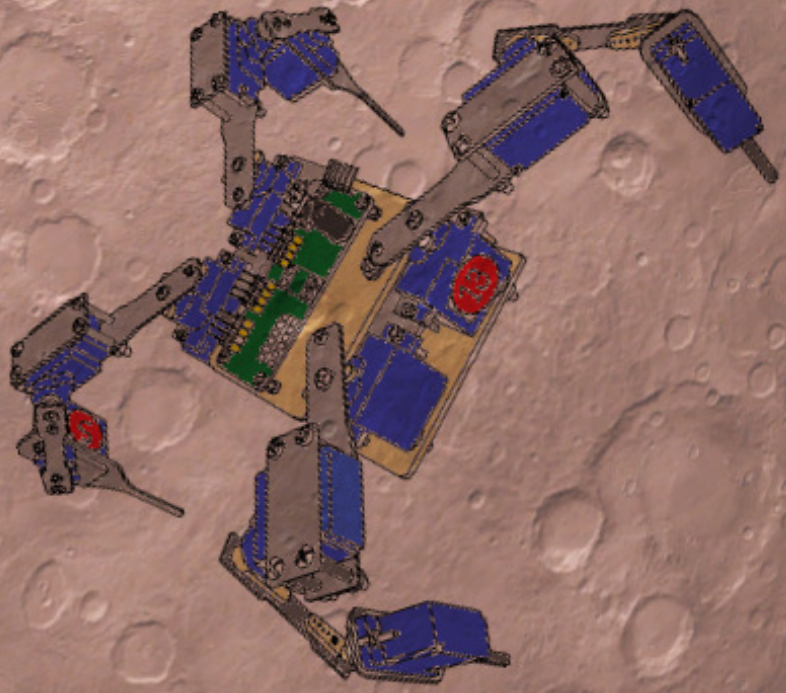
Sonde martienne JEKERT
Pilotage autonome avec une raquette de télécommande

TOME 5



Landing Site

Par Nulentout : Lundi 5 Mars 2018.

Développement du projet Sonde martienne JEKERT

Kourou sommeille. Au centre spatial tous les projecteurs sont braqués sur le lanceur encore relié à la tour de servitude par les cordons ombilicaux. Dans la grande salle de contrôle c'est l'effervescence. Les voyants qui conditionnent l'avenir sont au vert. Tout le tour de la Terre les stations de poursuite sont en alerte. La procédure de lancement s'étire avec une régularité de métronome, comme si le lancement d'une fusée relevait d'une routine anodine.

- Ici le DDO, le lanceur passe en chronologie automatique,

Les images sont retransmises en direct sur Internet. On voit nettement les lignes d'alimentation se débrancher et s'écarter d'Ariane qui semble inerte et endormie.

- Cinq, quatre, trois, allumage des moteurs Vulcain,

Le complexe s'embrase d'une furie lumineuse impressionnante. Des panaches de fumée ocres se bousculent pris de panique et semblent vouloir fuir cet enfer dans toutes les directions.

- Deux, un, zéro, allumage des booster à poudre,

Les moteurs sont au nominal, l'ajout des fusées d'appoint à poudre produit sur l'ensemble une poussée phénoménale. Pratiquement sans délai la fusée quitte son support et dépasse la tour de servitude.

- Décollage, tous les voyants sont au vert, la trajectoire est nominale ...

Il ne faut que quelques minutes pour que JEKERT passe le mur du son et aille rapidement se cacher derrière les nuages. On devine encore un moment sa présence car le ciel, bien que presque entièrement couvert, rougeoie dans la direction de la fusée qui déjà se trouve à plus de trente kilomètres.

Le lancement est un succès. La première orbite confirme que la petite sonde a parfaitement résisté au traumatisme du lancement. Après trois autres orbites, les paramètres de la trajectoire sont affinés et c'est l'allumage des moteurs orbitaux du petit vaisseau spatial. Quand le lendemain les journaux relate du lancement réussi, on effectue une dernière petite correction de trajectoire. L'exploratrice est sur les rails gravitationnels qui maintenant vont la guider sans faille jusqu'à la proche banlieue de Mars. Le voyage sera long, pratiquement six mois. Aussi on l'a fait passer en veille pour une hibernation qui la préservera du froid interplanétaire. Elle ne sera réveillée que pour l'apprentissage de nouvelles routines informatiques, ou s'il faut effectuer une ou deux petites corrections de trajectoire.

À la NDRMSE c'est la liesse, une joie non contenue car ce lancement réussi couronnes des années d'efforts. Toutefois, les petites étiquettes colorées n'ont pas été toutes retirées du planning mural. Il reste encore un quart de la surface encombrée de petits cartons multicolores. Il faut se remettre immédiatement au travail car tout doit être terminé quand arrivée à proximité de Mars le moteur orbital de JEKERT sera rallumé pour freiner et se mettre en orbite autour de la planète rouge. Pas de temps à perdre. On enfile les blouses blanches, on s'autorise un dernier café et sur le métier on replace l'ouvrage, car la route est encore longue avant de pouvoir prendre des vacances.

Piloter JEKERT à partir de la ligne USB et le moniteur de l'**IDE** est parfait pour développer le logiciel au moindre frais et surtout avec un maximum de facilité. Ce n'est toutefois pas la solution la plus conviviale car à proximité de l'ordinateur ne se trouve pas forcément une table suffisamment dégagée pour faire évoluer la petite machine. Par ailleurs, quand on désire présenter l'insecte robotisé à des amis, lors d'une soirée d'hiver, c'est au salon qu'ils sont reçus. Ce n'est pas l'endroit où habituellement réside le P.C. Aussi, si l'investissement matériel est envisageable, télécommander JEKERT avec un petit pupitre autonome est très alléchant. Sur le plan informatique, cette facette à débroussailler n'est pas élémentaire du tout car il va falloir faire communiquer deux machines. Du coup, ce sont deux logiciels qui vont devoir évoluer en parallèle : Celui sur JEKERT et celui sur la ***Raquette de commande***. On part de zéro pour défricher cette nouvelle perspective. Nous allons devoir définir entièrement l'aspect opérationnel, esthétique, convivial du dispositif à agencer. Autant dire que l'éventail d'investigations se montre merveilleusement vertigineux ...

44) 15/12/2017 : Une cure d'amaigrissement (MJD 58102)

Retour en S4 dans laquelle les ingénieurs logiciels ont repris le collier. Sur le mur une belle affiche montrant une coupe de la fusée avec tout en haut dans la coiffe la sonde bien lovée force la bonne humeur. Plusieurs articles ont été découpés dans la presse locale. L'autre nuit, le décollage avait un peu paralysé les esprits, car un lancement n'est jamais de la routine. Un petit rien peut tout faire capoter. Quand la puissance est nominale sur tous les moteurs et que l'ordinateur détectant une poussée supérieure au poids de l'ensemble débride la fusée, le sort du lanceur et de sa charge utile est définitivement scellé. On ne peut plus rien faire, il faut impérativement qu'une mise en orbite soit effective. Si ce n'est pas le cas, le projet tombe à l'eau ... de mer ! Et encore, faut-il que l'orbite atteinte permette ensuite une éjection vers Mars. On ne peut plus compter sur un sauvetage avec la Navette. Aussi, quand le D.D.O. à confirmé une trajectoire correcte vers la planète rouge, toute l'équipe a poussé un soupir de soulagement et fait éclater sa joie. La mission des "binaires" n'est pas pour autant terminée, il reste encore pas mal de travail sur la planche ...

➢ Faire des économies, un leitmotiv souvent entendu aux informations du 20H.

Force est de constater que le programme complet actuel est un peu obèse. Il intègre des fonctions qui sont bien utiles pour les programmeurs, mais dont les techniciens d'exploitation n'ont que faire. Par exemple lister le contenu de l'EEPROM. Franchement, faire afficher tous les octets est spectaculaire. Mais c'est totalement stérile pour s'occuper de JEKERT sur le sol poussiéreux. Aussi, si c'est pour vérifier le contenu de l'EEPROM quand on a modifié le programme pour des adaptations locale, rien n'interdit de téléverser P30, de vérifier à loisir le contenu de la mémoire non volatile puis de reloger le programme d'exploitation. Le nouveau noyau qui servira à dialoguer avec une raquette de pilotage est nommé P30_Programme_COMPLET_T5.ino et contient tous les changements qui vont se voir explicités dans les chapitres qui suivent de ce TOME 5.

Première modification : Avant d'envisager une purge, au contraire on va ajouter une commande car avec tous ces démonstrateur, arrive un moment où on ne sait plus quel est le dernier logiciel téléversé. Alors tant pis pour la boulimie, on ajoute ""*" qui aura pour effet d'indiquer sur le moniteur la version du code objet qui "tourne" actuellement sur la carte Arduino NANO. Cette référence est désignée comme constante dans les lignes //@@@@@@@@@@@@@@ et sera modifiée pour chaque nouveau démonstrateur.
GLUPS ! Quand on ajoute la ligne de code qui fait afficher la version ... la taille du programme diminue de 14 octets, c'est fabuleux. *(La compilation a parfois des effets difficiles à interpréter ...)* Par contre, dans les variables dynamiques on consomme 14 octets de plus correspondant à la constante chaîne de caractères "Version P30-T5.". Ce n'est donc pas totalement gratuit.

