**, mais ici elle s'est un peu écartée. Tout le long de la platine de raccordement **2** sur laquelle il y a Arduino, se trouve en arrière plan un complément expérimental **5** avec ampèremètre, huit LEDs de témoins logiques et des rampes de branchement. Pour ces activités elle est ignorée mais étant sur une semelle commune avec **2** elle est forcément présente et il ne faut pas en tenir compte. En **7** se trouve, pas très visible, le bruiteur et la LED orange mise en série. En **8** l'antenne du GPS est reliée à son module **9** et l'ensemble peut être placé horizontalement ou verticalement pour chercher si l'une des deux orientations ne serait pas plus favorable que l'autre. Le codeur rotatif relié avec des lignes un peu longues se trouve en **10** alors qu'en **11** se trouve le petit accumulateur de 9Vcc. Il est réputé présenter une capacité de 1000MAH. Des mesures ont montré que dans la réalité c'est assez exagéré. Le module GPS se taille la part du lion en gloutonnant ses 50mA. Le reste de l'électronique plafonne à 45mA. Avec une intensité de 95mA on peut toutefois espérer une autonomie comprise entre 4H et 6H à 7H, les mesures n'ont pas encore été effectuées pour le vérifier. Enfin, c'est toujours au moment d'utiliser un appareil que l'on réalise que les piles sont vides. Aussi, disposant d'un deuxième accumulateur qui a été chargé parallèlement à celui qui est en service, dans le boîtier **12** se trouve cet élément de réserve. Enfin, lors d'une "expédition expérimentale" il faut ajouter un petit carnet pour prendre des notes et naturellement ... de quoi écrire. La toute première sortie est un échec, raison pour laquelle l'écran **C** conserve son aspect provisoire. Pour ne pas encombrer le didacticiel, le document [Compte-rendu.txt](#) détaille l'expérience.

Un changement de stratégie vient simplifier le logiciel et faciliter les manipulations. La phase d'attente de données cohérentes est abandonnée. Le programme fonctionne directement dans l'affichage des pages-écran et le codeur incrémental est immédiatement fonctionnel. On peut ainsi consulter les options mémorisées même si les données ne sont pas cohérentes. La PAGE_4 change un peu de contenu. Le caractère '>' pour l'affichage de la cible a été supprimé n'étant pas pertinent. Ce n'est plus la PAGE_1 qui est affichée en démarrage du logiciel, mais la PAGE_7 en Fig.73 qui présente le nombre de satellites en visibilité et dans le petit encadré au haut à droite le nombre de ces derniers qui servent à extraire les données et à effectuer les calculs de position.



➤ La page de présence des satellites.

Gourmande en octets de code objet, elle sera toutefois retenue pour l'application n°3 car elle participe à rendre plus agréable l'utilisation du GPS et contribue à minimiser certaines séquences de code qui font gagner de la place pour le programme. Par ailleurs, à ce stade on ne consomme encore que 85% de la zone mémoire dédiée au code objet, et toutes les fonctions désirées initialement sont en place. On n'exploite pas les données d'orientation des satellites ni leur élévation au dessus de l'horizon, mais ce sont vraiment des informations secondaires. Il sera certainement plus "amusant" d'utiliser les possibilités graphiques de l'afficheur que de consommer des ressources pour montrer ces indications peu importantes.

Bien que ce ne soit pas fréquent, il peut arriver que l'on capte jusqu'à quatre trames de type **GSV**. Comme chaque trame peut contenir jusqu'à quatre identifications de satellites, on peut avoir à afficher potentiellement jusqu'à 16 colonnes représentant le niveau de réception de ceux qui sont visibles. L'un d'entre eux peut fort bien avoir un niveau de réception nul comme le cinquième par exemple. Jusqu'à présent le plus fort niveau reçu et observé a été de **43**. Si par la suite une valeur plus grande survenait, l'affichage déborderait en hauteur. Il suffirait dans ce cas de modifier l'instruction située en tête de listage `#define MAX_NIV_sat 43`.

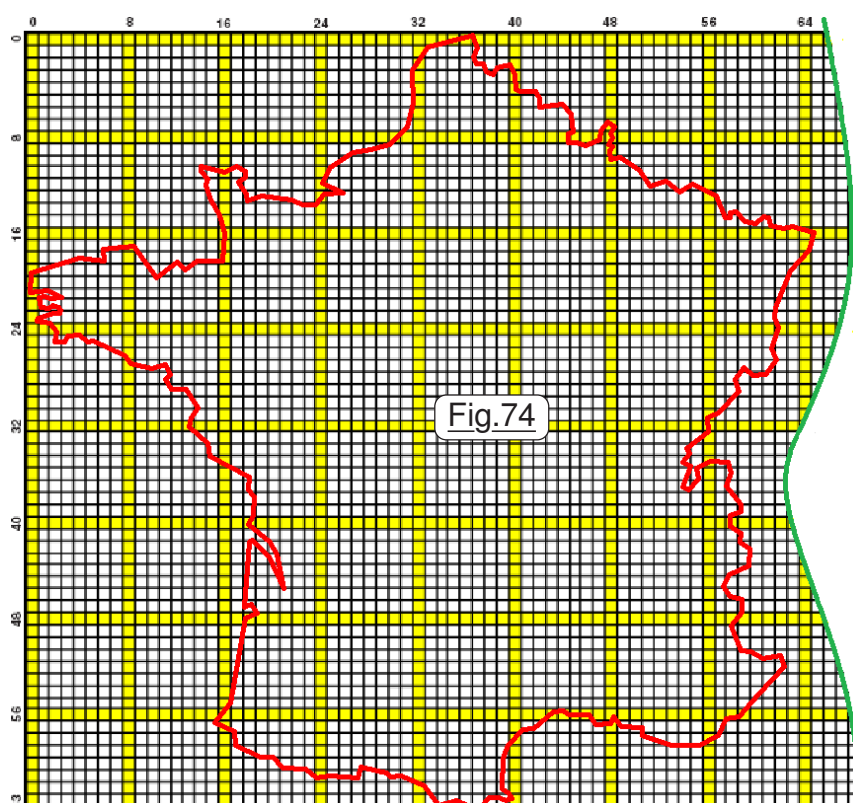
➤ L'application autonome n°4.

Elle reprend intégralement les fonctions des différentes pages-écran. Les protocoles d'initialisation des paramètres, le menu du RESET et le dialogue Homme/Machine avec le Moniteur de l'IDE sont inchangés. Naturellement, le schéma électronique reste intégralement celui présenté en Fiche n°21. L'un des avantages notable des afficheurs OLED réside dans l'aptitude à les piloter avec uniquement deux fils auquel naturellement on doit ajouter le +Vcc et GND. L'interface entre Arduino et le module électronique se fait par l'entremise des protocoles I2C. Pour les intégrer dans le logiciel on bénéficie de la bibliothèque "Two Wire Interface" native dans le compilateur. Pour celle et ceux qui désirent en savoir plus sur les protocoles I2C je vous propose les Fiche n°15 et Fiche n°16. Pour faire plus ample connaissance avec l'afficheur OLED de 1,3 pouce de diagonale, c'est la Fiche n°13 qu'il vous faut consulter. L'ordre d'affichage et un peu modifié. C'est toujours la page qui affiche la présence des satellites qui est présentée sur RESET. Mais on a intercalé une carte graphique qui affiche les contours de la France et y positionne le GPS et la Cible.

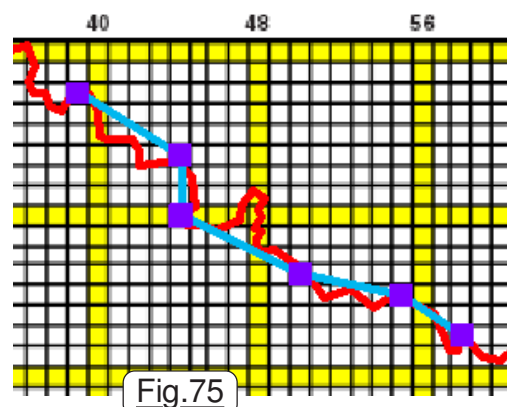
➤ Construire le dessin de la France sur l'écran graphique.

Avec une définition de 64 x 128 points lumineux, il ne faut pas rêver, on ne va pas présenter une carte avec le nom des rues et représenter la maisonnette de Tante Gertrude. D'autant plus, que la France s'inscrivant globalement dans un hexagone, la surface utilisée sera pratiquement carrée. La définition devient 64 x 64 PIXELs et le dessin va ressembler à une caricature. Toutefois, dans le cadre d'une activité ludique, il serait dommage de ne pas aborder le graphisme en mode filaire. L'objet de ce chapitre est de vous présenter l'approche utilisée pour construire cette petite carte rudimentaire.

Première étape : Préparer le travail de codage en C++. Il serait totalement illusoire de se mettre face à l'écran et de commencer à programmer. La première phase consiste à définir ce que l'on désire obtenir. Pour ma part, j'ai réalisé un dessin que j'imprime sur une feuille de papier format A4 horizontal. Vous pouvez en faire autant car je vous propose dans le répertoire <DOCUMENTS> ce petit outil dans le fichier GRILLE pour OLED.pdf qui outre la grille précise les coordonnées de chaque ligne et de chaque colonne. Il sera alors évident de repérer la position de n'importe quel point. On commence par décider dans quelle zone sera représenté le dessin désiré. Dans notre cas j'ai opté pour un décalage complètement à gauche, laissant ainsi la possibilité à droite d'ajouter du texte. L'idée consiste alors à faire entrer l'intégralité des formes désirées de façon à ce qu'elles utilisent toute la Hauteur. On se doute que la belle carte de la Fig.74 ne sera pas possible la définition de l'écran étant bien insuffisante. Par ailleurs il faut choisir entre tracer toutes les frontières par des



PIXELs qui se touchent, ou adopter un contour plus symbolique en traçant des segments de droite contigus. C'est cette deuxième option qui sera choisie, car avec environ 33 segments on arrive à tracer l'intégralité de la carte, alors qu'il faudrait définir environ 600 à 700 dalles pour tracer le contour, ce qui en terme de taille de programme est naturellement inenvisageable.



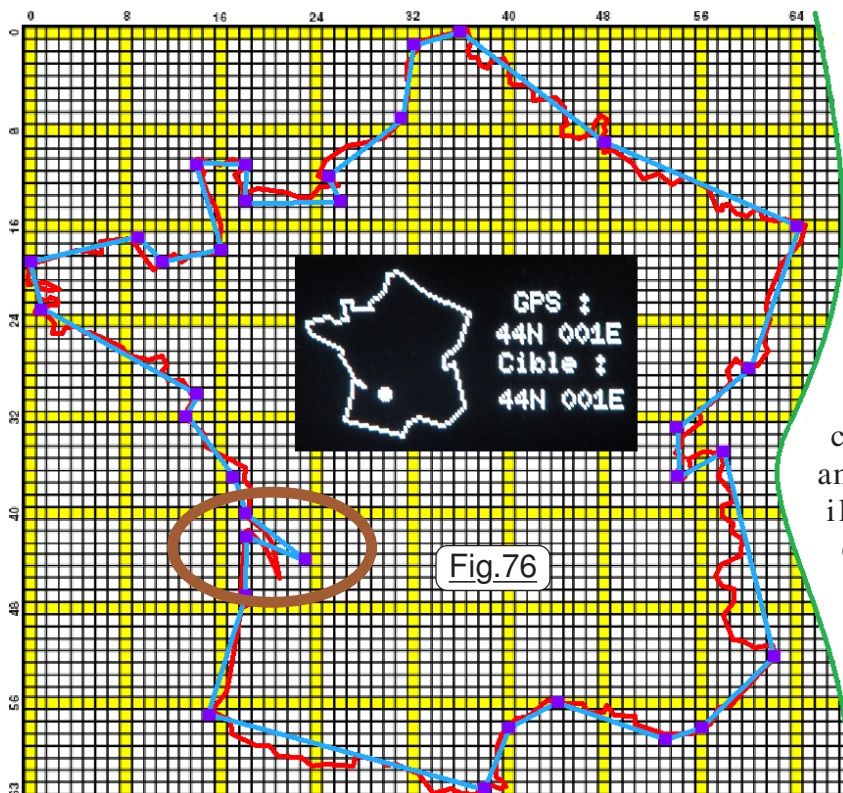


Fig.76

agréable à utiliser ne consomme que 7% de l'espace dédié au programme ce qui n'est pas exagéré, car les graphiques sont toujours boulimiques en octets de code objet. Sur la Fig.76 dans la copie d'écran en noir, il n'y a qu'un seul point représenté. Dans cet exemple, les coordonnées de la cible et celles du GPS sont identiques. C'est tout simplement que le véhicule automobile est arrivé à sa destination. Les séquences de calcul pour positionner les deux "points" ont été assez délicates à mettre au point. Aussi une campagne d'essais pertinente a été utilisée. Dans ce but,

Num 01 :	4445544N	00152192E	[Ma Maison.]
Num 02 :	4737359N	00312634E	[Donzy.]
Num 03 :	4437051N	00480745E	[Clansayes.]
Num 04 :	4383763N	00438378E	[NIMES.]
Num 05 :	4885365N	00235026E	[PARIS (Ntr Dame)] < RESET
Num 06 :	9000000N	00000000E	[Pole NORD.]
Num 07 :	9000000S	90000000W	[Pole SUD.]
Num 08 :	2718624S	10943481W	[Ile de Pâques.]
Num 09 :	4839964N	00449736W	[BREST.]
Num 10 :	4858698N	00774154E	[Strasbourg.]

Fig.77

le **BPC** active ou désactive le tracé de la loxodromie entre le GPS et la cible. Par exemple en **B** cette dernière est invalidée. C'est **Strasbourg** en zone violette qui a été choisie car située le plus à l'Est. En **C** c'est **BREST** (Zone verte.) qui est sélectionnée car située le plus à l'Ouest. Son disque représentatif n'est représenté qu'à moitié, le coté gauche étant hors matrice de PIXELs. Pour repérer ce disque incomplet la loxodromie a été validée. Il importe de noter que **le programme est simplifié et ne teste pas la position des deux disques. Si les positions sont hors de la France, ils seront tracés n'importe où**. Par exemple en **D** c'est l'île de Pâques qui est ciblée alors qu'en **E** c'est le Pôle SUD. Nous avons déjà vu que la saisie des données de PI n'est pas analysée sur les valeurs indiquées.

Alors la longitude de 900°W étant erronée, la loxodromie n'est pas verticale comme on s'y attendrait. Je vous invite fortement à tester avec Bordeaux, Marseille ou Dunkerque par exemple pour compléter les tests et valider la séquence.

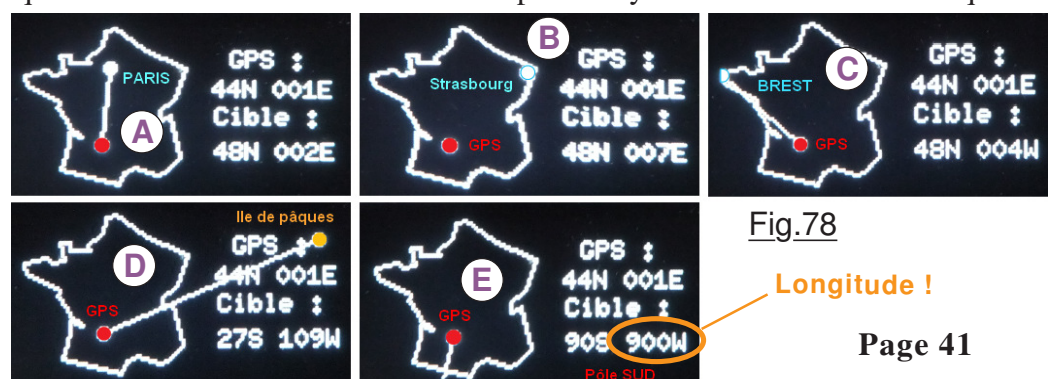
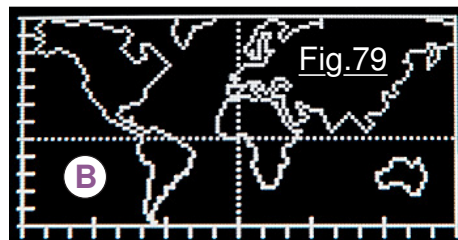
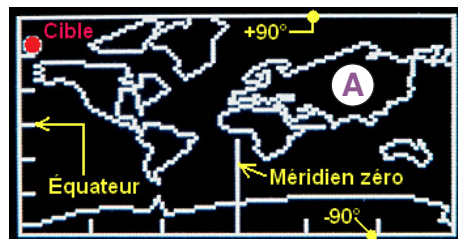


Fig.78

Longitude !

➤ Affichage du planisphère avec positionnement de la cible.

Constatant qu'il restait de la place pour loger du code et que dans mes applications je cherche toujours à "saturer" le matériel, j'ai décidé d'ajouter une page graphique couvrant l'intégralité du globe. Pour pouvoir voyager dans des contrées lointaines il faut disposer de temps de liberté et surtout de pouvoir financer les voyages. Il est assez-peu probable que ce sera la majorité des lectrices et des lecteurs. Toutefois, il m'a semblé séduisant de créer un petit GPS certes modeste, mais pouvant être utilisé n'importe où sur Terre. (*J'ai limité les latitudes entre -50° et $+70^\circ$ pour des raisons précisées plus avant, mais je reste persuadé que pratiquement personne n'ira se perdre dans les deux "solitudes polaires".*) Par ailleurs, cette nouvelle fonction est particulièrement pertinente au point de vue exemple de programmation car elle réalise ... l'impossible et on va y revenir. En première approche, j'avais envisagé comme montré sur la Fig.79A de représenter l'intégralité du globe. De fait, plus d'un tiers de la surface en hauteur était occupée par les régions polaires engendrant un tassement exagéré des continents habités. L'imprécision de positionnement qui résulte des informations données dans le document [Cartographie et Loxodromie.pdf](#) engendrait un positionnement un peu trop haut de la Cible qui sur cet écran est Point Hope. Bien trop au dessus de l'Alaska.



Comme aller dans ces régions désertiques glacée constitue un rare privilège d'exception, représenter ces zones "blanche" sur notre planisphère est strictement sans intérêt. J'ai donc recalculé tous les points représentatifs de la carte en limitant à -50° la latitude SUD et à $+70^\circ$ la latitude Nord. On peut observer sur la Fig.79B que le visuel est nettement plus crédible. D'autres parts le positionnement des points y est bien plus rigoureux. Enfin les graduations tous les 10° tant en latitude qu'en longitude sont plus faciles à tracer sur le petit écran.

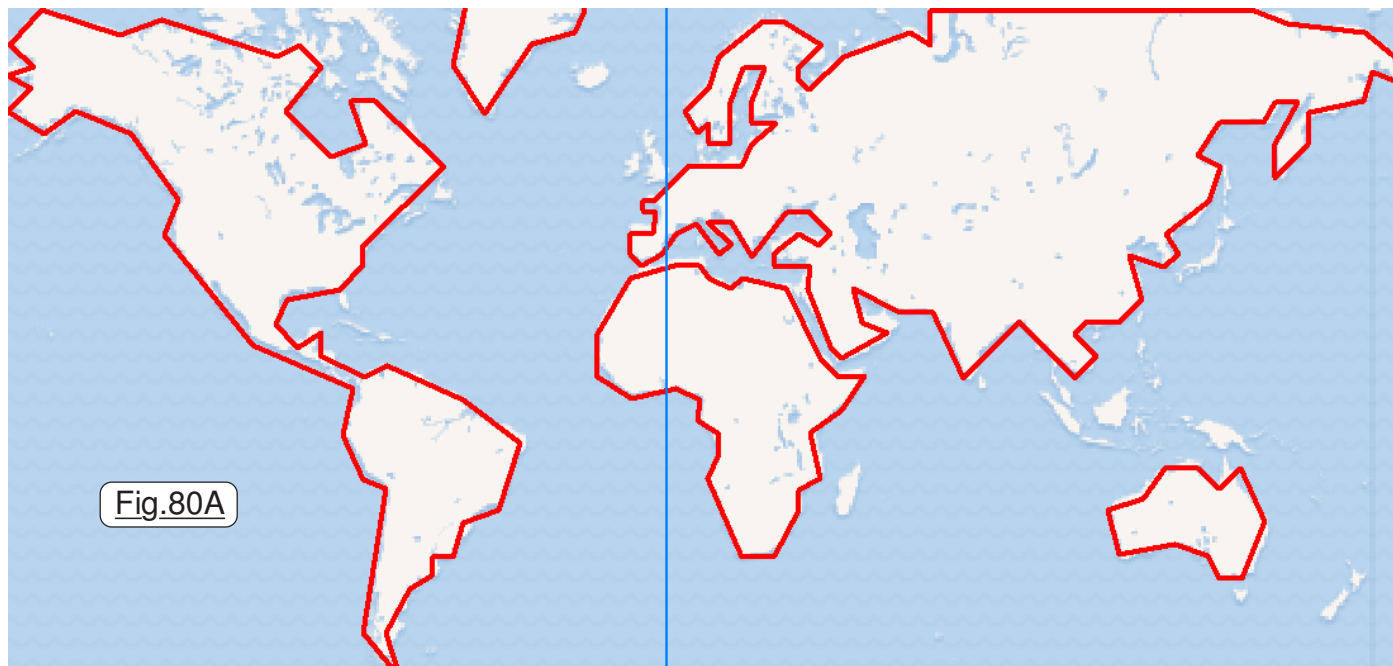
➤ Construction du planisphère.

L'impossible est contourné par logiciel. Avant de s'aventurer dans un travail aussi considérable que tracer segment de droite par segment de droite une carte sur un dallage de luminophores, il importe d'en vérifier la faisabilité. Pour tracer la carte de France, il a fallu 33 segments de droite. Chaque segment consomme 16 octets entre les coordonnées, les passages de paramètres et l'appel à la procédure. Pour tracer le planisphère, ce n'est pas moins de 178 segments sans compter les graduations. On aboutit à $16 \times 178 = 2848$ octets. C'est impossible car à ce stade il ne restait que 2406 octets de disponible pour le programme. Et pourtant on va rendre possible l'impossible !

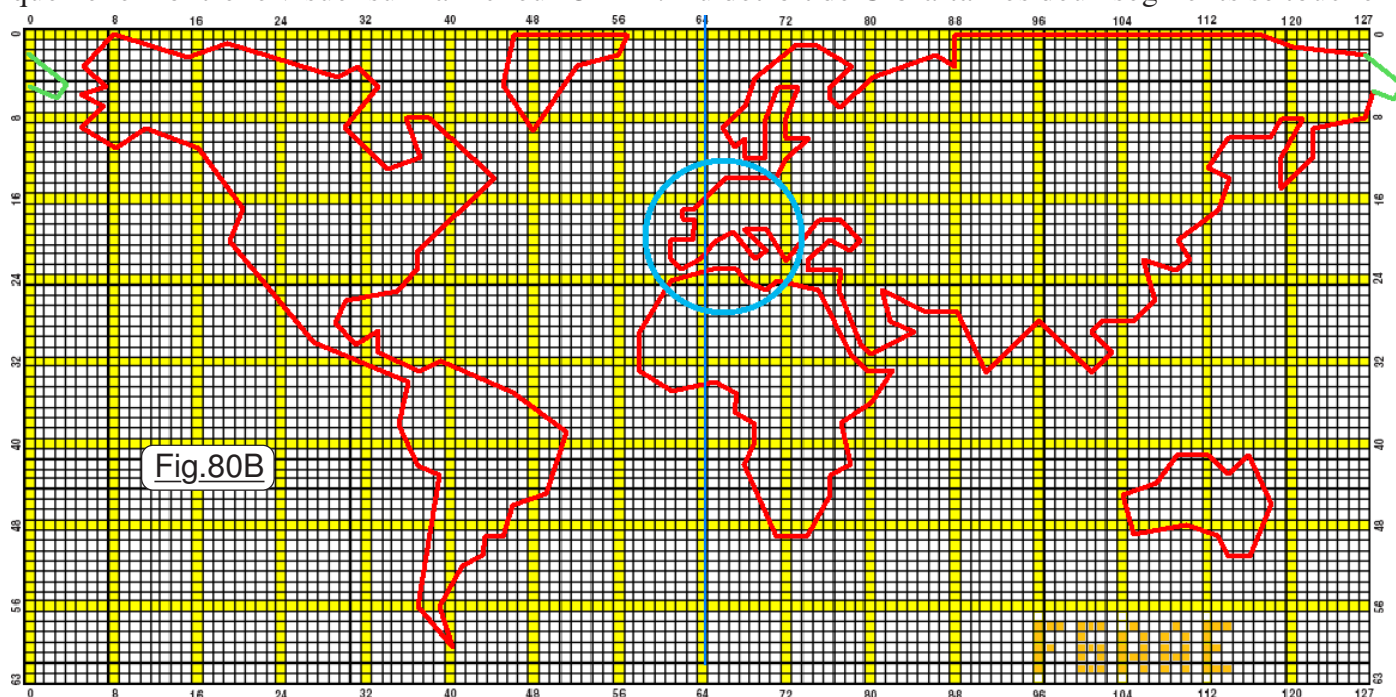
ANALYSE : Quand on utilise une instruction de type `u8g.drawLine(53,59,44,56)`; il faut pour chaque instruction préciser `Xd`, `Yd`, `Xf`, `Yf`. Hors sauf quatre cas particuliers, tous les segments sont contigus. Du coup on précise deux fois les mêmes points. Si l'on décrit les coordonnées dans un tableau de `byte`, il suffit de ne décrire qu'une fois chaque point. On divise déjà par deux la taille des données. Par ailleurs au lieu d'écrire chaque instruction de type `u8g.drawLine` on peut procéder par usage de boucles `for`. Pour l'ensemble de la carte on ne fait plus appel que 20 fois à cette procédure. Par adoption de cette approche, non seulement la fonction devient possible, mais entièrement émulée il reste encore 798 octets dans l'espace réservé au programme.

➤ Détermination des coordonnées des extrémités des segments.

Sachant que le méridien zéro passe par Greenwich, il semble tout naturel de cadrer horizontalement la mappemonde en centrant ce méridien horizontalement. Du coup, la France qui adopte l'heure de ce méridien se trouvera centrée latéralement sur la carte. La première étape montrée en Fig.80A pour déterminer les positions des extrémités des segments dans notre matrice de 128×64 points consiste à sélectionner la zone à représenter sur un planisphère et à y construire notre tracé en tenant compte du fait qu'il faut absolument aboutir à un compromis entre diminution maximale du nombre de segments et dessin résultant crédible. Comme on travaille sur une grande surface, celle du fichier [GRILLE pour OLED.pdf](#) imprimé sur une page de format A4, on a tendance à exagérer la finesse du tracé d'autant plus que sur l'écran OLED deux lignes qui se



touchent engendrent une surface et l'on ne distingue plus les deux segments. Sur cette mappemonde le méridien de Greenwich n'est pas centré. La première adaptation va consister à décaler latéralement l'ensemble pour centrer la "ligne bleue". La deuxième modification nous oblige à centrer les extrémités des segments sur les luminophores de la mosaïque de l'afficheur OLED. On aboutit au compromis de la Fig.80B sur laquelle on constate que la France et l'Italie dans l'encadré bleu sont plus détaillées que ne le montre le visuel sur l'afficheur OLED. Au détroit de Gibraltar les deux segments se touchent



et sur OLED la zone reste assez confuse. Il est clair qu'il aurait été possible de simplifier quelques segments dans cette zone, mais à ce stade du développement ce serait du "gagne petit". On remarquera que les trois segments verts "sortant du cadre" à droite n'ont pas été reproduits à gauche. Noyés dans les graduations de latitude ils n'auraient qu'ajouter de la confusion. Alors autant simplifier.

➤ **Un jeu d'essais complet.**

Dans la mesure où l'on ne peut pas démonter la justesse d'un programme, il faut impérativement organiser une campagne de test la plus complète possible qui embrasse les cas classiques, et surtout les cas particuliers avec spécifiquement ceux qui engendrent des "effets de bord". C'est particulièrement le cas sur un écran graphique où l'on va représenter la position de la cible par un petit disque qui peut se trouver centré sur le cadre extérieur de la carte. Il importe alors de vérifier que dans ces cas "marginaux" les tracés restent cohérents. Dans ce but nous allons

sélectionner à la surface du globe des cibles situées dans des endroits caractéristiques et vérifier que les petits disques de situation sont correctement placés. Outre l'île de Pâques qui était dans la liste des PI logés en EEPROM, on va compléter notre campagne de tests par les villes ou les points singuliers listés sur la copie d'écran de la Fig.81 qui dans les zones jaunes situent des villes "à l'intérieur du cadre". Sur toutes les photographies de l'afficheur OLED les cibles ont été surchargées par un petit disque rouge pour en faciliter le repérage. En **A** la ville de **Sydney** à été choisie car elle est loin à l'Est de l'hexagone et située très au Sud. Son positionnement sur la carte est parfait. Plus centrée en **B** et également très au Sud **Johannesburg** est également bien située. Enfin en **C** la célèbre **Miami** a été ajoutée car très éloignée à l'Ouest de la France

Num 01 :	5425339S	06679756W	[Ushuaïa.
Num 02 :	6466548N	17480279E	[Krasneno.
Num 03 :	3383868S	15102668E	[Sydney.
Num 04 :	2619940S	02800521E	[Johannesburg.
Num 05 :	6834644N	16681247W	[Point Hope.
Num 06 :	2575368N	08021325W	[Miami.
Num 07 :	9000000N	09000000E	[Pôle Nord 1.
Num 08 :	9000000N	18000000W	[Pôle Nord 2.
Num 09 :	9000000S	18000000E	[Pôle Sud 1.
Num 10 :	9000000S	00000000E	[Pôle Sud 2.

Fig.81

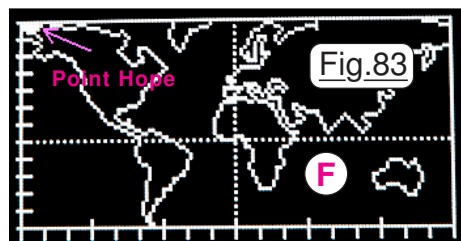
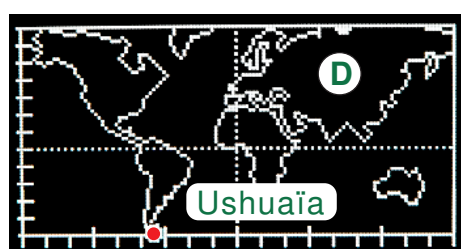
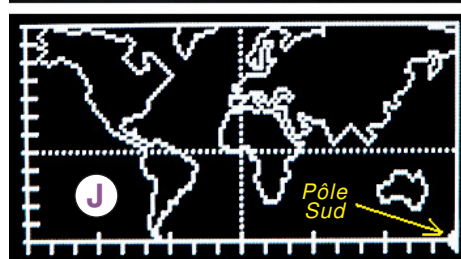


Fig.83



et située dans l'hémisphère Nord. Globalement les placements sur la carte sont corrects, il nous reste à tester les cas extrêmes. On commence avec la ville d'**Ushuaïa** en **D** de la Fig.83, célèbre car c'est la plus au Sud dans notre monde. Pour sa part, **Krasneno** en **E** est située complètement à droite et vers le Nord. Enfin, Tout à l'Ouest de l'Alaska en **F** se trouve complètement à gauche

Point Hope. Pour les deux extrêmes latérales le programme est également validé. Il reste encore à traiter le cas étrange du pôle Nord et du pôle sud *qui sont deux points représentés* ici par le cadre haut et le cadre bas *sur toute la largeur de l'écran*. On va tester avec diverses longitudes. En Fig.84 **G** le pole est positionné à 90° de latitude Est. En **H** il est à peine visible car tout en haut et tout à gauche de la matrice de PIXELs. Théoriquement il ne devrait pas se trouver sur l'écran car à +90° de latitude. Tout point situé plus haut que la latitude 70° Nord sera représenté sur le coté haut du cadre. Le demi-disque supérieur n'est pas représenté car hors mosaïques lumineuses. Pour les latitudes Sud on retrouve un filtrage analogue et toute latitude inférieure à -50° sera

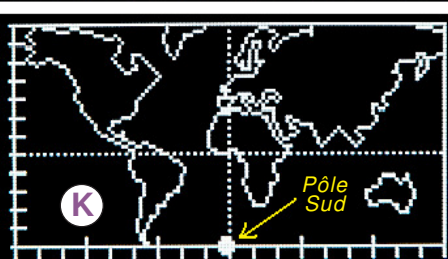
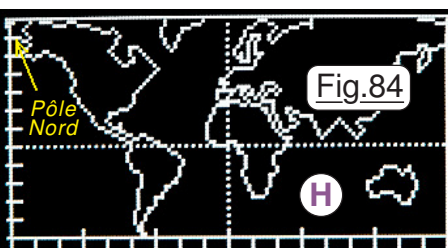


Fig.84

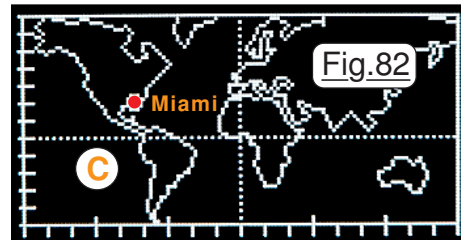
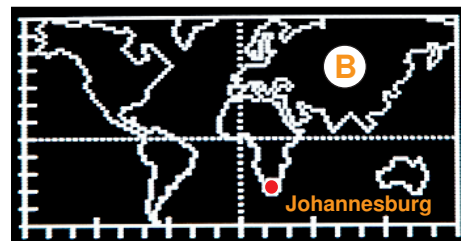
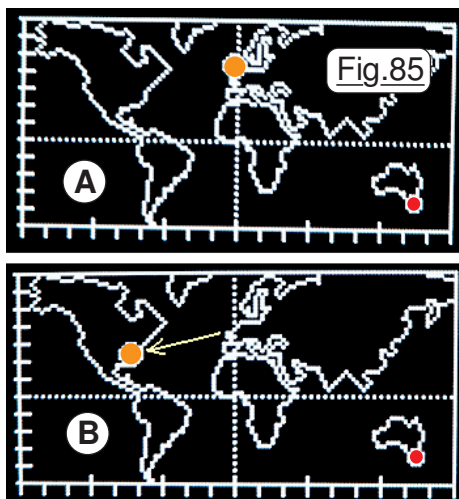


Fig.82

"rabotée" à cette valeur. Par contre, en **K** le disque symbole est entier car contenu dans la mosaïque contrairement à celui en **J** dont la moitié droite est hors luminophores. En **K** la longitude est nulle et le cercle semble placé sur la verticale à gauche de la latitude zéro de Greenwich tracée en pointillés. Le cadre est de largeur paire, alors le centre théorique est déterminé à ± 1 .