- Hééé, Totoche, j'ai trouvé un régime que plus tu manges, moins tu pèse !

Supprimer la fonction de listage de la mémoire fait économiser 272 octets. De plus, la commande "z*" redevient disponible. Elle sera donc utilisée pour faire afficher la version du programme.

➢ Reporter la responsabilité sur les autres !

Fausse bonne idée : La technique qui impose un programme préalable s'avère mal pensée. Elle paralyse globalement les manipulations, car on est en permanence agressé par des "!ERR 21!" alors que la commande envoyée est potentiellement sans danger. Par exemple envoyer deux fois de suite "p03*" provoque une erreur, alors que faire avancer deux fois la sonde ne risque pas de provoquer des interférences matérielles internes. Aussi, si l'on passe en revue les divers programmes, il n'y en a que trois ou quatre qui risquent de conduire à des incidents.

CHANGEMENT DE STRATÉGIE : ***Sur le programme qui anime JEKERT on élimine tous les tests relatifs au programme préalable. Pour les commandes qui potentiellement engendrent un risque, on traitera la vérification sur le programme de la raquette de pilotage.***

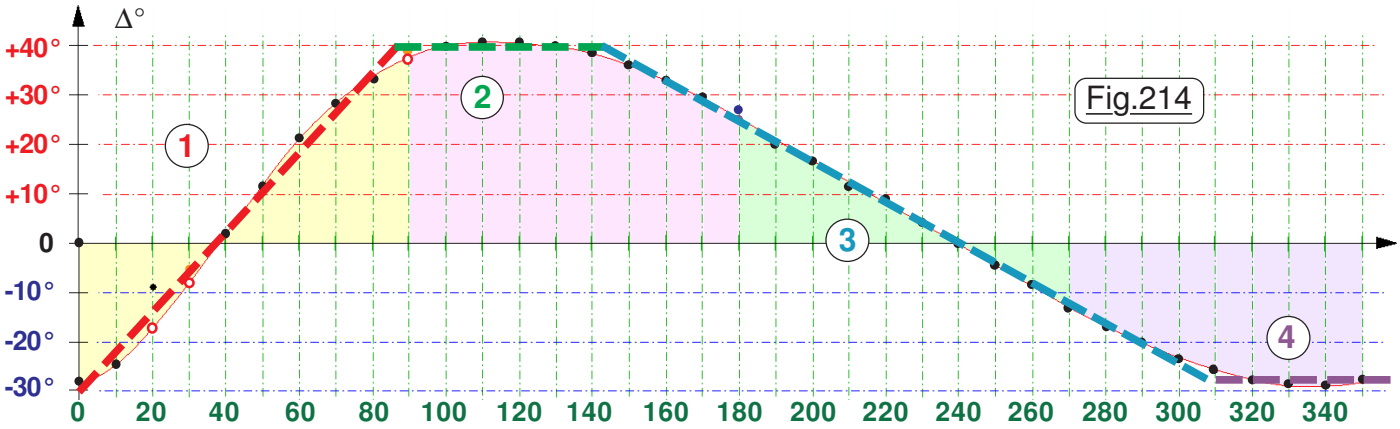
C'est cette modification qui justifie que sur la Fiche n°28 l'organigramme a été barré en vert.
Cette simplification rend les campagnes de test plus faciles et fait gagner encore 70 octets de programme. Par ailleurs on élimine le tableau des préalables ce qui diminue de 24 emplacements la place occupée en mémoire dynamique. Ces quelques modifications rendent les tests plus simples et font gagner 356 octets pour le programme, c'est loin d'être négligeable.

45) 16/12/2017 : Correction automatique du compas de route (MJD 58103)

Réinvestir un capital est un reflexe "capitaliste" pour ne pas qu'il perde de sa valeur. Ce propos pour le moins "financier" n'est pas forcément hors sujet dans un petit projet comme le nôtre. Dans cette manipulation nous allons réinvestir les 356 octets économisés par les mesures drastiques entreprises dans le chapitre précédent. Autant faire afficher des octets sur l'écran était un luxe dont on peut royalement se passer, autant utiliser la belle Fiche n°26 n'a rien de convivial. Si l'on désire orienter la sonde à un CAP magnétique particulier, il faudra constamment interpréter les données de correction. Aussi, envisager une correction automatique par le calculateur de bord est séduisant. La courbe de correction affichée en rouge sur la Fig.150 en page 30 du **TOME 3** fait un peu peur, car pour la reproduire avec des équations trigonométriques on devine confusément qu'il va falloir introduire des harmoniques et de la développée de Fourrier. BEURRRRKKKKK !

➢ Retrouver le Nord.

Outre le fait que ce ne sera pas facile de trouver les équations qui seront capables de traduire la courbe diabolique, leur traduction en langage C++ va engloutir tout le capital restant pour le code objet. Ce n'est évidemment pas acceptable. Toutefois, si l'on regarde un peu mieux le graphe de la Fig.150 et si l'on accepte une correction avec une imprécision pouvant aller jusqu'à ±5°, la situation n'est pas aussi dramatique qu'il n'y parait. Sur la Fig.214 on a superposé des droites à la



courbe rouge, et l'on peut vérifier que ces dernières masquent relativement bien l'ancienne trace. On peut alors considérer que la réalité sera proche des quatre cas **1**, **2**, **3** et **4**. Pour traiter les cas **2** et **4** la solution est immédiate, il suffit de retrancher 40° ou d'ajouter 29° au cap mesuré pour avoir l'orientation réelle. Pour les cas **1** et **3** il faut établir une **correction linéaire**.

- **1** : De 0 à 85° : **Interpoler** soit 0/+30 à 85/-40.
- **2** : De 86° à 143° : Enlever 40°.
- **3** : De 144° à 307° : **Interpoler** soit 144/-40 à 307/+28.
- **4** : De 308 à 359° : Ajouter 29°. *(Corriger si valeur >359)*

Il se trouve que ***linéariser une valeur entre deux limites*** est très facile avec le langage C++ de l'**IDE**, l'instruction map() est conçue dans ce but. Elle peut manipuler librement pour les limites des valeurs positives ou négatives et dans un ordre quelconque.
La procédure Mesure_orientation_magnetique() se terminera maintenant par la séquence de correction du compas de route. Pour ne pas augmenter la place consommée pour les variables du programme, elle réutilise localement Numerisation, CAN et LSB.

Considérons le listage de la séquence qui traite la correction magnétique :

L'instruction ① effectue une lecture du capteur puis divise la valeur par dix pour l'avoir en degrés. Pour faciliter l'aiguillage, car il faut distinguer les quatre cas, on transforme le float en int dans la variable Numerisation. Les quatre lignes qui suivent à partir de ② ont pour but de sélectionner le cas dans l'identificateur CAN par les représentations numériques **1**, **2**, **3** et **4**. La ligne ③ peut alors aiguiller le traitement en fonction de la valeur de Numerisation. S'il faut réaliser une transposition linéaire comme pour le cas **1** par exemple, on commence en ④ à calculer la valeur de la correction à appliquer dans la variable CAN. Puis en ligne ⑤ on ajoute cette ***correction linéaire algébrique*** à la valeur mesurée Numerisation. L'instruction break fait sauter les aiguillages qui

```
    // ========== Correction automatique du compas de route ==========
①   Numerisation = int(CAP_construit / 10); // Mesure en dixièmes de degrés.
②   if ((Numerisation >= 0) && (Numerisation < 86)) LSB = 1;     // De 0° à 85°.
    if ((Numerisation > 85) && (Numerisation < 144)) LSB = 2;    // De 86° à 143°.
    if ((Numerisation > 143) && (Numerisation < 308)) LSB = 3;   // De 144° à 307°.
    if ((Numerisation > 307) && (Numerisation <= 359)) LSB = 4;  // De 308° à 359°.
③   switch (LSB) {
④      case 1 : {CAN = map(Numerisation,0,85,+30,-40);
⑤                Numerisation = Numerisation + CAN; break;}
⑥      case 2 : {Numerisation = Numerisation - 40; break;}
       case 3 : {CAN = map(Numerisation,144,307,-40,+28);
                 Numerisation = Numerisation + CAN; break;}
⑦      case 4 : Numerisation = Numerisation + 29;}
⑧   if (Numerisation > 359) Numerisation = Numerisation -360; }
```

suivent car il est inutile de les tester. La ligne ⑥ traite le cas 2 et la ligne ⑦ l'intervalle 4. Cette ligne ⑦ peut engendrer un petit aléa. Supposons que la valeur mesurée CAP_construit conduise à une valeur de Numerisation égale à 345° par exemple. En ajoutant 29° on obtient 374° et l'on déborde des 359° correspondants à la rose des CAP. La ligne ⑧ corrige ce petit écart de conduite. En retranchant 360 on obtient 14° qui correspond bien au CAP indiqué sur une plage de 0 à 359°.
L'ensemble de ces instructions ajoutées consomme 310 octets. On a dilapidé dans cette séquence tout le bénéfice des économies passée. Peu importe, dans l'état actuel du développement il reste encore 1974 octets de disponibles, largement de quoi parer bien des problèmes.
Pour conclure, ce qui semble le plus intéressant à souligner, c'est la façon dont on a traité par approximation un problème qui mathématiquement s'avérait assez indigeste. Aussi, bien plus souvent qu'on n'y pense, ce type de transposition par linéarisation peut nous sortir élégamment de sévères impasses, et ce d'autant plus que les précisions que l'on programme dans nos applications sont souvent exagérées. Un programmeur averti en vaut deux ...

46) 23/12/2017 : Stabilisation gyroscopique de JEKERT (MJD 58110)

Samedi qui approche du réveillon de Noël, tous les personnels sont libérés pour préparer cette soirée si chaleureuse. Les magasins sont bondés, car tous celles et ceux qui ont tardé à faire leurs achats se bousculent entre les étalages. Comme beaucoup, complètement pris par le développement du projet, nous avons repoussé jusqu'à la dernière minute nos emplettes de fin d'année. Et là, juste en face de nous, trône sur l'une des plus belles étagères de la boutique le rêve de notre vie robotiquoloisir. On vérifie sur l'étiquette. Et oui, c'est bien écrit "Module de stabilisation gyroscopique". Avec un peu de réticence on constate que cette petite merveille est vendue au prix de 504 octets. Comme dans notre bourse il y a encore 1974 de ces précieux billet, le calcul est vite fait. Si on prend ce joli paquet, il ne restera plus que 1470 cellules binaires dans notre porte-monnaie.

L'image est belle sur la boiboite logicielle. On imagine JEKERT posée sur un plateau en bois que l'on incline dans toutes les directions. Immédiatement les membres de la petite machine s'animent et rapidement elle se stabilise, le plateau de la centrale MPU6050 restant à l'horizontale. C'est ça la stabilisation gyroscopique. On demande un défilement permanent des données gravitationnelles. Si sans stabilisation on incline le plateau, Tangage et Roulis changent de valeurs et adoptent des grandeurs positives ou négatives. Avec une commande spécifique à un caractère on valide l'auto-stabilisation. Chaque changement d'inclinaison anime les servomoteurs, et rapidement Tangage et Roulis reviennent à l'inclinaison zéro. C'est magique ... mais 504 octets !

➢ Principe de la stabilisation gravitationnelle.

Hooooo, et puis mince alors, c'est Noël, on peut s'offrir cette petite folie. Il reste encore largement de quoi parer des problèmes logiciels éventuels, sans compter que ne pas utiliser une centrale inertielle pour ce type d'automatisation, ce serait quasiment Hors propos. C'est bon, on se laisse tenter et le démonstrateur P17_Stabilisation_gyroscopique_T5.ino est à nous pour la somme annoncée sur l'étiquette. La notice indique que pour valider ou annuler ce mode la commande à un caractère "_*" a été réservée. Attention, il s'agit du "souligné" et non du moins.

Ce caractère de soulignement nommé aussi "Underscore", et propre à l'avènement informatique, a été choisi car il suggère l'horizontalité de la plateforme inertielle. Il sera ainsi relativement facile à mémoriser, tout au moins pour celles et ceux qui en resteront au pilotage direct à partir de la ligne série USB du P.C. *(Moniteur de l'**IDE**.)*

```
i* >
-------------------------
Bouclier au sol : OUI
LASER actif :  NON
Phares actifs : NON
CAN Neutralisee : NON
Conversion CAN  = 6 (0)
Bloquer les niveaux : NON
Valeur numerique = 6
Eclairage = 6
LASER = = 6
Sommeil : NON
Tension Moteurs : 1.63V
Torsion = NON
Moteurs OFF : NON
Manuel : NON
STB Mvts Rapides
Dernier PGM actif : 1
-------------------------
!PGM i OK!
```

Fig.215

Actuellement, il est difficile d'ajouter une LED de signalisation sur le matériel. Pas impossible, certes, puisque la broche analogique **A1** reste disponible. Tout redémonter, modifier le circuit imprimé principal, trouver une place pour qu'elle soit visible ...

- Môamôa pas du tout envie !

Suite à diverses tentatives de réutilisation d'une LED déjà sur le circuit imprimé, la simplification informatique a d'emblé éliminé S12 à S15 du multiplexeur. L'expérience a montré que passer en double signalisation l'une des autres sources lumineuses n'était pas idéal car trop gourmand en octets pour différencier les deux cas d'éclairement possibles et leur éviter de se contrarier. Par ailleurs, l'allumage pouvant signifier deux états différents, l'interprétation était forcément ambigüe. Au final, une solution simple a émergé : Faire changer la fréquence de clignotement de la boucle de base. Le ***Mode stabilisation gyroscopique*** se superpose à toutes les postures et peut dans certains cas diverger. Aussi, il faut impérativement qu'il soit "très présent" en visualisation. Si on active la gyrostabilisation, la LED verte de "la boucle de base" clignote très rapidement. "Impossible de l'oublier". Dès que ce mode est invalidé, elle reprend sa cadence normale signalant que la boucle de base "tourne comme une montre".

Télémesure oblige, quand la sonde sera sur Mars, il ne sera plus possible d'en observer les témoins lumineux. Il importe de pouvoir en permanence se faire indiquer l'état complet de la machine. Aussi, quand avec la commande "i*" on télécommandera la transmission d'un état les lieux, si le ***Mode stabilisation gyroscopique*** est actif, ce sera précisé comme montré sur la Fig.215 par ajout de l'indication "STB" en début de la ligne d'état des célérités servomoteurs.

➢ *Erreur tragique, le bug bien camouflé ...*

Galère de galère, arriver à faire fonctionner cette scongregneugneu de stabilisation n'a pas été immédiat. Et pourtant sur le papier l'étude préalable était soignée. Les essais se succédaient aux essais avec une furieuse tendance à la divergence. Certains moteurs allaient en butée et le bouton de panique s'est avéré d'une mesure d'urgence qui statistiquement dépassait largement l'usage habituel. La cause de cette difficulté résidait dans la Fiche n°10 particulièrement utile quand on étudie de nouvelles postures ou mouvements coordonnés. Lors de la rédaction de ce type de document, on ne frappe pas au clavier toutes les informations textuelles. Le Copier/Coller fait gagner un temps considérable. Mais ce travail routinier engendre de l'entropie ... avec inversion *(Voir la Fig.216)* de deux textes pour le moteur n°**3**. L'analyse du ***Mode stabilisation gyroscopique*** utilisait ces deux éléments, et piloter un moteur dans le mauvais sens augmentait l'inclinaison, avec "correction infinie". Pas question de laisser trainer une telle erreur. Aussi, dans le petit "paquet" des diverses fiches qui accompagnent ce **TOME5** vous trouverez un "clone corrigé" de cette Fiche n°10.

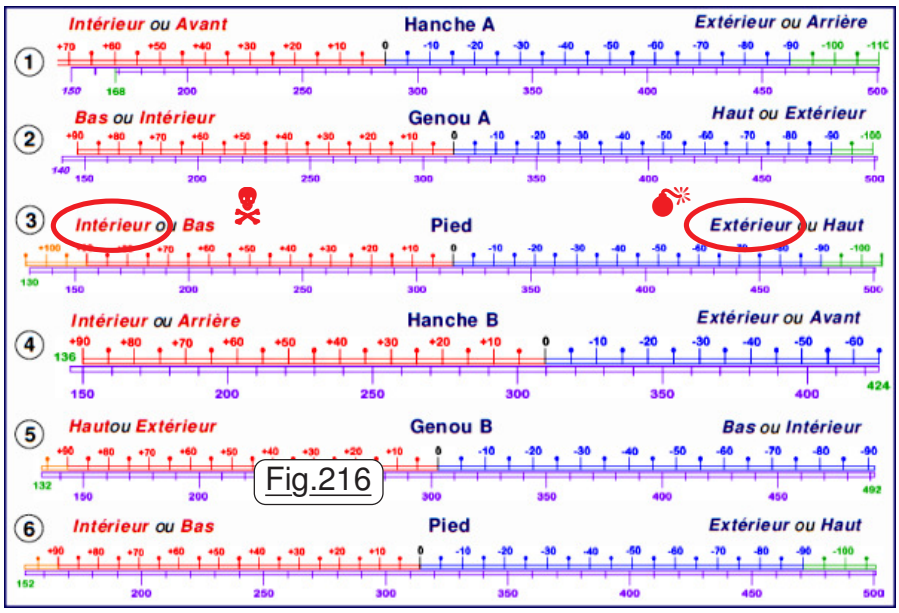


Fig.216

On peut repartir sur de nouvelles bases, le raisonnement qui en dérive redevient fiable.

➢ Principe de la stabilisation en roulis.