Sur toutes les images du chapitre précédent seul le disque représentatif de la **Cible** est représenté. Si l'on désire ajouter le symbole de position du **GPS** il suffit de cliquer sur le **BPC** qui se comporte comme une bascule de type OUI/NON. Pour différencier les deux repères, celui du **GPS** est légèrement plus grand. Du coup, on le constate sur la Fig.85 **A**, il masque complètement la France et l'Italie. Aussi, comme généralement nous allons utiliser ce petit appareil en France, j'ai pensé que de ne pas l'afficher par défaut serait préférable. Son affichage est donc une option. Noter qu'il faut cliquer assez longuement pour que le **BPC** soit pris en compte, car il n'est testé qu'une fois par seconde. Pour vérifier l'universalité de comportement du programme, en **B** nous avons voyagé virtuellement jusqu'à **Miami**.

Il suffit dans le programme de modifier en **true** le **false** la ligne `#define Voyager_a_Miami false` située dans les paramètres en tête de listage. Du reste vous pouvez tester n'importe quelle autre destination en remplaçant les coordonnées dans l'instruction qui utilise `if (Voyager_a_Miami)`.

➤ Le mode cheminement.

Ayant intégré la fonction Planisphère, force est de constater qu'il reste encore 798 octets de disponible pour le programme. *(En fait si on compile avec une version plus récente de l'IDE, par exemple la version 1.8 on gagnerait encore environ 1800 octets car elle est bien plus optimisée. C'est pour des raisons personnelles que je préfère la 1.7.9 et je ne passe à plus récent que lorsque je n'ai plus assez de place pour arriver à développer un projet plus conséquent.)* Ne pas employer ces 798 octets, c'est un peu agassif, une sorte de gaspillage. Aussi, j'ai cherché une idée qui pourrait s'avérer séduisante, d'où l'idée de la fonction "CHEMINEMENT". L'idée directrice est assez simple. Elle consiste à considérer qu'une suite de PI sauvegardés en EEPROM constitue une route que l'on désire parcourir jalonnée par ces points de passages. Quand on tourne le codeur incrémental en antihoraire, juste avant la page de l'affichage graphique de la France, on ouvre la **Fonction Cheminement**. Pour nous prévenir que l'on ne peut sortir de cette fonction qu'avec le **BPC**, la LED bleue s'allume jusqu'à revenir dans le **Menu de base du GPS** sur la page d'affichage de la date et de l'heure. En **Fonction Cheminement** la rotation du codeur incrémental fait changer de cible en permutation circulaire, les "sauts" étant imposés par le sens de rotation. L'écran OLED présente les informations de la Fig.86 qui précise qu'elle est la prochaine **Cible** et la distance à laquelle elle se trouve. Si durant le voyage on n'intervient pas, chaque fois que l'on va se trouver à moins de 2Km de l'un des jalons, le programme changera automatiquement de **Cible** et passera à la suivante sauf si c'est la dernière. Si par exemple on a changé de route pour éviter un incident et que le jalon prévu n'a pas été approché suffisamment, il suffit de tourner le codeur incrémental pour passer au suivant. L'implémentation de la **Fonction Cheminement** ne rend pas caduque le choix du PI par défaut car on doit pouvoir conserver cet automatisme, nous évitant d'avoir à choisir un PI chaque fois que le GPS est remis sous tension. Par contre, si on va un peu loin, et que le GPS soit coupé durant les pauses, il faut reprendre l'initialisation des phases **A** et **B**. En entrée de fonction, lorsque la LED bleue s'allume, en Fig.87 **A** le programme nous invite à lui préciser le PI d'arrivée qui peut être n'importe laquelle des cibles enregistrées en EEPROM. Puis en **B** c'est le premier PI du cheminement qu'il faut indiquer au logiciel. Tous les PI trouvés dans l'ordre entre **DEBUT** et **ARRIVEE** constituent le cheminement supposé et seront surveillés au cours du déroulement du programme. *(L'interface de dialogue commence par demander l'ARRIVÉE car ainsi on diminue un peu la taille du programme.)*



Fig.86

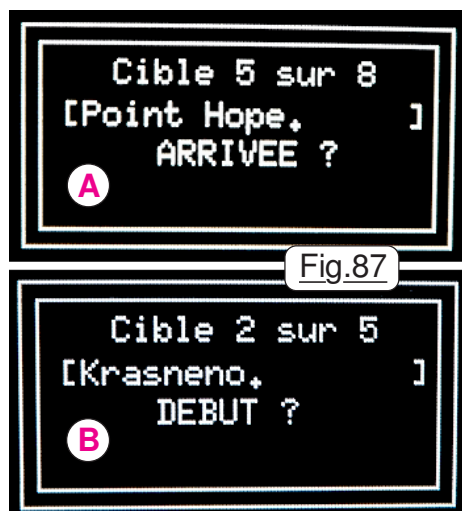


Fig.87

➤ Un petit exercice pratique.

Observant la Fig.81 on arrive à la conclusion que le voyageur va partir de France, aller à Krasneno, puis à Sydney, puis à Johannesburg pour finir à Point Hope. Soit il fait son voyage en vélo, soit il prend l'avion ... et bonjour le réchauffement climatique ! Plus simplement, dans l'exercice qui va suivre, nous allons envisager le cas d'un transporteur qui va aller charger à PARIS, puis livrer à LYON et à MARSEILLES. Ce petit parcours a pour but de montrer que l'on peut logger un cheminement n'importe où en EEPROM. En situation de départ on a la liste de PI montrés sur le listage de la Fig.81 dans laquelle on doit faire un peu de place.

Manipulations :

- 01) Pour commencer sur une base commune, nous allons effectuer un RESET en ouvrant le **Moniteur** de l'**IDE** le **BPC** étant activé.
- 02) Attendre que la LED Arduino s'allume et le relâcher.
- 03) Tourner le codeur incrémental pour faire afficher l'option **OUI**.
- 04) Cliquer sur le **BPC** : La liste des PI s'affiche dans la fenêtre contextuelle du **Moniteur**.
- 05) Effacer les trois derniers PI n°8, n°9 et n°10.
- 06) Ajouter PARIS comme premier jalon. (48886198N 00234655E PARIS. (Halles))
- 07) Déplacer ce PI n°8 en position n°4 à la place de Johannesburg.
- 08) Ajouter LYON comme deuxième jalon. (4576270N 00485100E LYON. (P Bocuse))
- 09) Déplacer ce PI n°9 à la place de celui de Point Hope en n°5.
- 10) Enfin compléter avec le dernier jalon. (4329312N 00537331E MARSEILLES)
- 11) Il nous reste à déplacer le PI n° 10 à la place de celui de Miami.

On se trouve avec l'organisation de la Fig.88 dans laquelle en vert se trouvent les PI déplacés, le but étant de montrer qu'un cheminement peut se trouver n'importe où en EEPROM.

- 12) Commande '**q**' pour quitter le dialogue USB et retrouver le **Menu de base du GPS**.
- 13) Revenir en arrière de trois pas pour activer la **Fonction Cheminement**.
- 14) Désigner **MARSEILLES** comme jalon.
- 15) Cliquer sur le **BPC** et indexer **PARIS** qui sera notre première cible pour le trajet envisagé.
- 16) Activer le **BPC** et aller sur la carte de France. L'image confirme la position.
- 17) Revenir sur la **Fonction Cheminement**. Et reprendre l'initialisation n°6 puis n°4.
- 18) Tourner le codeur incrémental et cette fois pointer **LYON**.
- 19) Sortir et revenir sur la carte de France : Lyon est bien situé sur le dessin.
- 20) Réinvoquer la **Fonction Cheminement** et réitérer l'initialisation n°6 puis n°4.
- 21) Enfin tourner le codeur incrémental pour désigner la destination d'arrivée.
- 22) Un clic un peu long sur le **BPC** pour retrouver le **Menu de base du GPS**.
- 23) Encore trois pas dans le sens antihoraire pour faire afficher la carte de France. OUF, la cité phocéenne est toujours au bon endroit.

Avec ces manipulations on a vu que chaque fois que l'on quittera un cheminement pour une quelconque raison, il faudra à nouveau réintroduire les jalons de **DEBUT** et d'**ARRIVEE**. Ce n'est pas tragique car c'est assez rapide. Si on a oublié l'ordre des jalons, dans un premier temps on peut tous les faire "défiler" à l'écran. Actuellement il ne reste plus que 48 octets de disponibles en mémoire réservée pour le programme, et l'EEPROM est saturée. Autant dire que pour la rentabilité matérielle on ne peut pas faire mieux, on a utilisé pratiquement toutes les ressources de l'ATmega328. Espérons que nous n'aurons pas besoin de place pour corriger d'éventuelles erreurs de programme !

Fig.88

Num 01 :	5425339S	06679756W	[Ushuaia.]
Num 02 :	6466548N	17480279E	[Krasneno.]
Num 03 :	3383868S	15102668E	[Sydney.]
Num 04 :	4886198N	00234655E	[PARIS. (Halles)]
Num 05 :	4576270N	00485100E	[LYON. (P Bocuse)]
Num 06 :	4329312N	00537331E	[MARSEILLES]
Num 07 :	9000000N	09000000E	[Pôle Nord 1.]
Num 08 :	2619940S	02800521E	[Johannesburg.]
Num 09 :	6834644N	16681247W	[Point Hope.]
Num 10 :	2575368N	08021325W	[Miami.]



Ben Môa môa je dis qu'un transporteur qui n'a pas de GPS dans son gros camion et qui doit s'en faire un avec Arduino aurait tout intérêt à changer d'entreprise !

08) Cinquième application autonome.

L'internaute qui lit ce didacticiel pourrait trouver étrange que l'on passe à une nouvelle version du petit GPS, alors que dans les précédentes l'affichage de la vitesse sol et du cap suivi ne sont pas encore validés et totalement mis au point. Il se trouve que les expéditions du genre de celle abordée avec la Fig.72 ne sont pas aisées à mettre en œuvre imposant des déplacements dans les activités ordinaires. Ce type d'expérience sera grandement facilité lorsque l'appareil sera fonctionnel dans un petit boîtier. Hors, tant que toutes les options d'affichage n'auront pas été explorées, ne connaissant pas le composant final qui emportera ma préférence, il serait précipité de développer le circuit imprimé et d'agencer le coffret. Pour cette cinquième version, je vais conserver le codeur incrémental qui emporte ma préférence. Le logiciel sera entièrement refondu, car l'écran graphique sera ici un petit afficheur OLED de 0,96 pouce de diagonale qui pose de réels problèmes d'utilisation. Un peu petit pour ma vision actuelle, il n'emportera probablement pas mon choix, et ce d'autant plus qu'existant en bicolore il est alors affecté d'une discontinuité entre les deux mosaïques de luminophores. Il reste toutefois concevable que certaines et certains d'entre vous disposent déjà d'un tel afficheur et ne veulent pas investir dans un autre modèle. C'est pour ces lectrices et ces lecteurs que je vais reprendre entièrement les routines d'affichage ainsi que l'architecture du programme.

➤ Rendre possible l'usage du petit afficheur OLED de 0,96' de diagonale.

Bien que beaucoup plus petit en taille d'écran, il consomme bien plus d'espace en mémoire de programme et surtout en mémoire dynamique. En effet, sa police de caractères est nettement plus étendue et contient les accentués ainsi qu'une palanqué de petits symboles dont nous n'avons que faire. Qui dit police de caractères dit tableau logé à la fois dans le programme, mais également sur le TAS. Du coup, à peine avons-nous rédigé les routines de calcul et celles pour afficher la PAGE_1 et la PAGE_2 que le programme se bloque par collision entre la PILE et le TAS. La seule façon de pouvoir loger tout ce petit monde dans les ressources de

l'ATmega328 consiste à ranger tous les textes en EEPROM et à abandonner certaines fonctions "secondaire". Dans ce but, les textes affichés pour le dialogue H/M sur USB ont été allégés au maximum. On libère ainsi de la place pour les textes des fonctions du Menu du GPS. Par ailleurs, pour simplifier le programme et éviter des doublons, toutes les initialisations seront effectuées dans le Dialogue USB dont le menu devient celui de la Fig.89 où les lettres minuscules ne sont plus précisées. Pour la bascule sonore c'est la lettre 'B' qui est choisie car le texte "BIP" est plus court que "Silencieux". Pour la saison et les unités, l'optimisation conduit à 'H' pour l'heure d'Hiver et 'K' pour les distances en Kilomètres. Comme pour BIP, chaque utilisation de 'H' ou de 'K' inverse l'option et de plus l'inscrit en EEPROM. Aussi, pour faciliter l'usage du petit appareil, l'état actuel enregistré en mémoire non volatile est précisé et repéré dans le cadre rose. Toutes ces optimisations ne sont pas suffisantes pour dégager suffisamment de place entre la PILE et le TAS. Pour libérer 34 octets on abandonne l'affichage de la vitesse et du cap qui n'ont toujours pas été validés et on supprime Tampon_VTG[34]. À ce stade du développement le Dialogue USB est entièrement développé et il reste 254 octets entre la PILE et le TAS ce qui pour le moment semble suffisant. (La suite confirmera ou infirmera, car pour l'Application n°4 on en a 280 pour assurer sa stabilité.) Pour ouvrir le menu du Dialogue USB on active l'IDE sur un croquis quelconque, on maintien cliqué le BPC, on active le Moniteur. On attend que la LED d'Arduino s'illumine et l'on libère le BPC. L'écran OLED présente l'affichage de la Fig.90 et la LED bleue s'illumine. La fenêtre contextuelle du Moniteur affiche la liste des commandes suivie de la liste des PI actuellement disponibles en EEPROM.

? : Ce MENU.
A : Ajouter un PI.
B : BIP.
C : Corriger un PI.
D : Déplacer un PI.
E : Effacer un PI.
H : Hiver OUI
K : Km OUI
L : Lister les PI.
P : PI sur RESET.
Q : Quitter.



Fig.90



Fig.91

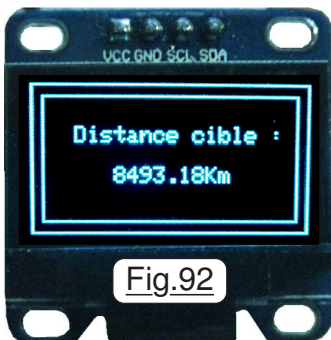
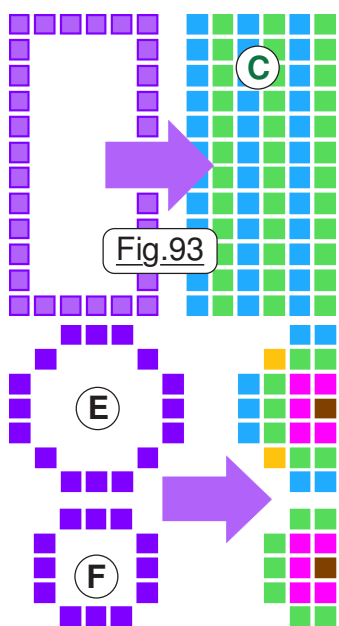


Fig.92

On peut alors utiliser toutes les commandes du menu du **Dialogue USB**. La commande '**q**' ramène au **Menu du GPS** qui présente comme montré sur la Fig.91 la date et l'heure. La rotation du codeur incrémental fait changer de PAGE-écran, la deuxième actuellement en place est celle de la Fig.92 indiquant la distance du GPS jusqu'à la Cible actuelle. On peut maintenant commencer à ajouter des fonctions au **Menu du GPS** et croiser les doigts pour la stabilité du programme.

➤ Particularités du petit afficheur de 0,96 pouce de diagonale.

Outre une police de caractère bien plus fournie que celle de son grand frère, (*Qui ici s'avère un inconvénient.*) la bibliothèque qui l'accompagne sait tracer des cercles et des rectangles, mais pas des disques et des rectangles pleins. Comme on désire utiliser les deux, on va devoir créer ces éléments géométriques particuliers. Par exemple avec les rectangles



disponibles, on obtient la Fig.94 **A** pour la représentation de l'état de réception des satellites. Pour retrouver la représentation en barres verticales **B** on ne fait plus appel au tracé des rectangle de la bibliothèque, mais à la technique **C** de la Fig.93 qui consiste à tracer six segments de droite verticaux les uns contre les autres et de hauteurs égales. Pour les cercles, l'approche est différente. Sur la

Fig.93 **E** et **F** sont représentés le grand et le petit cercle tels qu'ils sont tracés par la fonction **display.drawCircle** le moins grand ressemblant à un carré à cette échelle, vu la "pixellisation". On débute par un PIXEL au centre ici en marron entouré en rose par un carré. (*L'instruction étant celle d'un rectangle.*) Puis on trace le cercle vert pour le petit disque.