Partant d'une posture qui généralement ressemblera à ***Stable Transversal***, on comprend fort bien que la surface de sustentation étant rectangulaire, les comportements en Roulis et en Tangage seront bien différents. Le traitement qui sera effectué devra donc s'adapter à cette caractéristique morphologique de JEKERT. Dans le fonctionnement de la boucle de base, les deux axes sont traités simultanément. Néanmoins nous allons étudier séparément les deux directions.
Sur la Fig.217 la sonde est vu de l'arrière en regardant vers l'avant, et l'on suppose qu'elle est inclinée vers bâbord, le coté le plus bas étant donc à gauche. Dans ces conditions, la valeur retournée par le MPU6050 est positive pour le roulis. Pour compenser l'inclinaison, on va se contenter de faire tourner les Tibias dans le sens des flèches roses, ce qui va faire descendre les chaussettes des griffes de bâbord et monter celle de tribord. *(Flèches bleues.)* Sur la Fig.217 on a repéré les numéros des moteurs concernés. Ce comportement va se continuer jusqu'à ce que la centrale gyroscopique mesure un Roulis nul. La petite machine se retrouvera dans la configuration de la Fig.218 sur laquelle on constate que la largeur entre les divers points d'appui a tendance à légèrement augmenter. Ce phénomène génèrera un peu de frottement, mais surtout est favorable à la stabilité. La machine voit également sa hauteur diminuer, facteur également favorable au non basculement. Quand le point de contact **C** entre la chaussette et le sol dépasse la verticale du genou **G** la hauteur à bâbord se remet à diminuer. Cependant à tribord la hauteur diminue plus rapidement, et la correction continue normalement. La limite de l'inclinaison correspond à la posture pour laquelle la chaussette de la Jambe **A** vient en contact avec le sabot du bouclier. *(Voir la Fig.219 et **5** sur la Fig.220.)* Ce cas correspond à une inclinaison latérale de 9° environ. Profitons de la Fig.220 qui montre la petite machine de face posée sur une pente **α** inclinée à 9° sur tribord pour faire le point. Pour cette photographie c'est l'inverse de l'inclinaison étudiée sur les dessins. Issu du connecteur HE14 situé dans la région cachée **R** sur le dessus du circuit imprimé du condensateur de 470µF, la ligne d'alimentation de puissance traverse la machine pour ressortir à bâbord en **9**. Puis ce toron part en **6** passant vers l'arrière entre le bouclier et le circuit imprimé principal. On reconnait facilement en **4** le LASER alimenté par la petite ligne **1**. En **7** on repère l'un des sabots du bouclier avec en **8** l'un des deux sabots pour l'anti-basculement. Pour maintenir bien à l'horizontale le circuit imprimé **15**, donc la centrale gyroscopique, le servomoteur du Genou **2** a fait tourner le Tibia **3** vers le bas. Au

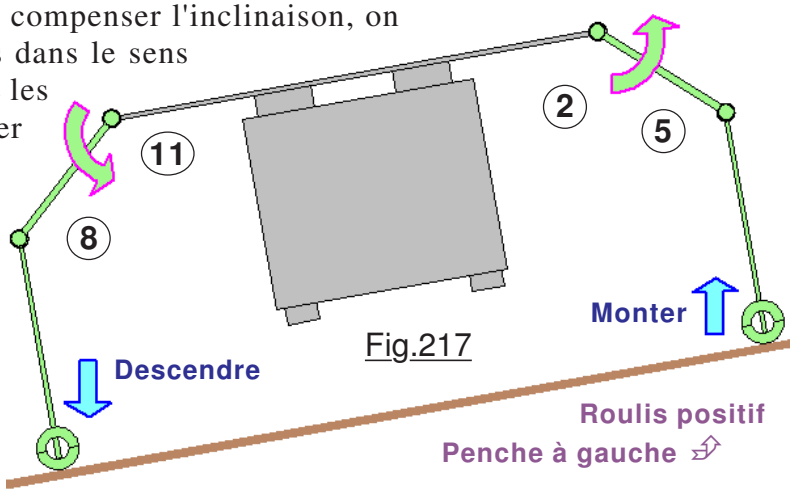


Fig.217

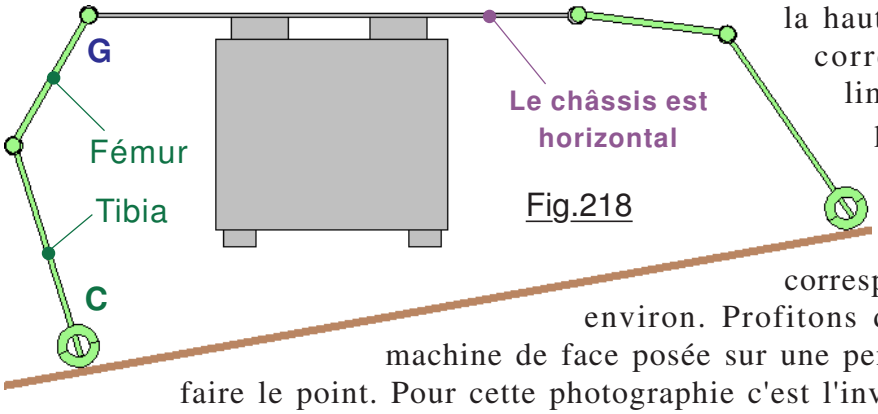


Fig.218

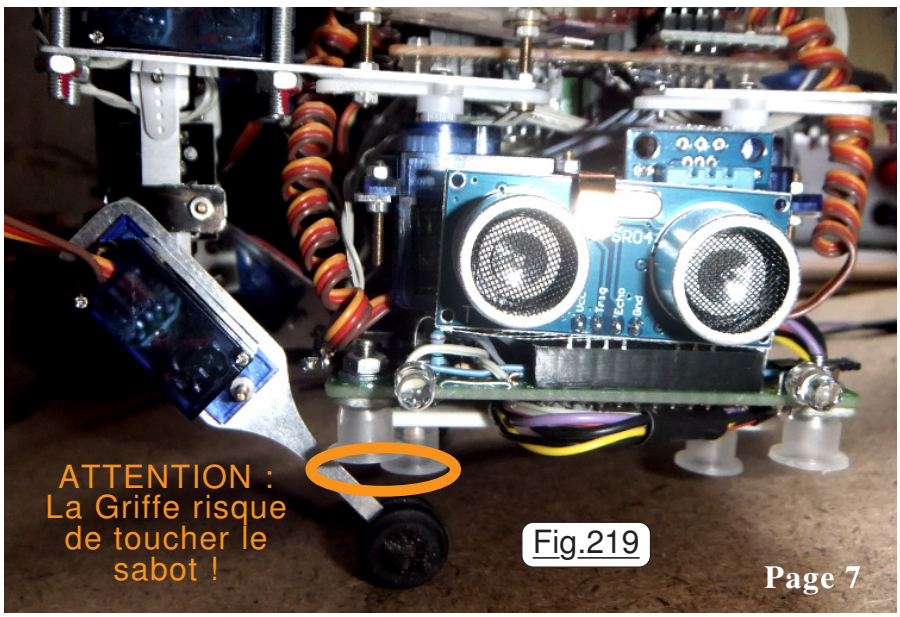


Fig.219

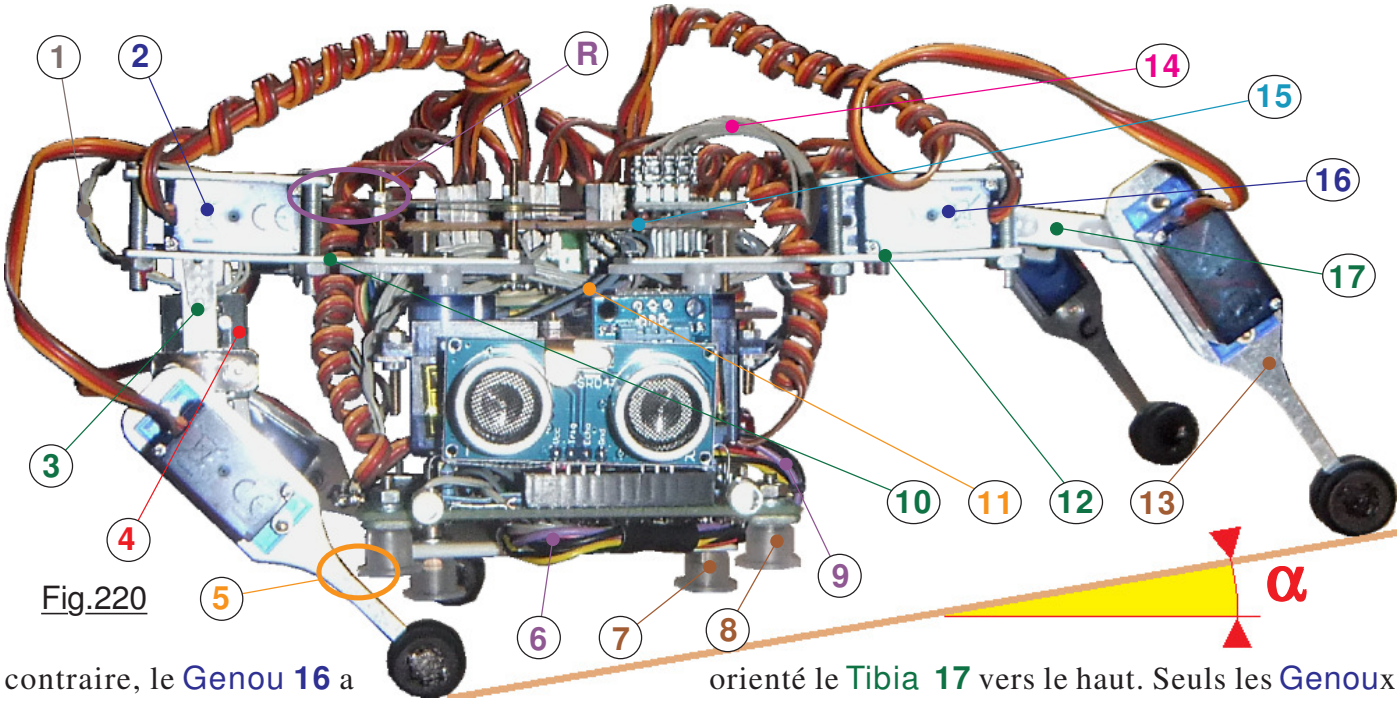


Fig.220

contraire, le Genou **16** a orienté le Tibia **17** vers le haut. Seuls les Genoux tournent sur les servomoteurs, ce qui fait que les angles d'orientation entre les Tibias tels que **17** et les Griffes telles que **13** ne changent pas. Sur cette photographie la posture est presque en limite. Dans la zone **5** le contact entre la Griffe et le sabot du bouclier est presque établi, le moteur du Pied serait alors en situation de forçage. Quand on incline de plus en plus le support sur lequel repose la sonde, il est assez amusant de voir JEKERT se déformer latéralement vers le bas de la pente et se tasser vers le sol. Le circuit imprimé **15** reste parfaitement à l'horizontal. En **11** l'on observe le toron torsadé de la ligne qui réunit la centrale inertielle au circuit imprimé principal. Grande faiblesse de cette petite machine : Le manque manifeste de rigidité de certaines pièces mécaniques, tout particulièrement les Hanches qui supportent dans de mauvaises conditions le poids de JEKERT. On voit très bien en **10** et **12** que les deux Fémurs fléchissent comme les ailes d'un avion.

➢ *Principe de la stabilisation en tangage.*

Contrairement au Roulis, pour le Tangage il n'a pas été possible d'éviter des mouvements coordonnés, car tenter de n'agir que sur les Fémurs manque totalement d'efficacité, et pour arriver à compenser une faible inclinaison la machine se trouve complètement "écartelée". Du coup on ne peut corriger que de faibles inclinaisons, et surtout accompagnées de frottement de glissement inacceptable. Géométriquement, pour du roulis on monte ou l'on descend les chaussettes "en parallèle" de chaque coté. Si gauche monte, alors droite descend. Pour du tangage la procédure sera analogue, mais c'est devant et derrière que les déplacements auront des sens identiques. Par exemple si l'animal robotisé est cabré comme sur la Fig.221, les deux Griffes de devant doivent monter et celles de derrière descendre pour faire tourner le châssis comme montré par la flèche rose. Tous les dessins proposés dans ce chapitre sur le ***Mode stabilisation gyroscopique*** sont très schématisés mais ***réalisés à l'échelle. Les proportions des membres sont respectées, ainsi que la position des articulations***.

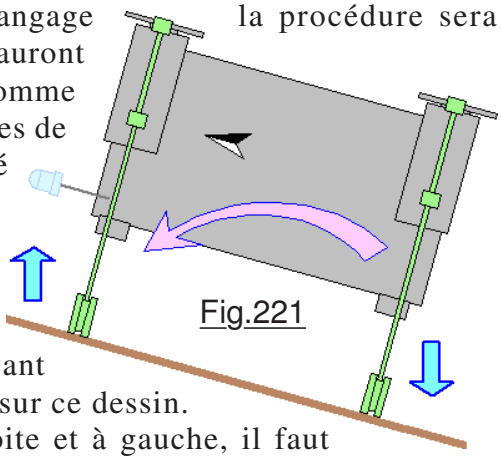


Fig.221

Considérons la Fig.222 sur laquelle la sonde est vue de devant et nous "regarde". C'est le train avant qui est représenté sur ce dessin. Si on désire baisser les deux chaussettes simultanément à droite et à gauche, il faut engendrer les rotations **R** sur les Genoux. Les extrémités des Tibias qui matérialisent les Pieds montent en **M**. Par déplacement réciproque on fait Descendre l'avant de la sonde, mouvement constituant précisément le but de cette manœuvre. La rotation **R** provoque une translation latérale **T** du Pied qui déporte aussi l'extrémité du Tibia vers l'extérieur. En sois ce petit déplacement n'est pas dramatique. Par contre, la rotation **R** ramenée en extrémité de Griffe engendre un écart latéral considérable. La machine est "écartelée et peut s'effondrer sous l'influence des efforts de

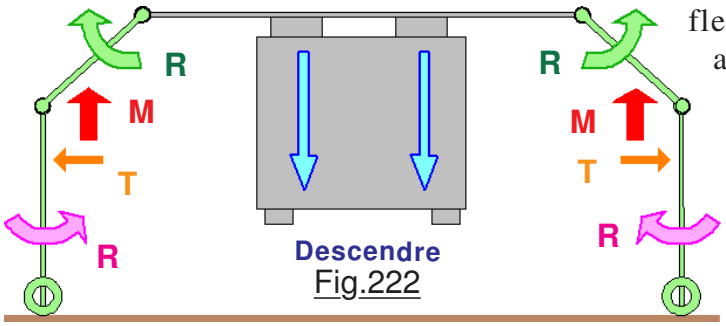


Fig.222

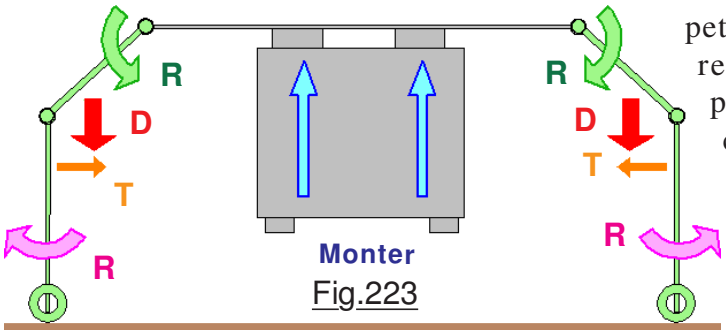


Fig.223

flexion. C'est pour parer ce problème, que nous allons simultanément coordonner une rotation inverse **R** de la Griffe. L'idéal pour éviter tout frottement, et donc de l'usure par abrasion, consisterait à annuler tout déplacement au sol de la chaussette. Dans la pratique, pour minimiser le code objet on se contente d'une approximation. Du coup, au cours de la stabilisation en Tangage on observera de petites variations de l'empattement, ces dernières restant tout à fait acceptables. Pour maintenir parfaitement constante la distance qui sépare la chaussette du plan médian de la machine il faudrait coordonner **R** et **R** par des amplitudes de consignes angulaires qui relèvent d'un calcul trigonométrique goinfre en octets de code machine. On se contente de faire varier linéairement les consignes qui traitent les rotations **R** et **R**. Avec la Fig.222 nous avons vu comment faire Descendre le nez de la petite machine. Si on doit le faire Monter, car la machine au lieu d'être cabrée s'est engagée dans une pente descendante, on fait l'inverse sur les rotations et provoquer de ce fait les descentes **D**. Naturellement, si au lieu de vouloir faire monter ou descendre l'avant de la machine on veut obtenir cet effet sur l'arrière, on procède exactement pareil mais sur les servomoteurs des Jambe **B** et **C** au lieu des membres sur **A** et **D**. Sur la photographie de la Fig.224 la sonde repose sur le plateau **P** dont l'inclinaison est maintenue au moyen de la cale **X**. On constate que l'amplitude de la stabilisation est considérable puisque sur cette vue la pente est d'environ 30°. Cette efficacité résulte de la distance relativement faible qui sépare le train avant du train arrière.

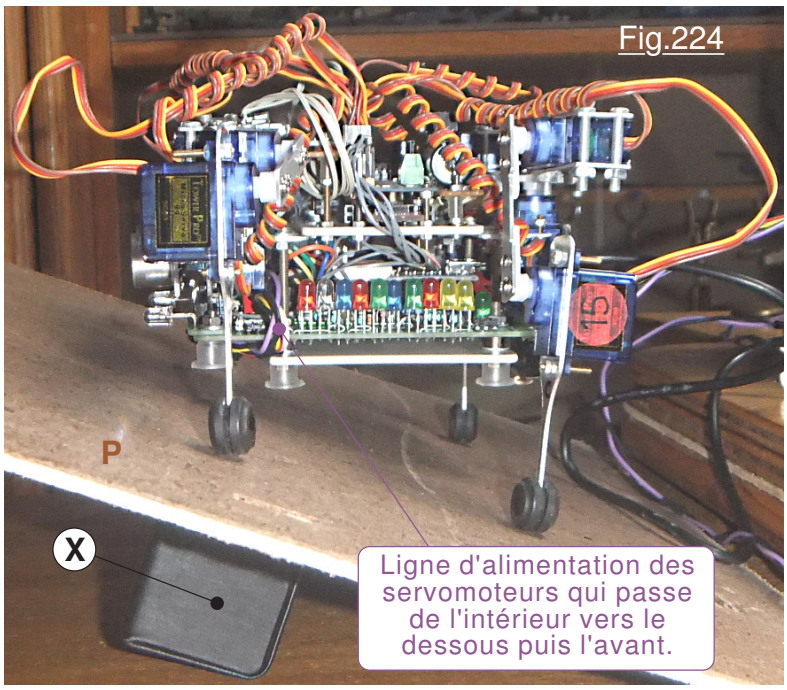


Fig.224

Dévoilant une foule de petits détails, la Fig.225 montre la sonde vue de dessus quand elle est en posture de stabilisation gyroscopique. Conformément aux explications relatives à la correction d'assiette en Tangage, pour Descendre les Griffes à l'arrière les Fémurs tournent vers le bas. On remarque bien sur la photographie Fig.225 que les Fémurs sont pratiquement verticaux. La rotation **R** compense un peu trop **R** ce qui engendre un glissement de la chaussette **G***int* vers l'intérieur. Si l'on adopte un support pas trop rugueux, ce glissement reste toutefois acceptable. Du reste, lorsque la sonde s'est stabilisée en attitude horizontale, on peut déclencher "p14*" pour libérer les efforts. Pour monter les Griffes à l'avant les Fémurs tournent vers le haut écartant les Genoux vers l'extérieur. La rotation **R** ne compense pas assez **R** ce qui engendre un glissement de la chaussette **G***ext* vers l'extérieur. On voit parfaitement sur cette image que l'animal mécanique est plus écartelé à l'avant qu'à l'arrière. *(Observation de la position des appuis au sol.)*