Pour le grand disque, on ajoute le carré vert suivi du cercle bleu clair. Dans la réalité, les quatre pixels oranges sont communs au deux dernières instructions. Ce n'est pas pénalisant puisque l'illumination des mosaïques réalise un OU logique si plusieurs tracés se superposent.

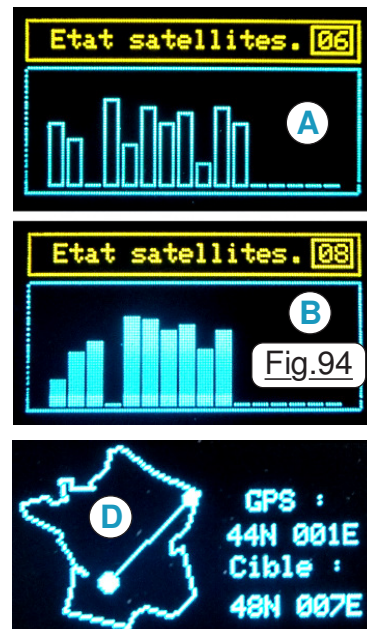
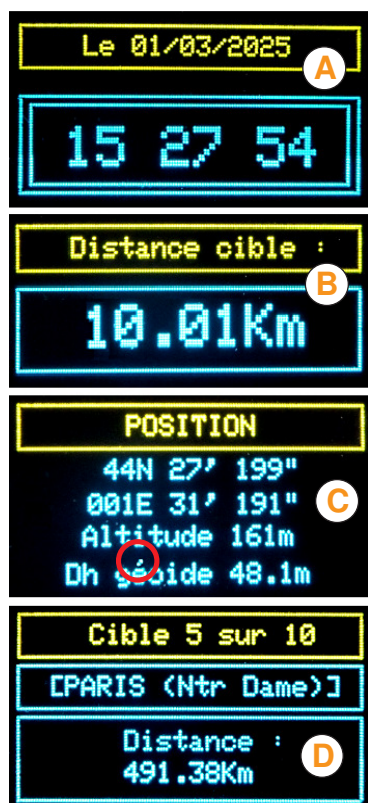


Fig.95

➤ Écran bicolore ou écran monochrome.



J'avoue que j'affectionne particulièrement le petit écran bicolore avec ses deux couleurs complémentaires. En monochrome il existe soit en bleu, soit en blanc. Tous présentent une définition identique de 64 x 128 luminophores. Les quatre exemples de la Fig.95 qui montre l'ordre croissant des affichages mettent bien en évidence dans le rectangle jaune un élément important des informations de la page affichée. En **A** on retrouve la date et l'heure légale. En **B** la distance qui sépare le GPS de la cible est centrée en largeur, c'est à dire que la marge de gauche est adaptée au nombre de chiffres indiquant la distance. En **C** l'accentué dans l'encadré rouge est obtenu artificiellement par deux PIXELs. Il aurait été facile en EEPROM de remplacer le '**e**' par le code affecté au '**é**' dans la police de caractères de l'afficheur OLED, mais vu qu'il me restait largement assez de place en zone dédiée au programme j'ai été paresseux et je n'ai pas cherché la valeur de ce code. Mon programme n'est donc pas optimisé à outrance, je suis sur ce point en contradiction avec mes "valeurs programmatiques". Enfin en **D** on retrouve l'interface Homme/Machine d'un cheminement qui utilise les dix cibles possibles. Actuellement on a indexé le cinquième jalon. Dans le chapitre précédent, dans les explications sur le **CHEMINEMENT** un point important a été oublié relatif à un "aléas" possible si on se trompe et que l'on désigne en premier jalon celui de la position actuelle du GPS.

Dans un cheminement désigner le lieu actuel du GPG comme premier jalon engendrera automatiquement le saut à celui d'arrivée à l'activation de la fonction. Le premier point où l'on se trouve a forcément été choisi par erreur car ce n'est pas une cible à atteindre. Au lieu de sortir de la fonction puis d'y revenir pour réactualiser **ARRIVEE** et **DEBUT**, il suffit de tourner le codeur incrémental dans le sens antihoraire pour revenir à celui désiré qui en principe est le deuxième de la liste désignée dans l'initialisation de la fonction.

Manifestement, chaque fois que j'ai opté pour ce petit afficheur dans mes applications informaticoélectroniques, j'ai préféré esthétiquement le bicolore. Il se trouve toutefois un cas particulier où ce modèle présente un inconvénient, c'est le cas où dans l'application on désire afficher un graphe qui va utiliser les 64 PIXELs de hauteur. En effet, le modèle bicolore présente la même définition que son homologue monochrome, avec toutefois une ligne de séparation entre les mosaïques jaunes et les luminophores bleus. Il en résulte alors une discontinuité dans le dessin graphique que montre bien la Fig.96 qui montre la seule page de tout le programme où ce petit inconvénient se fait sentir. Pour toutes les autres pages d'affichage j'ai tenu compte de la présence de ces deux zones, et le logiciel est totalement compatible avec les deux références. Au final, cette discontinuité n'est pas vraiment pénalisante et quel que soit le modèle qui est à votre disposition l'ensemble reste très séduisant. Toutefois, entièrement compatible ne signifie pas forcément idéal. En effet, le développement s'est effectué sur le bicolore qui concrètement possède deux zones distinctes séparées par une ligne non active. Le monochrome ne présente qu'une seule surface, du coup ce qui était jaune et ce qui était bleu n'est



Fig.97 plus séparé. On observe bien sur la Fig.97 que le texte sous le rectangle est plus proche. Aussi, pour celles et ceux qui possèdent le modèle monochrome, il sera esthétiquement favorable de revoir certains cadrages. Ce n'est absolument pas impératif, mais c'est un bon exercice cérébral.

➤ Compte-rendu impératif.

Commencer par noter que le programme ne fonctionnera correctement que si l'EEPROM contient les nouveaux textes adaptés et épurés. Il importe dans ce but de **téléverser** l'outil nommé **Ecrire_en_EEPROM_3.ino** qui en outre gère l'EEPROM avec dix PI pour nous aider à valider le programme **Application_GPS_5.ino** spécifique au petit afficheur OLED de 0,96 pouce de diagonale. Bien que la taille de disponible entre la **PILE** et le **TAS** ne fasse plus que 243 OCTETS, le programme s'avère stable. Dans ce dernier, j'ai abandonnée l'idée d'afficher le planisphère. Cet échec résulte de l'impossibilité que j'ai rencontré à obliger les tables des cartes à être des constantes logées dans la zone programme par l'artifice **PROGMEM**. Mes essais dans cette optique se sont avérés infructueux, et je n'ai pas trouvé pourquoi la lecture du tableau ainsi figé fournit des valeurs incorrectes. Le petit outil **Echec_de_PROGMEM.ino** permet de mettre en évidence le problème. Si un internaute en trouve la cause qu'il le fasse savoir à la communauté ...

Les **Fiches n°22** et **n°23** listent les textes actuellement logés en EEPROM. Il reste encore quarante emplacements pour modifier ou ajouter du texte dans la mémoire non volatile de l'ATmega328. Par exemple on peut remettre les minuscules dans la liste des commandes du dialogue USB etc. Par ailleurs, le programme actuel ne consomme que 87% de l'espace réservé pour le programme. Il est donc possible d'ajouter pas mal de code objet si on le désire pour améliorer le logiciel que je vous propose. Par exemple sur la page de la Fig.97 remplacer le 'D' de "Dh" par le caractère delta (*Soit la lettre Δ*) qui serait plus appropriée. Il sera facile de la construire avec trois droites pour former le triangle. C'est tout l'avantage d'un appareil "fait maison" que l'on peut améliorer ou modifier à sa guise, largement de quoi combler les longues soirées d'hiver ...

09) Sixième et septième application autonome.

Comme pour **Application_GPS_4.ino** avec **Application_GPS_6.ino** on reprend strictement les mêmes fonctions. Le comportement du programme est strictement identique. La seule modification consiste à changer un peu l'organisation des Entrées/Sorties pour faciliter la réalisation du circuit imprimé. Pour mémoire, la dilution n'est plus indiquée. Le cheminement simple est sans indication de la distance qui reste à parcourir. Le dialogue USB permet avec 'g' d'effacer tous les PI sauf un, avec une seule commande. C'est à ce stade qu'est réalisé le circuit imprimé pour rendre vraiment autonome et surtout "mobile" le petit instrument. Toutefois, le chapitre de réalisation du coffret est décrit plus loin, car je préfère regrouper ceux concernant les dernières versions logicielles.

➤ Version logicielle avec mode cheminement amélioré.

Cette nouvelle proposition ne remet absolument pas en cause le matériel. Électronique terminée et intégrée dans un coffret qui en fait un appareil autonome, on peut à tout moment changer le programme car la mini-prise USB qui permet de téléverser les croquis est directement accessible. C'est du reste un impératif pour effectuer le dialogue Homme / machine de gestion des PI en EEPROM. Cette version nommée **Application_GPS_7.ino** n'est qu'une variante de la précédente qui a pour motivation une idée d'utilisation qui a émergé juste avant de concrétiser le boîtier. Quand on utilise le **mode Cheminement**, la seule information affichée est la distance qui nous sépare "à vol d'oiseau" du prochain point de passage. Il semble séduisant de remplacer cette approche par une **fonction qui indique aussi la distance totale qui reste à couvrir** en supposant que la route entre chaque point de passage est une

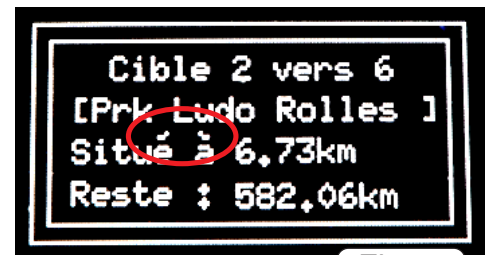
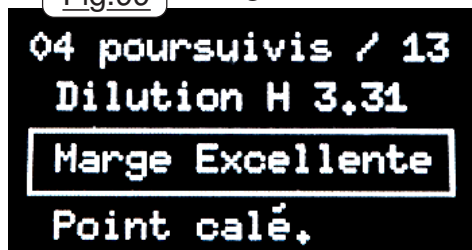


Fig.98

loxodromie, car il faut bien reconnaître que la distance à vol d'oiseau est certes une information souvent étonnante, mais moins intéressante que la distance qu'il faut encore franchir pour arriver à destination. Bien que calculer une distance totale en ajoutant la longueur de divers segment puisse sembler élémentaire, dans la pratique c'est assez compliqué car on se heurte à une succession de formats dont il faut gérer les spécificités, ces adaptations générant une boulimie en octets de programme. Pour arriver à loger le code spécifique au calcul des distances, il est devenu impératif de dégager de la place en zone réservée au programme. Du coup **j'ai sacrifié** l'une des fonctions que je préfère, mais qui reste assez secondaire vu que peu d'internautes se rendront à des distances qui justifient **le planisphère** à l'affichage. Ce n'est pas tragique, car **Application_6** et **Application_7** ne sont différentes que par leur croquis, les données en EEPROM restant inchangées. On peut à tout moment téléverser l'une ou l'autre à notre convenance. Sur la Fig.98 actuellement on va vers la **Cible n°2** en EEPROM et on en est éloigné de 6,73km. Le trajet complet comporte trois villes de passage pour aller jusqu'à la destination qui en EEPROM est le PI n°6. Noter dans l'encercle les deux accentués réalisés avec trois PIXELs pour chaque élément. Comme la circulation routière nous oblige à emprunter des trajectoires sinueuses et que les longueurs calculées entre les segments sont des loxodromies, pour afficher des valeurs plus proches de la réalité, les nombres calculés sont majorés avec un coefficient de 1.28 évalué expérimentalement. Pour pouvoir éventuellement l'affiner suite à de plus amples expérimentations en situation réelle, il suffira de modifier la directive **#define Correction_distance 1.28** située en tête de programme. Si dans les options nous avons choisi le **Mn** au lieu du **Km**, c'est que le mobile est maritime ou un aéronef. Dans les deux cas on navigue en

Fig.99



"ligne droite" et le coefficient de correction n'est pas appliqué par l'algorithme. Dans le programme, pour effectuer les calculs utiles à **Affiche_la_PAGE_7()** on utilise les coordonnées en EEPROM qui sont exprimées en degrés. On doit transformer les décimales des degrés en minutes d'angles. La chaîne de caractères est transformée en un **float** sur lequel on applique le coefficient 60. Puis il faut le reconvertir en chaîne de caractères pour loger l'équivalent de ce nombre en **Tampon_GGA[]**. Il serait possible d'utiliser la fonction **String** pour effectuer cette traduction. Si vous allez décortiquer les séquences de calcul vous constaterez que je n'ai pas utilisé cette facilité et codé par un algorithme personnel. Dans la pratique, j'ai éliminé l'option **String** car un seul appel à cette dernière

consomme 1684 octets. Grâce à toutes les optimisations réalisées, le programme actuel ne consomme que 30522 octets. Au final il nous reste encore 198 octets pour modifier ou corriger le logiciel. Profitant du fait qu'il restait de la place disponible en mémoire, de petits détails esthétiques ont été apportés comme en Fig.99 l'encadrement de la Marge ou en `Affiche_la_PAGE_3()` la valeur de la vitesse sol en double taille. **PILE - TAS** dépasse les 650 octets : Donc le programme sera stable.

➤ **Format Google ou position en degrés, minutes et secondes d'angle.**

Presque au bout, c'est l'avant dernière version du programme. Pour valider les fonctions de positionnement, on utilise la cartographie de **Google Maps** qui nous le savons est forcément entachée d'une imprécision inhérente à la précision du point sur un écran numérique. Pour le plaisir, lorsque je suis arrivé à destination, je note la position indiquée par le GPS qui est forcément plus précise. L'idée consiste alors à utiliser les valeurs GPS pour remplacer celle un peu approximatives en EEPROM. Le hiatus, c'est que notre GPS donne la position en degrés, minutes et secondes, alors qu'en EEPROM les coordonnées sont en degrés et ses décimales. Aussi, pour nous éviter d'avoir à transposer, les calculs à effectuer n'étant pas toujours évidents, l'idée consiste à ajouter dans les initialisations sur RESET une option "**Google**" sous laquelle, le format d'affichage pourra être celui de la Fig.71 **E** en page 38 ou un affichage de position au format adopté sur **Google Maps**. Sur la Fig.100 et la Fig.101 qui montre les deux options quand on tourne le codeur incrémental, on constate qu'au final c'est l'écran

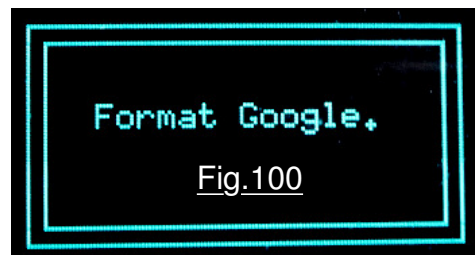


Fig.100



Fig.101

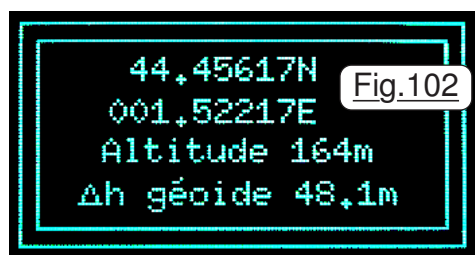


Fig.102

monochrome bleu qui équipe la version définitive dans le coffret. Comme pour les autres options, lorsque le choix que nous voulons privilégier est affiché sur OLED, en cliquant sur le **BPC** on valide l'option, et comme c'est la dernière dans la liste, on entre dans le **Menu de base** du

GPS. Naturellement l'option est inscrite en EEPROM et rechargée de façon automatique lors de la mise sous tension de l'appareil. La Fig.102 qui est à comparer avec la Fig.71 **E** présente l'affichage au format "Degrés angulaires" et décimales. Ce sont exactement ces valeurs qu'il faut proposer en EEPROM pour définir un point d'intérêt. Seuls les chiffres sont indiqués dans la ligne de saisie du moniteur, sans le point décimal. Sur la Fig.103 dans les encadrés jaunes **A** et **B** on observe que le choix effectué sur RESET est ajouté dans la page des options sur la même ligne que celle qui précise les unités pour indiquer les distances. Pour pouvoir indiquer cette option, il a été indispensable de libérer de la place dans l'espace réservé au code objet. C'est la fonction '**g**' du dialogue USB qui a été sacrifiée.