Pour réduire ces déplacements parasites, il a été tenté d'établir une meilleure proportionnalité dans les deux mouvements coordonnés. La technique a consisté à doubler la variation de consigne sur l'élément trop "lent". Les dérapages au sol étaient effectivement bien moindres. En contre partie, la flexion des membres associée à des accélérations plus importantes généraient des surcompensations se traduisant par une oscillation entretenue infiniment plus néfaste que le petit

inconvénient du frottement au sol des chaussettes. La photographie de la Fig.224 démontre que l'efficacité de la correction en Tangage est impressionnante puisqu'autant en cabrage qu'en piqué on peut compenser facilement des pentes de 30°. Cette particularité s'explique par le fait que la distance **D** qui sépare les appuis avant des appuis arrière est relativement faible. Du coup un différentiel de hauteur entre les deux trains se traduit par une variation en tangage importante.

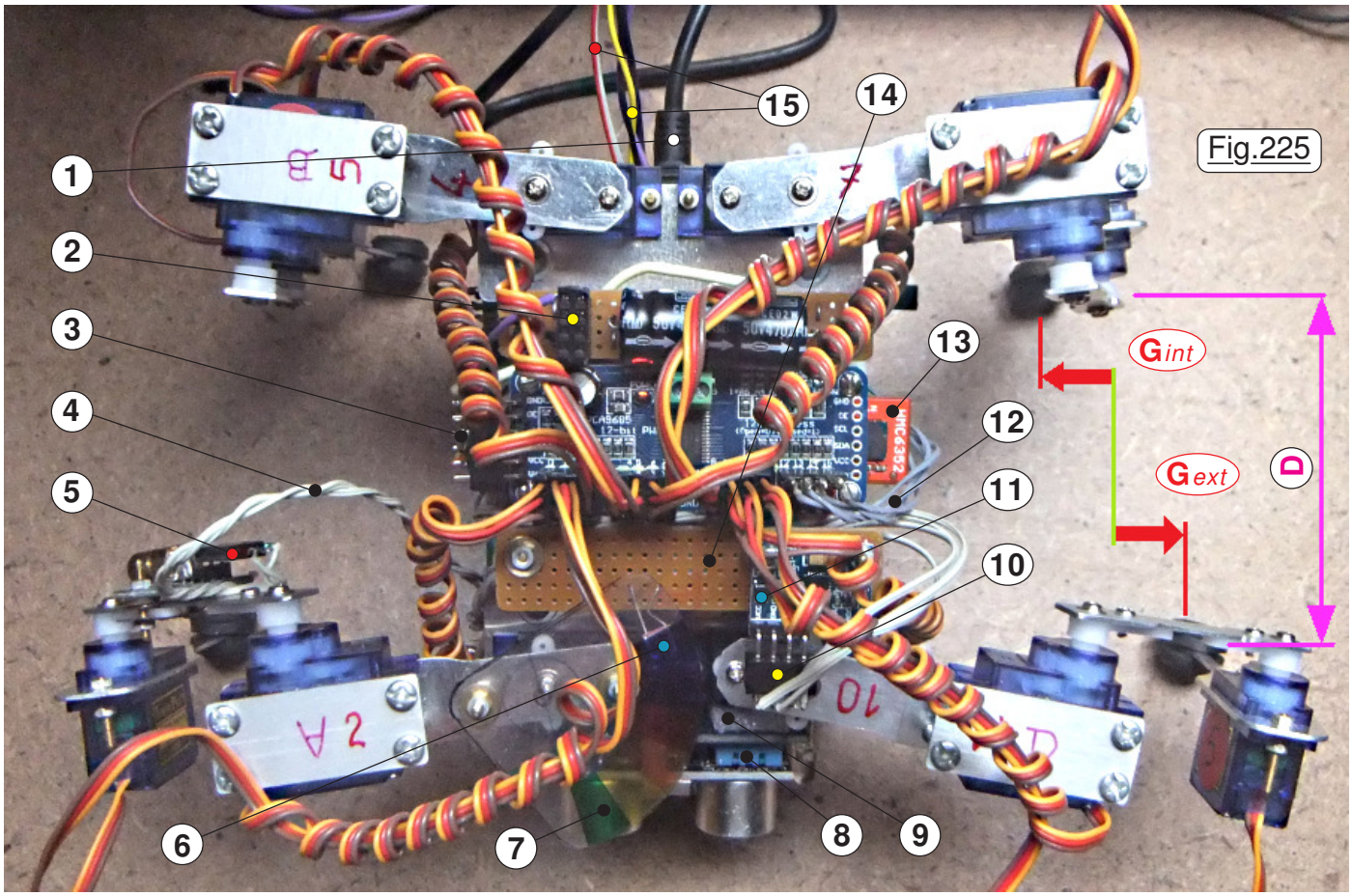


Fig.225

Revenons à cette photographie pour observer une fois de plus la petite machine dans sa version complète. En **1** le cordon USB passe entre les deux moteurs arrière et va vers le P.C. pour un pilotage par le moniteur de l'**IDE**. En **2** on distingue le connecteur HE14 double qui branche la ligne d'alimentation des moteurs sur le circuit imprimé du condensateur de 470µF avec en **3** le connecteur HE14 reliant le multiplexeur au circuit imprimé principal. La ligne **4** alimente le LASER **5**. Discrète, en **6** la cellule photorésistante se cache sous le filtre coloré **7**. Le capteur d'hygrométrie **8** est poussé vers l'arrière par le module du télémètre et s'appuie sur le petit bloc de mousse synthétique **9**. Le connecteur HE14 en **10** est branché sur la centrale gyroscopique **11** sa ligne de fils torsadés gris arrive du dessous du circuit imprimé **14**. Venant du circuit imprimé principal, la ligne **12** relie les sorties S12 à S15 du multiplexeur. Bien visible en **13** le module de la boussole statique dépasse un peu du multiplexeur. Enfin en **15** s'éloigne de la sonde par l'arrière le cordon ombilical de pilotage qui ira à la raquette de commande ainsi que la ligne d'alimentation en puissance des servomoteurs.

➢ *Séquences qui traitent la stabilisation gyroscopique.*

Fonction imposant une correction "en temps réel", on se doute que leur appel sera effectué dans la boucle de base du programme void **loop** () tout en assurant la prise en compte d'une consigne éventuelle de dialogue Homme / Machine arrivant sur la ligne série USB. On constate sur le listage proposé ci-dessous que dans la boucle de base on ne développe pas le code, on se contente de faire appel une procédure spécifique pour que le programme soit plus lisible :

```
//--------------- Gère la stabilisation Gyroscopique ---------------
 if (Stabilisation_gyroscopique) {Stabilisation_GYROSCOPIQUE();} }
```

Si Stabilisation_gyroscopique vaut true, il y a appel à Stabilisation_GYROSCOPIQUE(). On peut noter au passage que l'identificateur du booléen et le nom de la procédure sont identiques. Le compilateur fait la différence entre lettres majuscules et lettres minuscules, c'est un

avantage incontestable car on peut comme ici choisir des identificateurs parlants et assurer leur différenciation par le compilateur. La prise en compte d'une consigne sur un caractère "_*" se contente d'inverser l'état du booléen testé en permanence dans la boucle de base :

```
case '_' : {Stabilisation_gyroscopique = !Stabilisation_gyroscopique; break;}
```

L'écriture initiale des séquences de coordination des mouvements de stabilisation consiste à agir sur quatre moteurs pour le Roulis, et sur huit moteurs pour le Tangage. Pour chaque servomoteur, on va augmenter ou diminuer d'une unité sa valeur de consigne mémorisée dans le tableau dédié et constamment mis à jour Configuration_actuelle[Sortie]. Comme chaque mouvement élémentaire revient à agir sur une sortie en augmentant ou en diminuant de un sa valeur de consigne, il devient particulièrement rentable de créer une procédure Stabilise() recevant les paramètres nécessaires :

```
    //========== Routines pour la Stabilisation GYROSCOPIQUE ==========
① void Stabilise(boolean Plus, byte Sortie) {
②   Consigne_stabilisation = Configuration_actuelle[Sortie];
③   if (Plus) Consigne_stabilisation++; else Consigne_stabilisation--;
④   Mouvoir(Sortie,Consigne_stabilisation);}
```

Bien qu'elle effectue un travail considérable, la routine est presque élémentaire à comprendre. À son appel en ① on lui communique dans le booléen Plus le signe de la variation désignée pour le mouvement. Il est facile de deviner que le byte Sortie sert à indiquer à cette procédure sur quel servomoteur elle doit agir. En ② on récupère dans la variable Consigne_stabilisation la valeur actuelle de consigne pour la Sortie concernée. En ③ on va incrémenter ou décrémenter d'une unité la valeur de Consigne_stabilisation en fonction du sens indiqué par le paramètre Plus. Enfin en ④ on modifie la position actuelle du servomoteur piloté par Sortie, sachant que la procédure de servitude Mouvoir() ne le fait que si les moteurs ne sont pas figés avec"f*"et dans ce cas met à jour la valeur de la consigne dans Configuration_actuelle[Sortie].

```
 void Stabilisation_GYROSCOPIQUE() {
①  Mesure_les_valeurs_de_gravitation();
②  if (Roulis > 0) {
③     Stabilise(true, 1); Stabilise(false, 4); Stabilise(false, 7); Stabilise(true, 10);}
④  if (Roulis < 0) {
     Stabilise(false, 1); Stabilise(true, 4); Stabilise(true, 7); Stabilise(false, 10);}
⑤  if (Tangage > 0) {
⑥     Stabilise(true, 1); Stabilise(true, 2); Stabilise(true, 4); Stabilise(true, 5);
⑦     Stabilise(false, 7); Stabilise(false, 8); Stabilise(false, 10); Stabilise(false, 11);}
⑧  if (Tangage < 0) {
⑨     Stabilise(false, 1); Stabilise(false, 2); Stabilise(false, 4); Stabilise(false, 5);
⑩     Stabilise(true, 7); Stabilise(true, 8); Stabilise(true, 10); Stabilise(true, 11);} }
```

Réaliser la stabilisation consiste dans la procédure invoquée à chaque rotation de la boucle de base à ***mesurer en ligne*** ① ***les angles d'inclinaison*** Roulis et Tangage puis traiter en cascade les deux corrections ***si elles sont différentes de zéro***. Pour chaque déviation le programme détermine dans quel sens le machine penche. Pour le Roulis, si en ② la machine penche à gauche, alors on effectue une correction élémentaire en ③. Seuls dans ce cas les sorties 1, 4, 7 et 10 sont concernées. Si la sonde penche à droite, ④ va effectuer un traitement absolument identique en changeant uniquement les sens de rotation sur les moteurs. Pour le Tangage, ce sont les lignes ⑤ et ⑧ qui déclencheront une correction élémentaire. Supposons que JEKERT soit engagée sur une pente descendante. Piquant du nez, la valeur de Tangage sera négative. Les lignes ⑨ et ⑩ seront exécutées. Pour mieux comprendre les valeurs des booléens adoptées, considérons la Fig.226 qui symbolise le bouclier vu par dessus. Nous savons que les corrections en Tangage imposent des mouvements coordonnés sur deux moteurs par Jambe. Les n° des moteurs sont repérés en noir dans les petits cercles avec leurs sorties respectives en violet. Pour redresser la plan inertiel il faut faire Monter l'avant et Descendre l'arrière du robot. Naturellement on en déduit qu'il faut **D**escendre les Fémurs devant et les faire **M**onter derrière. Pour réduire les glissements des chaussettes sur le sol il faut faire tourner les Tibias vers l'*Extérieur* devant et vers l'*Intérieur* derrière. Tous ces détails sont résumés sur la Fig.226 sur laquelle est indiqué en couleur verte le sens dans lequel il faut modifier

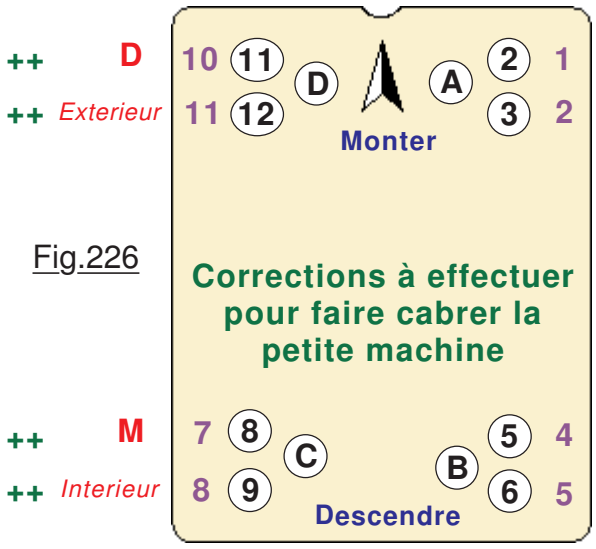


Fig.226

chaque Consigne_stabilisation. Pour déterminer s'il faut incrémenter la consigne avec ++ ou au contraire la diminuer avec l'instruction -- on va s'aider de la Fiche n°10. *(Attention, il importe de prendre celle qui est corrigée !)* Par exemple le Genou **2** en sortie **1** doit faire **D**escendre le Tibia donc aller vers le Bas. Quelle que soit la valeur de sa consigne actuelle on devra tendre vers la valeur 140 ***la plus faible***. Il faut ***donc décrémenter*** avec --. Pour sa part le Pied **3** en sortie **2** doit faire tourner la Griffe vers l'*Extérieur*. Quelle que soit la valeur de sa consigne actuelle on devra tendre vers la valeur 130 ***la plus faible***. Il faut ***donc décrémenter*** avec --. Pour ce qui est des mouvements à gauche on doit produire les mêmes effets, cependant le sens de variation des consignes est différent. Par exemple le Genou **11** en sortie **10** doit faire **D**escendre le Tibia donc aller vers le Bas. Quelle que soit la valeur de sa consigne actuelle on devra tendre vers la valeur 470 ***la plus forte***. Il faut ***donc incrémenter*** avec ++. Pour sa part le Pied **12** en sortie **11** doit faire tourner la Griffe vers l'*Extérieur*. Quelle que soit la valeur de la consigne actuelle on devra tendre vers la valeur 500 ***la plus forte***. Il faut ***donc incrémenter*** avec ++. Il sera facile, Fiche n°10 corrigée en main, d'effectuer ces raisonnements pour tous les cas de figure. Ceci dit, quand on a analysé avec soin pour une action à cabrer, pour un Tangage positif il suffit d'inverser la valeur de tous les booléens et en déduire les lignes d'instruction ⑥ et ⑦ du test effectué en ligne ⑤ pour agir à piquer.

Dans le démonstrateur P17_Stabilisation_gyroscopique_T5.ino pour un petit surcout de quatre octets, on peut annuler le mode "Listage gyro en continu" quand on passe en mode Sommeil avec "s*" ou lorsque l'on quitte le programme avec la commande "q*". Cette petite amélioration n'a pas été introduite dans P30 car elle y ajoute dix octets, considéré alors comme du gaspillage.

➢ *Intégration de la stabilisation gyroscopique à P30.*

Avant d'aborder le vif du sujet, c'est à dire débuter les analyses pour un pupitre de commande autonome, on va dans ce chapitre en terminer avec P30_Programme_COMPLET_T5.ino et en faire un ***logiciel complet qui intègre vraiment tous les derniers développements***. En effet, le programme P30 du **TOME 4** n'émulait pas le spectroscope colorimétrique qui est ajouté à la version du **TOME 5**. Cette fonction ajoutée consomme 206 octets. Bien entendu, on ajoute également la stabilisation gyroscopique. Dans l'état final du programme "complet archi complet" l'augmentation du code objet est de 512 octets. Il faut toutefois préciser qu'un changement de stratégie est également intervenu. À partir de cette version du logiciel, l'appel à la fonction Libere_efforts() n'est plus effectué à chaque mouvement élémentaire mais uniquement avec "p14*", et quelques programmes spécifique comme REVEILLER, retour de hauteur maximale etc. Cette modification a été ajoutée car invoquer Libere_efforts() après chaque mouvement élémentaire pénalisait trop en durée les déplacements avec répétition. Par ailleurs, la machine n'étant pas assez rigide, les flexions parasites contrecarrent notablement Libere_efforts() qui par voie de conséquences manque d'efficacité.

Dans l'état actuel du logiciel complet il reste 1256 octets de disponibles s'il faut modifier le programme. Cette marge est satisfaisante pour envisager sereinement la version qui dialoguera avec la raquette de commande. Il faut s'attendre à des modifications "boulimiques", mais elles seront probablement en partie compensées par le fait que toutes les procédures de "bavardages" sur la ligne série de dialogue avec le Moniteur de l'**IDE** vont se voir simplifiées car les affichages pour les opérateurs de maitrise seront à la charge du logiciel installé sur le pupitre. Il est temps de débuter les études visant à faire dialoguer les deux cartes Arduino NANO.

CONCLUSION : Le programme P30_Programme_COMPLET_T5.ino constitue la **version ultime à utiliser** par celles et ceux qui pour le moment veulent en rester à la machine pilotée par l'ordinateur au moyen du dialogue par l'entremise de la ligne série USB du Moniteur de l'**IDE** et du cordon USB de la carte Arduino NANO.