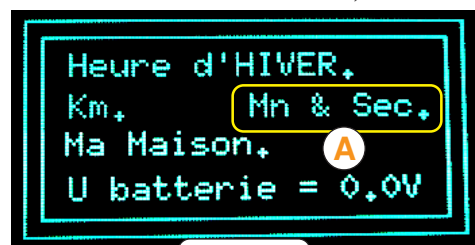
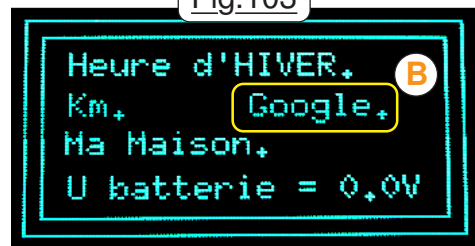


Fig.103



Si on désire tout effacer (*Sauf un PI nous le savons.*) il faudra enchaîner neuf fois la fonction '**e**' ce qui n'est pas tragique du tout. À ce stade du développement, le programme consomme 30702 octets sur 30720 de disponibles, autant dire que l'ATméga328 est bien rentabilisé. Il est peu probable que les 18 octets qui restent puissent être suffisants pour corriger une éventuelle vermine, il serait alors indispensable de générer de la place. Il suffirait d'enlever ici ou là un complément "de luxe" tel qu'un accentué généré par pixels par exemple. Bref, c'est la vie banale du programmeur fut-il de loisir. L'espace entre la **PILE** et le **TAS** est de 633 octets à l'ouverture du dialogue avec le **Moniteur USB**. En faisant appel aux diverses fonctions de ce dernier suivi par '**m**' on constate que l'intervalle peut diminuer de quelques octets, mais relativement peu. Aussi, avec un espace de plus de 600 octets je peux vous assurer de la stabilité du programme. S'il advenait qu'il présente un comportement imprévu, c'est dans les algorithmes qu'il faudrait alors chercher la source du problème.

Enfin, dans l'EEPROM il reste encore un peu de place pour logger des textes. Les utiliser obligerait à modifier un peu [Ecrire_en_EEPROM_3.ino](#) mais reste un moyen aisé pour diminuer d'environ 39 octets la taille du programme ... c'est déjà ça de gagné. Pour pouvoir mettre au point la page qui affiche la vitesse et le cap, terminer la réalisation matérielle pour faire du dispositif un appareil autonome est indispensable. Le bout du tunnel est en vue !

10) Avant dernière application autonome.

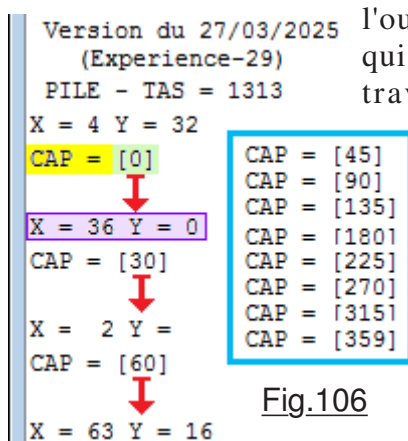
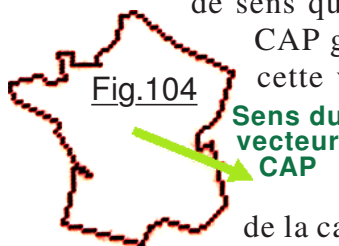
C'est promis, [Application_GPS_8.ino](#) sera l'avant dernière variante proposée dans le didacticiel vous laissant l'initiative d'en développer par vous-même pour adapter le dispositif à vos désirs personnels. Il ne s'agit pas d'un bouleversement, à peine un petit plus, sorte de cerise sur le gâteau. Quand on examine la page-écran qui indique la vitesse et le CAP, données qui n'ont

de sens que lorsque l'on est dans un mobile, force est de constater que la valeur du CAP géographique bien que "banale" reste un peu confuse à visualiser. L'idée de cette variante consiste à profiter de l'affichage graphique de la carte de France pour y superposer un rayon vecteur qui indique dans quelle direction on se déplace par rapport au nord géographique. Il semble logique, comme précisé sur la Fig.104 de placer l'origine du vecteur au centre de la carte. Sur l'écran graphique photographié sur la Fig.105

le rayon vecteur est un simple trait sans indication du sens. Ce dernier sera donc conventionnellement dirigé du centre de la carte vers l'extérieur. Dès que le GPS ne se déplace plus assez rapidement pour pouvoir déterminer la valeur de la vitesse et celle du CAP, le segment de droite n'est plus tracé et le texte **CAP = nnn** en bas à droite n'est plus précisé. En revanche, lorsque ces données sont cohérentes, l'affichage est réalisé et le segment de droite s'ajoutera à celui du tracé de la loxodromie vers la **Cible** si l'option est validée. Tracer sur un écran graphique une "aiguille" qui tourne autour d'un point fixe en fonction d'un angle donné n'est pas si évident que nous pourrions le penser. D'une part le logiciel doit déterminer la valeur de l'angle, d'autre part on doit faire appel à la trigonométrie de base. Ce n'est pas spécialement compliqué, mais **sinus** et **cosinus** sont des entités qui présentent un signe positif ou négatif en fonction des valeurs de l'angle. Angle qui doit être exprimé en radians ne

l'oublions pas. Aussi, pour mettre au point la procédure et les instructions qui seront chargées de tracer le vecteur CAP, personnellement je préfère travailler sur un programme allégé spécial qui sert d'outil. C'est

[Experience_29.ino](#) qui est créée pour faciliter le travail de développement. Ce démonstrateur est simplifié au maximum. Il se contente d'afficher le rayon vecteur en fonction de la valeur de l'angle que l'on fournit par dialogue avec le **Moniteur** de l'**IDE**. Naturellement les angles sont à fournir en degrés et pas en radians ! Quand la fenêtre contextuelle du **Moniteur** s'ouvre, le programme se présente et affiche le texte **CAP** qui sur la Fig.106 est colorié en jaune. La valeur frappée dans la ligne de saisie est alors affichée entre crochets dans la zone verte. Si cette valeur est supérieure à 359 degrés ou inférieure à zéro, un BIP d'alerte est déclenché. Si l'angle est correct, les coordonnées du trait de représentation de la direction sont calculées et le rayon vecteur est tracé sur la carte de France. Juste en dessous de la ligne "CAP =" décalé par le symbole de la flèche rouge, les coordonnées de l'extrémité du segment indiquant la direction suivie sont précisées. On peut ainsi vérifier ce que donne le logiciel et calculer à la main ce que l'on doit trouver. Outre les [Fiches A5.pdf](#) qui accompagnent le didacticiel se trouve le dossier [Fiches personnelles.pdf](#) servant de complément. Les premières servent à documenter certains points du didacticiel, et personnellement je n'en est pas l'usage. Les seconde sont relatives aux points techniques qui sur le long terme seront utiles pour la maintenance. En particulier y figurent quelques points "mathématiques" conduisant au développement logiciel qui pourront s'avérer utile lors de reprise du programme. En particulier la [Fiche n°10](#) et la [Fiche n°20](#) sont relatives au tracé du vecteur CAP et illustrent en complément ce chapitre. Enfin, au format A6 pour être "aux dimensions du petit appareil" et l'accompagnant "partout", on trouve résumé dans le fichier [UTILISER.pdf](#) tout ce qu'il faut pour se servir du petit GPS et en assurer la maintenance tant électronique qu'informatique. Le moment est enfin venu de concrétiser le petit boîtier. **Page 52**



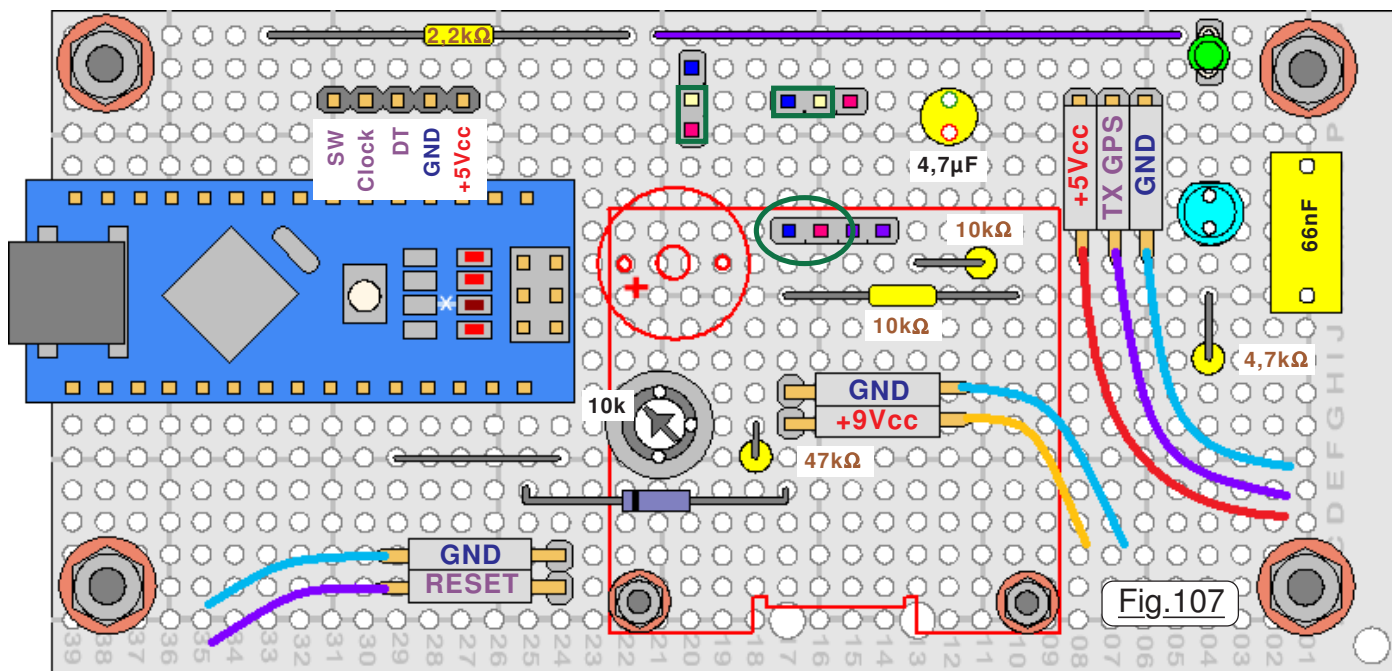
11) La réalisation pratique du circuit imprimé.

Logiciel entièrement développé, (Ou presque car pour la vitesse et le cap il sera toujours temps de modifier le programme ultérieurement.) le moment est venu de concrétiser l'appareil en développant un circuit imprimé pour assembler l'électronique et en réalisant un coffret pour en faire un instrument utilisable digne de ce nom. En ce qui me concerne, mon choix se porte sur l'Application n°4 qui fait usage d'un codeur incrémental et d'un afficheur OLED de 1,3 pouce de diagonale. Si vous optez pour l'une des autres versions, il faudra vous inspirer de mes études pour adapter le circuit imprimé. Les contraintes de conception sont les suivantes :

- Coffret aussi petit que possible.
- Possibilité de modifier le logiciel sans avoir à ouvrir le boîtier. (La mini prise USB devra être facilement accessible, elle permettra éventuellement d'alimenter l'ensemble par le secteur.)
- Alimentation par un accumulateur 9v rechargeable directement par une mini prise USB intégrée dans son boîtier et accessible de l'extérieur.
- RESET sur un bouton poussoir extérieur pour disponibilité permanente.

➤ Le dessin du circuit imprimé.

Fidèle à une technique qui évite d'avoir des possibilités de gravure d'une plaque cuivrée et de la percer pour insérer chaque composant, j'utilise systématiquement des plaquette du commerce directement utilisable pour élaborer des prototypes. Il suffit de les tailler à la bonne dimension, éventuellement d'ajouter ou agrandir des trous pour la traversée de la visserie et le module électronique est terminé. Par exemple, pour réaliser le circuit de la Fig.108 j'ai consommé entre cinq heures et six heures maximum. C'est une affaire d'habitude. La Fig.107 épurée présente



l'implantation des composants et surtout précise le branchement des divers connecteurs HE14. Prendre garde à la position des mini ponts représentés en vert. Leurs positions correspondent aux broches de l'afficheur OLED utilisé dont la polarité est précisée dans l'encerclé vert. Si votre exemplaire est inversé, il suffit de déplacer les deux mini ponts sur l'autre position de chaque petit connecteur HE14. Comme précisé sur le schéma de la [Fiche n°21](#) la résistance de limitation **R** pour chaque diode électroluminescente devra être adaptée à son rendement pour obtenir un éclairage correct. Il est évident que j'ai choisi des diamètres et des couleurs en fonction de mes critères esthétiques. Vous pouvez naturellement adapter à vos désirs. Le dessin de la Fig.108 présente le circuit imprimé vu coté composants, alors que sur la Fig.109 nous avons l'autre face vu coté cuivre. Cette plaquette est taillée dans un circuit plus grand facile à se procurer sur internet. À titre indicatif et sans en tirer un bénéfice personnel quelconque, pour ma part je les ai approvisionné sur :

https://www.amazon.fr/dp/B07H3RTFFG?ref=ppx_yo2ov_dt_b_fed_asin_title&th=1

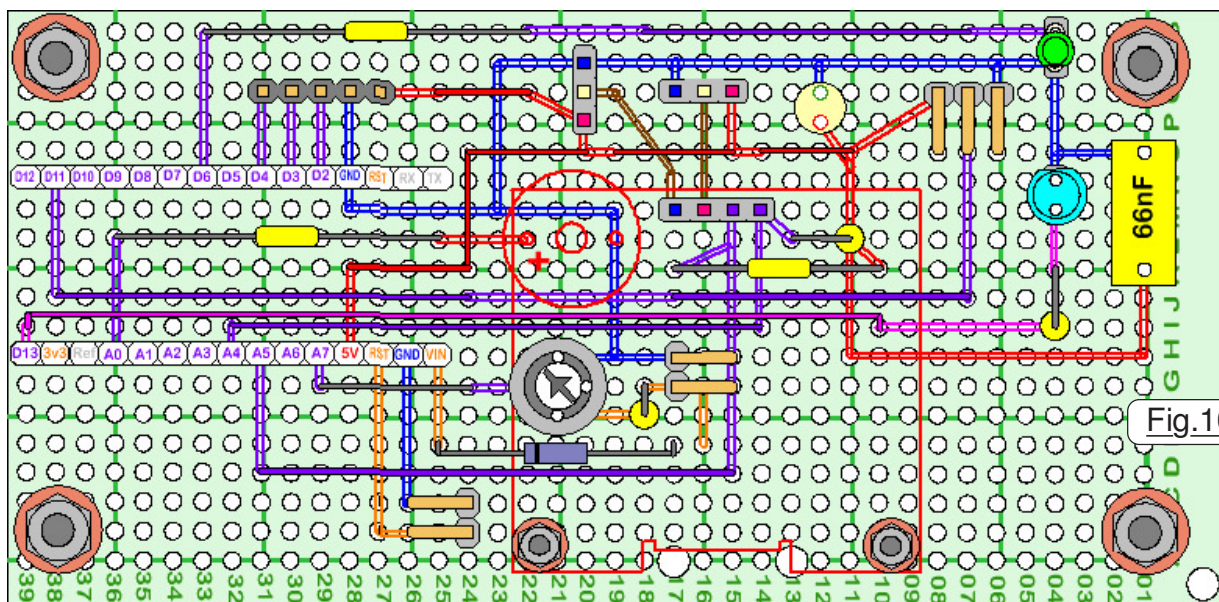


Fig.108

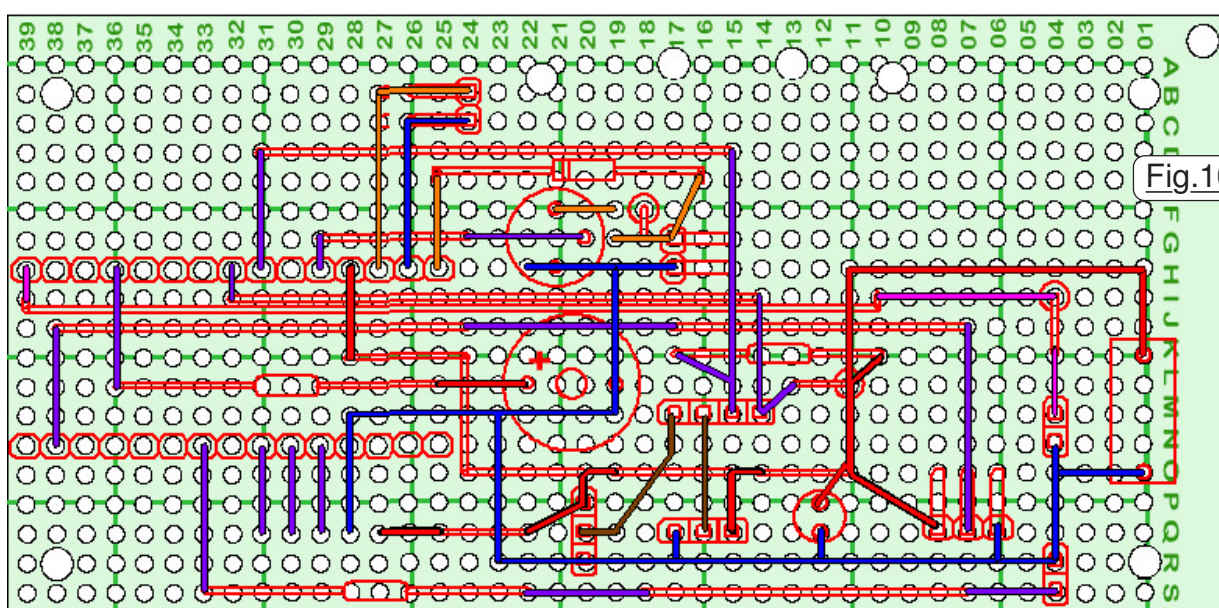
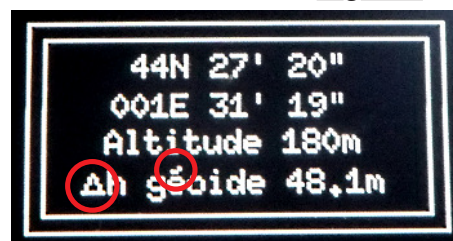


Fig.109

➤ Réimplantation de deux broches.

Fig.110

P our faciliter l'implantation des composants sur le circuit imprimé, on peut constater que maintenant la LED d'Arduino **D13** pilote la LED bleue de 5mm de diamètre. Quand à l'ancienne sortie **D12** devenue inutilisée, maintenant c'est **D6** qui pilote une petite LED verte qui clignote "en boucle de base. Le schéma de la **Fiche n°21** a été mis à jour. Du coup, le programme a été entièrement remanié pour tenir compte de ces changements et d'en améliorer certains détails opérationnels. C'est **Application_GPS_6.ino** qui sera le logiciel à téléverser dans Arduino pour bénéficier de la dernière version adaptée au circuit imprimé. Divers correctifs ont imposé de gagner un peu de place en mémoire de programme. Ces modifications sont consignées en tête du listage du programme. Du coup, comme il restait un peu de place le '**D**' de "**Dh**" a été remplacé par la lettre Delta montré dans l'encadré rouge de la Fig.110 avec en prime l'accent sur le '**é**' de **géoïde**. Noter au passage que suite à des aléas expérimentaux, un moment je me suis fourvoyé. Le programme ne démarrait pas correctement sur le circuit imprimé. J'ai initialement pensé que ça résultait de la configuration de la platine expérimentale sur laquelle j'avais installé deux condensateurs de découplage sur l'alimentation. Je les ai donc ajoutés sur le circuit imprimé. (Un de 66nF et un de 4,7µF polarisé.) Et bien le problème venait en réalité du fait que j'avais mal branché le module GPS. **Conclusion : Ces deux condensateurs ne sont pas du tout indispensables, vous pouvez ne pas les implanter sur votre exemplaire.**



➤ Réalisation pratique du circuit imprimé.

Rechercher un maximum de compacité pour l'appareil engendre un réel problème d'intégration des divers composants. Comme vous allez le constater sur les photographies, on aboutit à un tau d'imbrication peu courant. Il faut en outre que l'assemblage et le désassemblage dans le coffret ne soit pas trop galère. Des compromis sont délicats à concilier et imposent une étude préalable sérieuse. Sur [IMAGE 08.JPG](#) la plaquette expérimentale est percée des divers trous devant être traversés par de la visserie alors que sur [IMAGE 09.JPG](#) les composants les moins hauts sont en place. C'est sur [IMAGE 10.JPG](#) que les connecteurs les plus hauts ont été ajoutés. On remarque que le but de ces HE14 dont les broches sont relativement longues consiste à surélever l'écran OLED et à placer les LED pour qu'elles puissent traverser le dessus du coffret. On

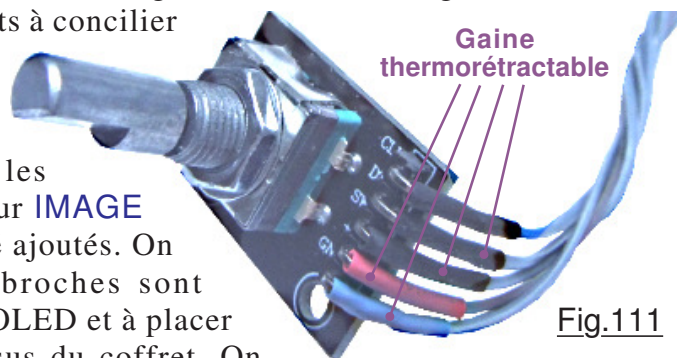


Fig.111

Fig.112



continue avec [IMAGE 11.JPG](#) à ajouter les fils de liaison qui remplacent les pistes cuivrées. Puis, on prépare la ligne qui relie le codeur incrémental Fig.111 ou [IMAGE 12.JPG](#) au connecteur HE14 mâle à cinq broches du circuit imprimé. Les deux extrémités de la ligne sont consolidées mécaniquement avec de la gaine thermorétractable qui si elle est en couleur permet de repérer le sens de branchement de la fiche. Même traitement en Fig.112 pour le bouton poussoir extérieur du RESET qui prend la forme d'un micro-Switch à galet. Ce type de composant

présente l'avantage de pouvoir facilement être assemblé sur le circuit imprimé. Sur [IMAGE 13.JPG](#) on constate que le micro-Switch est placé entre les deux vis qui soutiennent l'afficheur OLED. Au préalable sa ligne a été préparée Fig.112 avec gaine thermorétractable sur le petit connecteur HE14. Après avoir vérifié, vérifié et encore vérifié le câblage du circuit imprimé, on met en place les deux petits ponts de polarisation de l'afficheur, on branche le module GPS et le codeur incrémental. On réunit provisoirement l'afficheur à son support mais il reste écarté du circuit imprimé pour accéder à l'ajustable de **10kΩ**. On branche le

petit accumulateur rechargeable sur le connecteur Ouf, après une seconde d'initialisations l'écran affiche la page de caractéristiques des satellites. Tournant le codeur incrémental on vérifie que l'effet obtenu est bien dans le sens

attendu. Si ce n'est pas le cas il faut croiser **DT** et **CLK**. Tourner le codeur pour faire afficher le résumé des options. Ajuster alors le petit potentiomètre d'[IMAGE 14A.JPG](#) pour calibrer la mesure de tension. Dans ce but un voltmètre mesure la tension actuelle de l'accumulateur. On ajuste alors l'affichage à cette valeur. **ATTENTION : Ne pas brancher l'accumulateur 9v sur la carte Arduino si elle est déjà alimentée par sa prise mini-USB.** On va alors pouvoir installer l'afficheur OLED sur le circuit imprimé qui sur [IMAGE 14B.JPG](#) est entièrement terminé. Mais avant il faut préparer le petit carton d'[IMAGE 15.JPG](#) que l'on immobilise sur le petit écran avec les deux vis ϕ M2 un peu longues (25mm sous tête) en intercalant deux petits manchons pour donner de la souplesse. (Voir [IMAGE 16.JPG](#) et [IMAGE 17.JPG](#).) Si vous ne possédez pas ce genre d'accessoires une simple rondelle plate conviendra. Au final, sur d'innombrables essais avec les trois modèles de modules GPS présentés sur [IMAGE 01.JPG](#) je n'ai pas été en mesure de déterminer quel est le plus sensible, tous semblent équivalents. Du coup, j'ai opté pour celui en **A** car il présente une liaison plus longue que celui en **B** et une antenne plus large que celui en **C**. L'expérience montre que l'antenne placée à plat sous l'afficheur OLED fonctionne correctement. Plaçant ainsi le petit bloc de céramique, et le circuit imprimé verticalement à coté du circuit imprimé principal, on aboutit à la compacité désirée. Le petit carton d'[IMAGE 15.JPG](#) constitue une alvéole dans laquelle est simplement glissée l'antenne. Comme le montre [IMAGE 18.JPG](#), [IMAGE 19.JPG](#) et [IMAGE 20.JPG](#) le volume au dessus du circuit imprimé est utilisé à outrance. L'électronique étant globalement terminée, le moment est venu dans le planning de passer à la réalisation du boîtier.

12) L'agencement du coffret.

C'est à la fois la pierre d'achoppement, car ce n'est pas facile pour tout un chacun d'avoir le matériel et le savoir faire pour réaliser un coffret personnel, mais c'est aussi la cerise sur le gâteau, car on aboutit à exactement ce que l'on désire, et c'est toujours valorisant de se dire : Ce petit appareil, je l'ai façonné entièrement de mes mains, je l'ai vraiment mérité. Dans ce chapitre nous allons passer en revue pas à pas la façon dont j'ai réalisé le mien. Vous pouvez constater sur la Fig.1 et sur les photographies d'[IMAGE 40.JPG](#) à [IMAGE 42.JPG](#), qu'avec l'utilisation du **polystyrène choc**, on peut aboutir à des coffrets très présentables et exactement aux proportions et aux dimensions que l'on désire. C'est l'immense avantage du "fait main". J'utilise ce matériau depuis très longtemps, car il se façonne parfaitement, se colle avec aisance et nécessite très peu d'outillage pour le mettre en œuvre. Dans le dossier <DOCUMENTS> est proposé [Réalisation des coffrets.pdf](#) qui vous dévoile les techniques que j'utilise depuis des années. Par rapport à ce document, maintenant je simplifie un peu en substituant les écrous prisonniers en taraudant directement les "quadruples" épaisseur à ϕ M3 ce que montrent [IMAGE 25.JPG](#), [IMAGE 26.JPG](#) et [IMAGE 29.JPG](#).

➤ **Le modèle informatique.**

Bien que pour concrétiser mon boîtier je me suis contenté de deux ou trois dessins tracés à la main avec précision, lorsque mon coffret a été entièrement terminé, l'ensemble assemblé et validé, j'ai décidé de vous tortiller un équivalent virtuel sur un modèleur 3D pour pouvoir vous détailler son agencement et surtout vous fournir quelques dessins cotés. Il sera alors plus simple pour vous d'établir un parallèle entre la réalité et les 22 photographies commentées.

Globalement, le boîtier présente l'aspect de la Fig.113 sauf que constitué de plaques de polystyrène choc blanches de 3mm d'épaisseur, mon exemplaire n'est pas peint. Du reste si vous le désirez, rien n'interdit de le faire. J'ai déjà utilisé des bombes aérosol de peinture pour automobiles, le rendu est parfait et vous avez le choix des couleurs. Sur la Fig.114 la en marron. En 1 et 3 en vert on trouve avant 2 coté usager est en violet alors que la En 6 les plaquettes jaunes sont des renforts d'angle.

Enfin en 7 et 8 se trouvent en rose pastel les traverses avec les trous taraudés à ϕ M3 sur lesquelles on immobilise le couvercle. Sur les dessins et la cotation, ces traverses sont réalisées avec quatre plaques collées face contre face la hauteur étant alors de 12mm. À l'usage une telle longueur pour les trous taraudés est un peu inutile. Je vous conseille de n'utiliser que trois couches, la hauteur de 9mm sera largement assez importante.

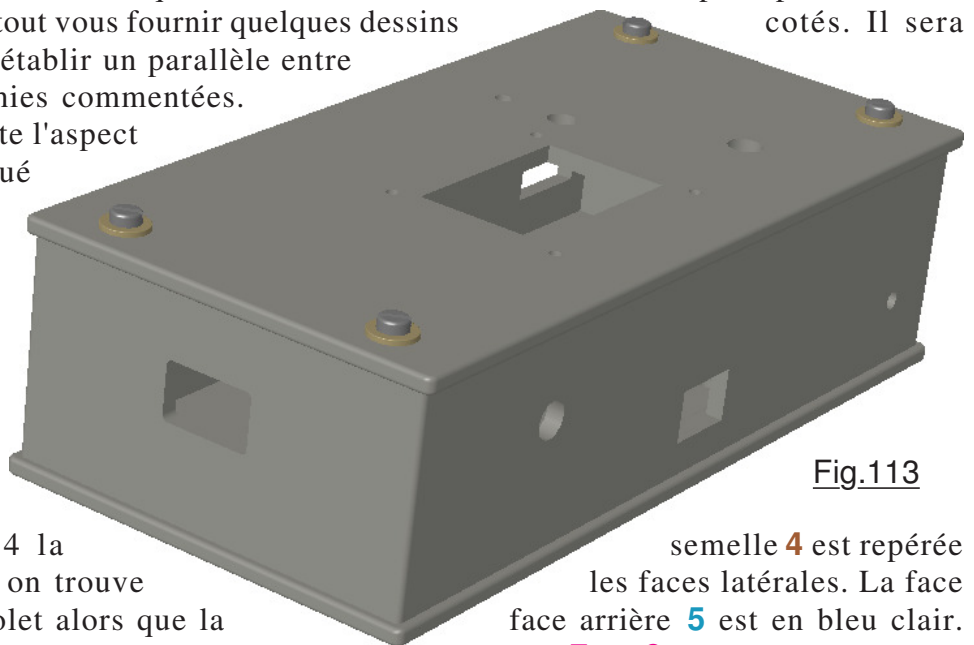


Fig.113

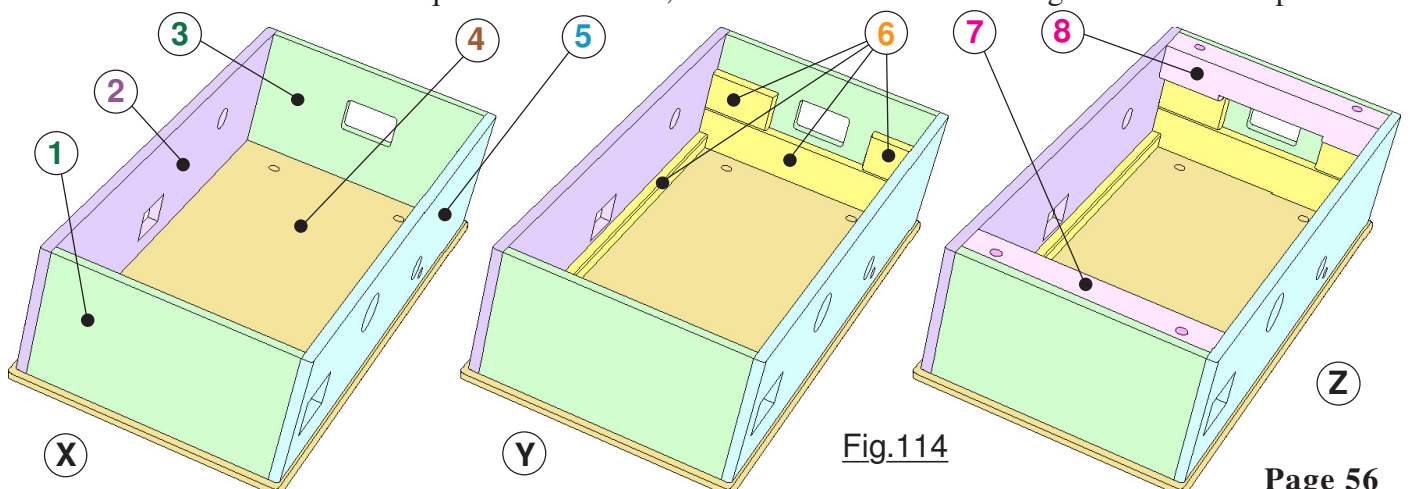
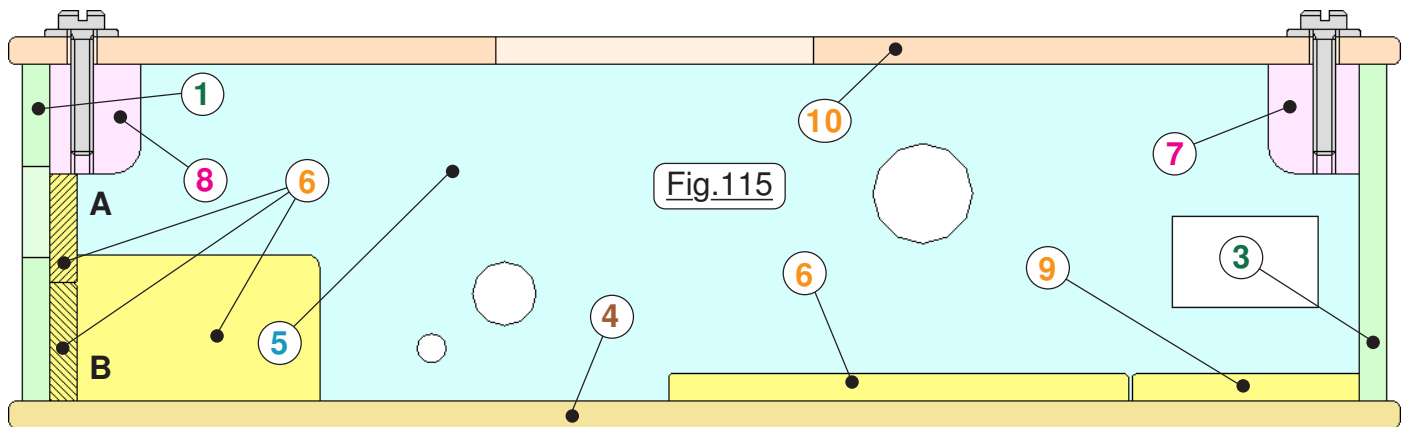


Fig.114



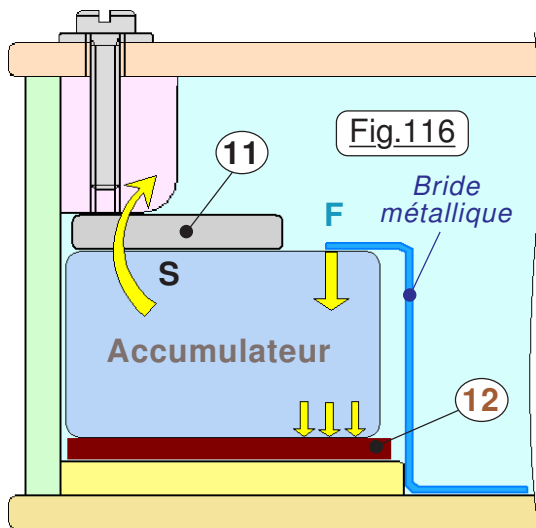
La Fig.115 propose une coupe médiane transversale du boîtier avec son couvercle 10. Le dessin respecte les couleurs des éléments de la Fig.111 le dessus étant en orange pastel. Il reprend les mêmes repères. Le renfort d'angle 9 visible en IMAGE 31.JPG et IMAGE 27.JPG consolide le boîtier *mais sert surtout à surélever de 3mm le petit accumulateur rechargeable* de 9Vcc. Le renfort 6 hachuré en B consolide l'ensemble. Les deux renforts d'angle A également hachurés sur la Fig.112 montrés sur IMAGE 28.JPG servent également à positionner en hauteur la traverse 8 au moment de son collage sur 1. Sur IMAGE 26.JPG on observe facilement les renforts d'angles 6 et 9.

➤ Réalisation pratique du boîtier principal.

Trois étapes présentées sur la Fig.111 permettent de concrétiser le petit coffret. La première en X consiste à créer et assembler la semelle et les quatre faces latérales. *Naturellement tous les trous de passage de la visserie et les diverses lumières sont réalisés avant le collage des plaques* entre elles et sur la semelle. Puis on prépare tous les renforts d'angles 6 et en Y ils sont "soudés" sur le boîtier qui ainsi présente alors une résistance mécanique à toute épreuve. Enfin on crée les deux traverses 7 et 8 que l'on ajoute en Z. Quand le collage est "durci" on perce alors et on taraude à ϕ M3 les quatre trous d'immobilisation du couvercle. La première phase de cette entreprise consiste à découper, percer et réaliser les ouvertures sur les quatre faces latérales ce que montre Image 21.JPG. Les dessins des diverses pièces avec les cotations sont disponibles dans DESSINS du coffret.pdf rangé dans le dossier <DOCUMENTS>. La photographie au zoom d'Image 21.JPG dévoile le petit usinage de dégagement pour la palette du micro Switch. Puis les cinq éléments étant usinés et validés, on procède sur Image 23.JPG et sur Image 24.JPG à leur collage au diluant cellulosique. (*Concrètement le matériau "fond" et se soude*) Puis, on réalise les traverses. Sur Image 25.JPG la pièce est dégagée au centre pour ne pas obstruer sur le haut la lumière qui sert à brancher la ligne USB sur la carte Arduino. Les traverses sur le prototype sont constituées de quatre plaquettes superposées. Mais comme déjà signalé, trois sont largement suffisantes, du coup le dégagement central pour 8 n'est plus à faire. Le boîtier proprement dit est terminé. On façonne alors le dessus. Pour arriver à percer les quatre trous de fixation exactement au dessus des trous taraudés, on commence par placer une feuille de papier comme "couvercle". Puis avec une quelconque aiguille, on perce les trous un à un. Dès qu'un trou est repéré, on visse le papier. Puis on passe au suivant. Les quatre trous étant repérés et les vis en place, on trace le contour de la pièce. On dévisse le tout, on découpe le périmètre et l'on obtient ainsi un patron pour arriver à percer les trous de passage des vis avec précision. Ce patron est également utilisé pour situer avec précision la fenêtre de l'afficheur OLED ainsi que le positionnement des deux trous de passage des LEDs.

➤ Intégration et câblage des éléments.

C'est l'aboutissement, la récompense. Couvercle terminé nous avons également improvisé sur Image 29.JPG la cale longitudinale qui interdit tout mouvement transversal à la pile rechargeable de 9V. On commence par préparer la liaison avec la source d'énergie. Sur Image 30.JPG c'est la ligne qui va vers l'accumulateur qui est agencée, et sur Image 34.JPG celle qui la prolonge jusqu'au circuit imprimé et à l'interrupteur. C'est avec Image 31.JPG que débute l'intégration des composants. Sur la Fig.116 on constate que la Bride métallique exerce sur l'accumulateur un effort F décalé latéralement comprimant de façon asymétrique la cale en carton 12. Le petit objet présente alors la tendance à se soulever en S. Pour éviter qu'il n'adopte une position inclinée,



une petite cale de 3mm d'épaisseur **11** est intercalée sous la traverses. Elle est réalisée en polystyrène choc et assortie d'un trou de 4 mm à la verticale de la vis de fixation du couvercle pour ne pas que cette dernière vienne talonner sur l'ensemble. Accumulateur immobilisé, c'est au tour du circuit imprimé à regagner sa place réservée. L'opération peut s'avérer délicate, aussi, pour vous éviter des tâtonnements je vous livre une fiche supplémentaire qui donne avec précision l'ordre des opérations à effectuer pour que les manipulations soient faciles et sans piège. Les deux photographies d'[Image 32.JPG](#) et d'[Image 33.JPG](#) sont bien utiles pour illustrer l'assemblage du module GPS. Quand l'antenne du module GPS est dans son alvéole en carton sous l'afficheur OLED, l'expérience montre qu'elle peut se balader transversalement et occasionne un petit bruit intrigant quand

Fiche n°25.

Ordre d'intégration des éléments dans le coffret.

- 01) Placer le carton long et l'accumulateur dans le coffret.
- 02) Ajouter le carton court et la cale prismatique derrière l'accumulateur rechargeable.
- 03) Placer sur le dessus la petite cale en polystyrène choc en veillant à placer son trou de dégagement sous le trou taraudé du boîtier.
- 04) Immobiliser la bride en plaçant les rondelles larges sur le dessus de l'équerre inférieure.

Préparer le circuit imprimé :

- 05) Les deux LEDs sont déposées. Brancher le HE14 à deux broches femelles qui va à l'interrupteur et glisser ses deux fils sous le Micro Switch. Glisser les deux fils du connecteur mâle entre le condensateur jaune et les supports HE14 des LEDs.
- 06) Mettre en place l'afficheur OLED. Débrancher le RESET pour dégager le trou de liaison du C.I. avec la semelle.

Intégration du circuit imprimé :

- 07) Placer les quatre vis de liaison C.I. et leurs entretoises.
- 08) Surélever la semelle pour que les vis soient au raz des entretoises.
- 09) Mettre en place le C.I. et enlever les cales. Serrer les écrous.
- 10) Rebrancher le RESET sur son HE14 à deux broches.
- 11) **Glisser l'antenne du module GPS dans l'alvéole en carton** située sous l'afficheur OLED.
- 12) Immobiliser l'inverseur MA/AR sur le flanc du coffret.
- 13) Brancher le HE14 d'alimentation avec l'accumulateur.
- 14) Immobiliser le module GPS sur la face arrière.
- 15) Brancher le module GPS au connecteur HE14 à trois broches mâles de raccordement du circuit imprimé.
- 16) Insérer les deux LEDs sur leurs supports respectifs.
- 17) Brancher le codeur incrémental et positionner le couvercle en veillant à placer correctement le toron de fils entre les supports de LEDs et le support de l'afficheur OLED.
- 18) Immobiliser le couvercle avec ses quatre vis.
- 19) Par l'ouverture arrière remonter éventuellement la LED verte sous le couvercle si elle a été un peu enfoncée sur son support.

on manipule le petit appareil. Aussi, il est avantageux de caler vers l'avant et l'arrière ce petit élément avec de la mousse synthétique souple. [Image 35.JPG](#), [Image 36.JPG](#) et [Image 37.JPG](#) présentent l'appareil terminé. Pour valider le câblage il n'est pas obligatoire de brancher le codeur incrémental, encore qu'une liaison temporaire n'est pas délicate à établir. Il ne reste plus qu'à préparer le couvercle. La petite "vitre" taillée dans une plaque en matière thermoplastique d'environ 1mm d'épaisseur est disponible avec ses trous de liaison percé pour de la visserie ϕ M2. (*C'est une plaque que l'on trouve pour réaliser des encadrements photographiques. Approvisionner une référence parfaitement transparente et sans "anti reflets".*) Elle est assemblée sur le dessus coté intérieur du coffret et montrée sur [Image 38.JPG](#). Comme le serrage des écrous doit rester modéré pour ne pas risquer de fendre le fragile matériau, une petite larve de vernis à ongles assure un freinage très efficace de la liaison. On peut alors immobiliser comme sur [Image 39.JPG](#) le codeur incrémental sur la plaque du dessus. On branche ce dernier sur son connecteur HE14, on insère les deux LEDs sur leurs supports respectif et l'on peut enfin terminer l'assemblage en [Image 40.JPG](#) et [Image 41.JPG](#). Il se trouve que la petite LED verte de 3mm de diamètre peut glisser et s'enfoncer notablement dans son HE14 femelle à deux contacts. Du coup la mise en place du dessus peut la pousser vers le bas. Pour qu'elle puisse traverser le couvercle il suffit de la remonter au moyen d'une pince Précelle d'électronicien grâce à l'ouverture bien visible sur [Image 35.JPG](#). Enfin, pour achever d'équiper notre appareil, on le retourne et comme montré sur [Image 42.JPG](#) on le munit de quatre petits pieds en feutre pour ne pas risquer de rayer le lustre des meubles en le posant sur les têtes de vis. Dernière mise sous tension pour "signer le bon de sortie". Il est maintenant très facile de l'emmener avec nous lors de nos sortie quotidiennes pour effectuer les tests ultimes de validation lorsque l'on est en déplacement dans une automobile ... ou à l'intérieur d'un aéronef !

13) Dernière application autonome.

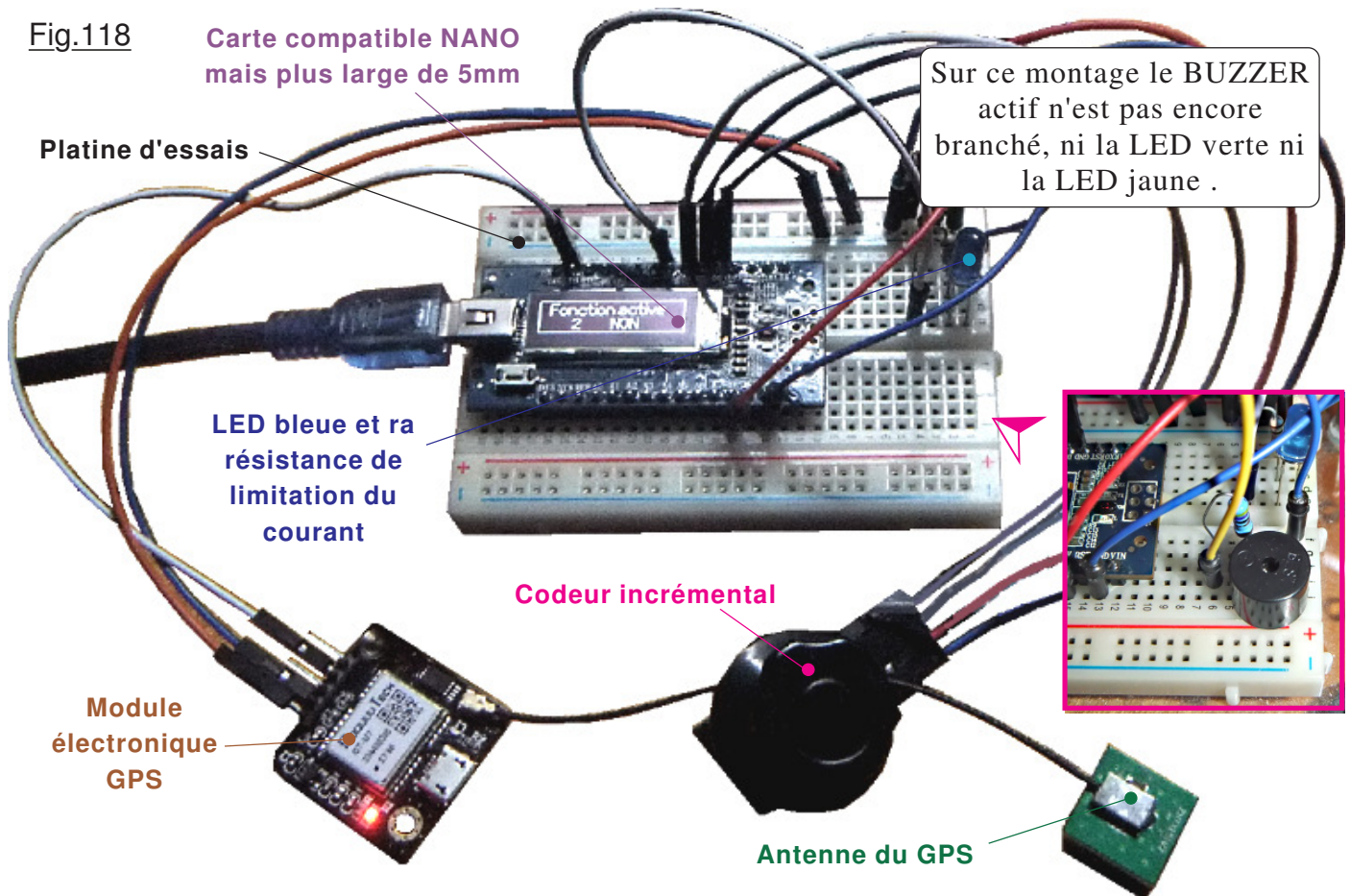
Fig.117

Cette expérience [Application_GPS_9.ino](#) n'est qu'une variante dans laquelle on utilise la carte Arduino un peu spécifique de la Fig.117 qui incorpore sur le circuit imprimé un afficheur OLED de définition 128 x 32. Elle présente le même format qu'une carte Arduino NANO en étant juste plus large de 5mm. Ses caractéristiques sont strictement identiques à celles d'une NANO, mis à part le fait que

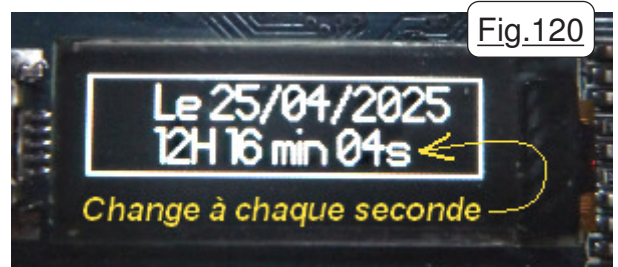


A4 et **A5** sont réservées et reliées à **SDA** et **SCL** de l'afficheur. On pourrait à juste titre se demander pourquoi la version avec coffret n'est pas décrite la dernière. Je n'avais pas envisagée cette ultime proposition, car impossible depuis plusieurs semaines d'arriver à faire fonctionner cette carte. Et puis, alors que je n'ai pas encore mis en ligne, j'ai réussi à téléverser du code et trouvé la bibliothèque adaptée à ce module. Bien qu'ayant utilisé [CH34x_Install_Windows_v3_4.EXE](#), le dialogue entre le PC et la carte s'avérait impossible. J'ai tenté de remplacer le **Boot Loader** depuis une carte Arduino UNO et les téléversements ont immédiatement fonctionné. Enfin, j'ai rencontré de réelles difficultés à me procurer une bibliothèque [U8g2lib.h](#) qui soit acceptée par le compilateur. Aussi, je vous la fournis dans le dossier [<EXPERIENCE_30>](#). À toutes fins utiles, je vous propose également dans le dossier [<EXPERIENCE_30>](#) trois petits démonstrateurs qui m'ont permis de tester les instructions de base de cette bibliothèque dédiée à l'afficheur local à cette carte Arduino. C'est l'un d'eux qui sur la Fig.117 sert à tester l'afficheur, conjointement au codeur incrémental. Sur la Fig.118 on peut voir le montage provisoire de mise au point du programme. Il est évident que si vous adoptez cette carte, l'électronique sera plus compacte et il vous faudra adapter les dimensions du coffret et simplifier le circuit imprimé pour optimiser l'ensemble. Compte tenu de la faible résolution verticale de l'afficheur, le concept qui préside la conception du programme consiste à adopter la police de caractère la plus grande possible, permettant comme pour la version [Application_GPS_1.ino](#) de disposer de deux lignes de 16 caractères. Nous allons donc reprendre globalement les pages-écran de l'afficheur LCD avec juste en plus l'encadrement extérieur comme sur la Fig.117 qui constitue un test d'utilisation de l'afficheur avec le codeur incrémental relié au module Arduino. Il aurait été plus séduisant de ne pas encadrer et d'employer une police de caractère plus grande, mais parmi toutes celles disponibles et compatibles qui permettait de loger 16 caractères en largeur reste introuvable.

Fig.118



Lorsque l'on effectue un RESET ou à la mise sous tension, la première page-écran qui sera affichée est celle de la Fig.119 qui présente la Date ainsi que l'heure. La date ainsi que toutes les données issues du système ne seront correctes que lorsqu'un minimum de satellites sera capté dans des conditions utilisables. Comme montré sur la Fig.120 le changement de l'heure est cadencé au rythme de réception des groupes de données, c'est à dire une fois par seconde. Sur la Fig.119 on peut observer que la LED rouge PW de la carte Arduino qui était vraiment trop lumineuse se fait discrète, car elle a été masquée par de l'encre de chine noire. Elle reste toutefois légèrement lumineuse pour permettre des vérifications tout en laissant la vedette à la LED sur **D13** et celles de **TX** et **RX**. Noter au passage que dès le démarrage du programme une LED jaune sur **D6** s'illumine durant le traitement des données en moyenne une fois par seconde pour témoigner du bon fonctionnement de la boucle de base.



➤ Les manipulations logicielles indispensables.

Attention, le programme `Application_GPS_9.ino` ne fonctionnera pas si vous vous contentez de le téléverser directement. Il faut au préalable utiliser deux outils logiciels. Bien que ce soit précisé en tête de listage, il me semble raisonnable d'insister ici sur ce préalable impératif. Avant de pouvoir utiliser ce croquis d'exploitation, il faut commencer par téléverser `Ecrire_en_EEPROM_2.ino` sur la carte Arduino et surtout activer le **Moniteur de l'IDE** pour que les données de base soient effectivement inscrites en EEPROM. Au départ il sera également indispensable d'inscrire des points d'intérêt en mémoire non volatile en utilisant l'outil nommé `Emplir_EEPROM.ino` mais **ATTENTION : Prendre celui qui est rangé dans le sous-dossier <Experience_30>** qui regroupe les démonstrateurs relatif à cette version ultime du GPS. (Nécessaire car sur RESET le programme charge en mémoire vive le P.I. par défaut.)

➤ Les écrans de base dans le sens horaire.

Tout changement de position du codeur incrémental allume la LED bleue pour nous informer que l'action a été prise en compte. Elle s'éteint lorsque le programme qui doit terminer la séquence en cours commence à traiter la requête de changement de Page-écran. Dans l'ordre croissant, c'est à dire pour une rotation dans le sens horaire, on voit défiler les uns après les autres les écrans de la Fig.119 à la Fig.123 qui enchainent sur la page qui ouvre la *fonction de cheminement* explicitée dans le chapitre suivant. En Fig.121 l'appareil précise la distance qui nous sépare de notre cible qui par défaut est chargée depuis l'EEPROM sur RESET. En Fig.122 ce sont les coordonnées géographiques de positionnement qui sont indiquées. C'est un peu le minimum que l'on puisse exiger d'un GPS ! La page de la Fig.123 indique le décalage en hauteur par rapport au niveau moyen des océans sur la ligne du haut. Sur la ligne du bas le programme précise le nombre de satellites en poursuite et le nombre de ceux utilisés pour extraire les données de positionnement. *(Comme déjà précisé, tant que le nombre de satellites en position favorable n'est pas suffisant, les données affichées par notre GPS sont incomplètes ou incorrectes.)* Enfin, lors d'un RESET ou étant en page de la Fig.119 on tourne le codeur incrémental dans le sens négatif, on ouvre la page de la Fig.124 qui précise l'état des deux options qui peuvent être librement modifiées lors du *dialogue homme / machine sur RESET* décrit plus avant.

Fig.121

Fig.122

Fig.123

Fig.124

➤ La fonction cheminement.

Déjà développé en page 45 et en page 50, le principe est strictement analogue, seule la façon de saisir les deux jalons de début et de fin diffèrent ainsi que la présentation de l'écran durant l'utilisation de cette fonction. Comme pour *Application_GPS_8.ino* l'ouverture du *mode cheminement* commence par saisir en Fig.125 le point d'intérêt d'ARRIVEE de notre route. Pour montrer que l'on est en phase d'initialisation du cheminement, l'écran ne trace pas l'encadrement. La rotation du codeur incrémental fait défiler en ligne du bas les noms des PI disponibles en EEPROM. Pour mémoire ils doivent être placés les uns à la suite des autres. Aussi, pour faciliter le travail, demandant la cible d'ARRIVEE le programme indexe la dernière disponible en EEPROM. Durant la rotation en fin de ligne du haut est précisé le numéro d'ordre en EEPROM du PI pointé. Pour valider on clique sur le **BPC**, la page écran de la Fig.126 s'affiche alors pour nous faire préciser le premier point de passage servant de cible. Tourner le codeur incrémental fait défiler les PI dans un sens ou dans l'autre en commençant par indiquer le premier situé en EEPROM car **DEBUT** doit dans cette dernière se trouver avant **ARRIVEE**. Cliquer sur le **BPC** fait passer en Fig.127 c'est à dire à l'affichage courant du mode cheminement. On retrouve l'encadrement dans la présentation des données. Sur la ligne du haut est indiqué le nom du prochain point de passage vers lequel on est en train de se déplacer. Sur la ligne du bas on trouve *à gauche la distance qui nous en sépare. Sur la droite la distance qui nous sépare de la destination d'ARRIVEE*. Les deux distances sont indiquées dans les mêmes unités et comme on peut le constater sur la Fig.127 et sur la Fig.128 on pourra à notre guise choisir entre les kilomètres ou les miles marins notés **Nm**. La difficulté de mise en page avec l'afficheur intégré à la carte Arduino réside dans ses polices de caractères dont la

Fig.125

Fig.126

Fig.127

largeur des lettres est différentes en fonction de leurs formes. Par exemple le **1** est bien plus étroit que le **8** et sur la Fig.127 on voit que le point décimal de **21.73** est relativement petit et à peine visible sur la photographie pourtant saisie en "MACRO". Par ailleurs, on désire pouvoir afficher les distances avec une précision de 10m. Hors, si on envisage un déplacement à un point opposé du notre sur la Terre, l'affichage devrait ressembler à **1234.56 / 22375.42 km**. Une telle information est bien trop large pour l'écran. Aussi, j'ai adopté le compromis qui consiste à n'afficher les deux décimales que lorsque la valeur de la distance devient inférieure à 100 quelles que soient les unités adoptées. Ainsi, sur la Fig.127 on se trouve à **21,73**km du prochain point de passage, avec les 100m et les 10m, lors que la distance jusqu'à l'arrivée est de 3456km, les décimales étant alors ignorées.



Fig.128

➤ Le dialogue USB sur RESET.

C'est une solution qui impose l'utilisation d'un ordinateur avec l'**IDE** installé sur ce dernier. Du coup l'appareil n'est pas totalement autonome. En revanche, la gestion des PI en EEPROM est particulièrement agréable, et l'on n'a vraiment pas besoin d'avoir en tête tout un tas de protocoles. Le dialogue reprend strictement les techniques décrites en page 24 et page 25 avec quelques commandes différentes. Pour ouvrir le menu du dialogue série il faut :


- 01) Ouvrir le **Moniteur de l'IDE** et imposer la vitesse de 19200bauds.
- 02) Maintenir enfoncé le bouton central du codeur incrémental. (**BPC.**)
- 03) Cliquer sur la puce "loupe"  pour ouvrir le **Moniteur de l'IDE**.
- 04) Attendre que la LED d'Arduino s'illumine et relâcher le **BPC.**

Fig.129

La fenêtre de la Fig 129 s'affiche sur l'écran OLED et la LED bleue se met à clignoter à environ 2Hz. Conjointement, la fenêtre contextuelle du moniteur affiche les données de la Fig.130 sur l'écran de l'ordinateur.



- 05) **Modifier à convenance toutes les données à mettre à jour.**
- 06) Saisir la commande "**q**" pour revenir **sans préavis** au **Menu de base**.

Version du 08/05/2025
PILE - TAS = 190

Fig.130

? ou , : Ce MENU.

A ou a : Ajouter un PI. _____

C ou c : Corriger un PI. _____

D ou d : Déplacer un PI. _____

E ou e : Effacer un PI. _____

G ou g : Gommer tous les PI - 1. _____

H ou h : **Hiver** **NON**

L ou l : Listage des PI en EEPROM.

M ou m : Mémoire dynamique. _____

N ou n : **Dst en Nautiques** **NON**

P ou p : PI sur RESET. _____

Q ou q : Quitter les saisies.

S ou s : **Silencieux** **OUI**

Num 01 :	4445544N	00152192E	[Ma Maison.]
Num 02 :	4737359N	00312634E	[Donzy.]
Num 03 :	4383763N	00438378E	[NIMES.]
Num 04 :	4437051N	00480745E	[Clansayes.]
Num 05 :	4885365N	00235026E	[PARIS (Ntr Dame)]]
Num 06 :	9000000N	00000000E	[Pole NORD.]
Num 07 :	9000000S	90000000W	[Pole SUD.]

Lorsque le dialogue série sur RESET démarre, le programme commence par préciser sa version dans la zone jaune. Puis, en deuxième ligne la "classique" information sur la place qui reste entre la **PILE** et le **TAS** avec pour "incidence" le regret qui clôtüre ce dernier chapitre. La zone bleue pastel est la liste des PI actuellement en EEPROM, leur nombre maximal étant de dix. Pouvoir les ordonner avec la commande "**d**" est important pour construire un cheminement. Les PI doivent se suivre dans l'ordre de rencontre sur la route. Par exemple ici, le cheminement prévu "commence" en n°2 jusqu'en n°5 en passant par le n°3 puis le n°4. Si on doit en ajouter, ils se trouvent forcément à la fin. Avec "**d**" il est facile ensuite de les ordonner. Dans cette liste, celui qui est chargé automatiquement sur un RESET est pointé par

< **RESET** le texte **<RESET**. Il est facile à modifier avec la commande "**p**".

Trois options sont initialisables et leur état est précisé dans l'affichage du menu. En vert c'est uniquement le mode **dialogue homme / machine sur RESET** qui est concerné. En rose, ce

sont les deux options pour le *Menu de base*. Chaque fois que l'on utilise "h" ou "n" l'option concernée est inversée. Son état dans le listage des commandes est alors mis à jour. Quand l'option *Dst en Nautique* est à OUI, les distances seront affichées en miles marins. À NON elles seront en kilomètres. Les commandes relatives à la gestion des PI en EEPROM sont repérées par les lignes oranges. Il aurait été logique de les regrouper dans le rappel des commandes. Toutefois, j'ai préféré les placer dans l'ordre alphabétique d'où leur dispersion apparente. Si certaines commandes ne vous semblent pas évidentes à utiliser, elles sont toutes décrites dans les chapitres précédents.

J'aurais bien aimé ajouter une Page-écran qui indique la vitesse du mobile ainsi que le cap suivi, ces deux fonctions étant au point sur *Application_GPS_8.ino* il suffirait de modifier la présentation des données à l'écran. Mais toutes les fonctions ajoutées obligent à placer des textes directement dans le programme. Nous savons que les textes, même s'ils ne sont utilisés que pour des menus, c'est à dire qu'ils sont des constantes, pour le compilateur ils sont considérés comme des tableaux de *char*. C'est la double peine, car chaque texte augmente la taille du programme de sa longueur, mais surtout diminue d'un même nombre d'octets la zone libre entre la *PILE* et le *TAS*. Dans l'état actuel du logiciel d'exploitation, il n'est plus possible d'ajouter du texte en EEPROM, et *PILE - TAS* = 190 à l'ouverture du dialogue Homme/Machine. C'est tout juste suffisant pour ne pas se heurter à un problème de *collision de pile*. Du reste, le phénomène se produisait quand j'ai ajouté *Affiche_la_PAGE_6()*. Pour arriver à faire fonctionner le programme, j'ai été obligé de réduire certains textes et d'adopter un style plus "télégraphique" pour arriver aux 190 octets actuels. Donc, bien qu'il reste encore plus de 3000 octets de disponibles en mémoire de programme, permettant d'ajouter encore beaucoup de code, il est difficile de compléter avec des fonctions, car chacune imposerait du texte sur l'écran, et actuellement ce n'est plus envisageable.

14) Avant de se quitter : Un petit bilan.

Chère lectrice, cher lecteur, à l'analyse des listages de tous ces démonstrateurs, vous allez constater des répétitions, des constances, des redondances dans les commentaires. C'est que tous ces logiciels sont organisés sur une trame identique pour des raisons de rigueur. Par ailleurs, pour certains, les démonstrateurs seront visités "aléatoirement" et aux désirs des expérimentateurs. Il importe qu'ils soient indépendants et totalement "autonomes". C'est la raison pour laquelle des dizaines de fois on retrouve en tête des commentaires analogues. C'est un peu comme ces publicités à la télévision que l'on vous impose à longueur d'année comme si vous n'aviez rien compris. Ici c'est pour des raisons d'indépendance de chaque croquis qui forme "un tout". Enfin mes démonstrateurs s'adressent en priorité aux débutantes et aux débutants. Alors je préfère en écrire trop que pas assez.

Programmer n'est pas un long fleuve tranquille. On établit des stratégies, on teste, on avance et survient un problème qui remet les approches précédentes en cause. On change de route et petit à petit, ces chassés croisés finissent par porter leurs fruits et l'on aboutit à un programme dont on est fier. Quand je m'engage dans un tel projet, je sais qu'il va m'occuper plusieurs semaines. Aussi, je suis dans l'obligation de rédiger le didacticiel "au jour le jour", car à la fin il serait impossible de tout retrouver pour justifier les divers choix effectués et l'agencement des démonstrateurs. Du coup, le tutoriel n'est pas linéaire. On chemine de découvertes en découvertes avec de fréquents petits détours, de nouvelles fonctions, des présentations qui se modifient. Bref, si vous testez les diverses expériences proposées, vous "subissez" un peu les changements de stratégie et les remises en cause. Je pense que c'est presque un avantage, car au final les démonstrateurs explicitent toutes les décisions et les choix effectués. Tous apportent leur pierre à l'édifice et peuvent enrichir les lectrices et les lecteurs dans notre domaine ludique de programmation sur Arduino.

Dès que le coffret a été terminé, j'ai apporté les dernières petites modifications et rédigé le chapitre sur sa réalisation. Puis j'ai entièrement créé ce coffret en virtuel sur un modelleur 3D pour pouvoir vous proposer les dessins cotés. Enfin, j'ai osé mettre en ligne alors que la *fonction* qui traite le *cheminement* a été entièrement achevée et *validée*, mais à mon sens pas suffisamment *sur plusieurs trajets un peu longs*. Pour les autres fonctions, les innombrables tests de vérification me permettent d'être assez serein sur la validation du logiciel. Nous savons tous que l'on n'est jamais à l'abri d'une petite vermine qui profitera de circonstances tellement particulières dans la combinatoire des possibles, que le test n'aura pas été envisagé. Bref, un "bug" peut toujours survenir. Ceci étant précisé, je reste confiant sur le comportement sain du programme, cette éventualité

reste à mon sens peu probable. Il est évident que seul un usage "intensif en conditions réelles" sur de long parcours confortera la confirmation d'un comportement fiable. Et si un hiatus émergeait de l'usage en situation, le programme sera repris pour le corriger, et ce d'autant plus facilement que le raccordement au PC pour les téléversements est prévu dans l'usage standard du petit appareil.

Finalement l'internaute se retrouve face à huit versions logicielles possibles et peut se sentir légèrement perdu pour effectuer un choix avant de s'engager. Si vous disposez du matériel nécessaire, je vous invite à toutes les tester avant de privilégier celle qui remportera votre suffrage, étant entendu que pour ma part la version graphique avec affichage de la carte de France emporte ma décision. Si vous n'avez qu'un afficheur LCD, l'impossibilité de représenter cette carte rudimentaire n'est pas du tout tragique. Ce n'est qu'un luxe et surtout le plaisir de l'avoir programmé. Par contre, j'insiste sur le choix du composant qui pour cette version doit être vraiment "LCD" et ne pas imposer un rétro-éclairage beaucoup trop gourmand en énergie. Si vous voulez limiter l'investissement matériel, le tableau donné ci-dessous vous aidera à faire un choix avant de passer des commandes de matériel.

Version	Afficheur	interface	Particularité
GPS_1.ino	LCD	Codeur rotatif	Pas besoin de programmer l'EEPROM.
GPS_2.ino	LCD	Clavier à deux BP	Textes en EEPROM. (Voir Fiches 19 et 20)
GPS_3.ino	LCD	Codeur rotatif	Textes en EEPROM. (Voir Fiches 19 et 20)
GPS_4.ino	OLED 1,3'	Codeur rotatif	Graphique France et planisphère.
GPS_5.ino	OLED 0,96'	Codeur rotatif	Graphique France mais pas le planisphère.
GPS_6.ino	OLED 1,3'	Codeur rotatif	GPS_4.ino qui facilite l'étude du C.I.
GPS_7.ino	OLED 1,3'	Codeur rotatif	Sans planisphère et cheminement amélioré.
GPS_8.ino	OLED 1,3'	Codeur rotatif	GPS_7.ino avec Cap vectorisé sur la carte.

Chères lectrices, chers lecteurs, cette (trop) longue saga arrive à son terme. Tout à une fin, mis à part l'Univers, et arrive forcément un moment où il faut raisonnablement considérer que "le travail" est terminé.

Je souhaite intensément que certaines et certains oseront s'engager dans la réalisation d'un clone, je ne doute pas de leur réussite. Surtout, je vous souhaite à toutes et à tous de trouver dans ces lignes le plaisir de la découverte. Si d'aventure vous engagez vos heures de liberté dans une telle réalisation et que vous rencontriez une difficulté, les amis du forum pourront probablement vous aider. Dans le pire des cas, vous pouvez me contacter sur : michel.droui@laposte.net et dans les limites de mon temps de libre, c'est avec grand plaisir que je tenterai de vous dépanner. Je vous souhaite à toutes et à tous agréable lecture et ... de ne plus vous égarez grâce à ce petit appareil.

Chaleureusement : Nulentout.