47) 26/12/2017 : Dialogue entre machines (MJD 58113)

Oublié le réveillon, soirée en famille qui a redynamisé les équipes. Il était temps, car les semaines passées ont été rudes, un peu de repos était devenu indispensable. Alors, se retrouver en famille à proximité du beau sapin relevait autant d'une pause médicale que d'une tradition séculaire. De retour en salle informatique S4, le cliquetis des claviers d'ordinateurs a réinvesti le bruit de fond discret de d'espace de travail encombré de consoles informatiques. Les ingénieurs s'affairent, car ils vont développer du conversationnel entre deux blocs de silicium et ils savent pertinemment que ce n'est pas gagné d'avance ...

Faire dialoguer deux machines n'est absolument pas élémentaire. Contrairement aux bavardages humains pour lesquels les messages sont transportés par l'air ambiant, sur des machines ce sont des impulsions binaires qui se succèdent à cadences élevées sur les lignes filaires de communication. Dans notre cas ce seront **TX** et **RX** sur **D0** et **D1** qui seront affectées à cette fonction palabresque. Non seulement pour se comprendre les deux machines devront échanger des propos

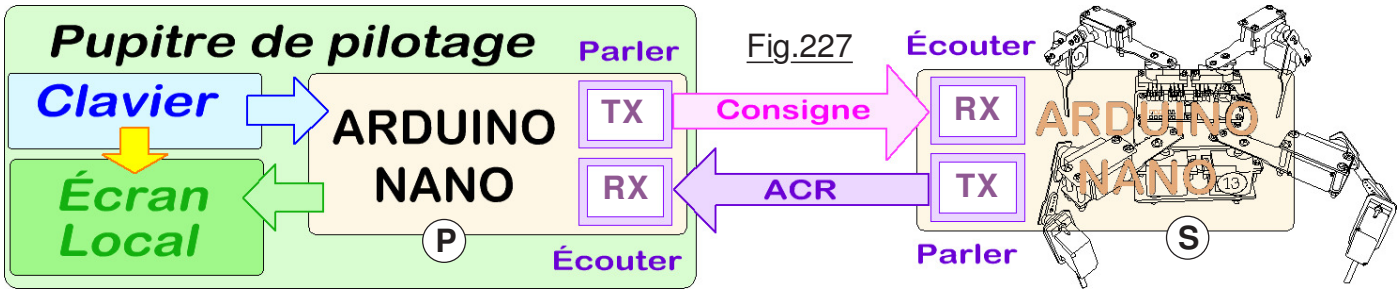


Fig.227

par le biais de protocoles très précis, mais surtout des problèmes de synchronisation sérieux pourront rompre et bloquer "les discutions". La Fig.227 résume l'architecture globale qui présidera la conduite des études indigestes à venir.

➢ Dialogue Homme / Machine : Le retour.

Globalement nous allons retrouver les préceptes qui dominaient le dialogue entre le P.C. et JEKERT. La grande différence réside dans le fait que maintenant c'est une carte Arduino NANO qui sera chargée de gérer un Clavier et un Écran local intégrés dans un **P**upitre de commande que l'on nommera aussi ***Raquette de commande***, vocable suggéré par ses dimensions les plus réduites possibles. Décortiquons le principe d'un échange "verbal" entre l'Humain et les deux Machines :

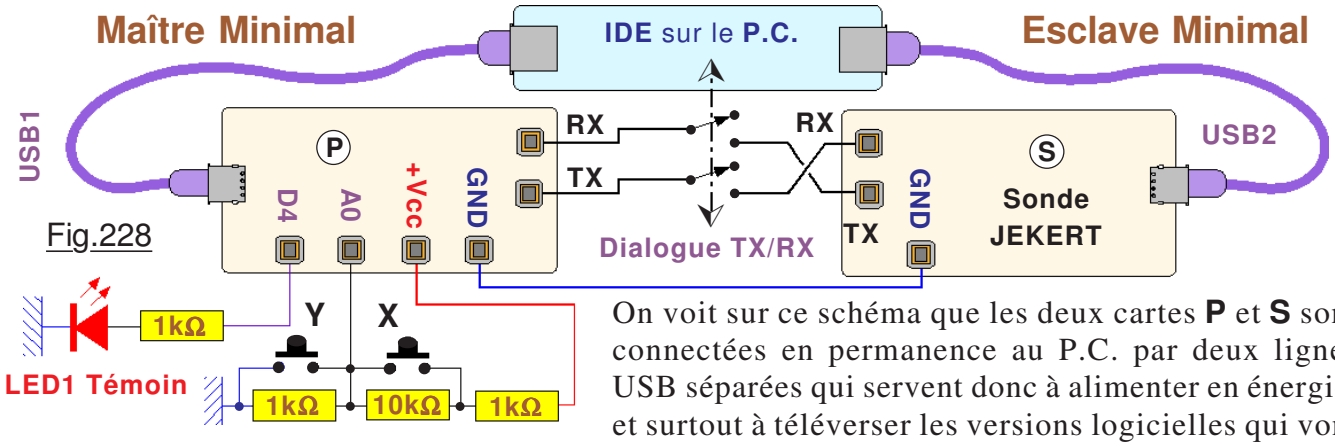
1) Le programme du pupitre **P** est en attente d'une directive issue du *Clavier*, l'information qui en résulte transite comme l'indique la flèche bleue. *(Par exemple on cliquera sur un bouton poussoir.)*
2) Pour aider le technicien qui exploite la sonde, **P** affiche sur l'*Écran local* des informations traduisant les actions de l'opérateur sur le *Clavier*. *(Flèche symbole jaune.)*
3) La directive de l'opérateur est alors traduite en Consigne et **P** va Parler sur **TX**. Puis, comme on va privilégier un **mode alternat**, **P** va immédiatement Écouter sur sa broche **RX**.
4) Le ***logiciel Esclave*** **S** étant à l'Écoute sur sa broche **RX** enregistre le message reçu.
5) L'analyseur syntaxixe décode cette ***Consigne***. Si elle est valide, immédiatement elle est exécutée. Après une éventuelle action valide, **S** ensuite Parle sur **TX** pour **AC**cuser **R**éception.
6) Le ***logiciel Maître*** du pupitre **P** décortique l'**ACR** reçu sur **RX** et rend compte sur l'*Écran local* après une mise en forme, puis immédiatement se replace en surveillance du *Clavier*.

> **NOTE :** À partir d'ici, le mot ***Consigne*** ne désignera plus une valeur de positionnement angulaire pour les servomoteurs, mais un ***ordre à exécuter*** par le ***logiciel Esclave***.

Étant donné que les affichages sur l'*Écran local* sont gérés par la carte **P**, il devient totalement parasite d'encombrer les échanges Consigne / **ACR** par du "bavardage" de présentation claire pour l'humain. Rendre ces "données Machine" aussi compactes que possible aura deux conséquences positives. D'une part les dialogues binaires prendront moins de temps en ligne **TX**/**RX**. D'autres parts le ***logiciel Esclave*** va considérablement maigrir. Il faut donc revoir entièrement les protocoles de dialogue entre les deux ATmega328. Il importe aussi de déterminer quel type de clavier sera utilisé, et choisir un module électronique pour équiper l'*Écran local* de la Raquette de commande. Autant pour le clavier on peut se servir de boutons poussoir provisoires, autant pour l'écran il va falloir porter notre dévolu sur un modèle précis, car le développement sera tributaire de sa présence pour arriver à savoir ce que se disent les deux machines ... espionnage oblige !

➢ Organiser le développement.

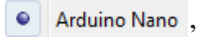
Difficulté nouvelle, deux entités sont concernées à chaque étape des essais. Ce n'est plus un seul démonstrateur qu'il faut mettre au point, mais associer deux programmes qui seront téléversés. L'un sur la carte Arduino NANO **S** de la sonde, l'autre sur un module **P** quelconque placé sur une plaque de contacts à essais, entouré d'une foule d'éléments provisoires reliés par plein plein plein de fils électriques. Entre ces deux circuits ATmega328 on va croiser deux lignes filaires sur **TX** et **RX** puis, mettre à contribution l'**IDE** pour développer les deux démonstrateurs.
Facile à dire ! Dans la pratique, la mise en œuvre se complique car on doit programmer pour **S**, téléverser vers la sonde, puis développer pour **P** et téléverser. Puis, les deux démonstrateurs étant résidants dans les deux cartes, établir les deux liaisons de dialogue et procéder aux essais.
PROBLÈME : La ligne mini USB qui permet de dialoguer avec le P.C. interfère sur les broches **D0** et **D1** et réciproquement. Vu le nombre d'essais que l'on va devoir conduire, on ne va pas passer notre temps à brancher et débrancher des lignes USB, sans compter qu'à la fin, la fiabilité des prises risquerait fort de se voir dégrader. Nous avons beaucoup de chance, car l'**IDE** est conçu pour pouvoir développer plusieurs "Sketch" simultanément. Considérons la Fig.228 qui est dérivée de celle de la Fiche n°31 résumant les techniques à utiliser. Oublions dans un premier temps la façon dont seront alimentées les deux cartes Arduino. Pour le moment ce seront les deux prises USB du P.C. qui fourniront l'énergie aux deux modules, la masse GND étant commune on s'en doutait bien.

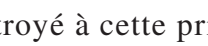


On voit sur ce schéma que les deux cartes **P** et **S** sont connectées en permanence au P.C. par deux lignes USB séparées qui servent donc à alimenter en énergie, et surtout à téléverser les versions logicielles qui vont se succéder. Sur la carte **P** nous avons connecté en **A0** un clavier provisoire à deux boutons **X** et **Y** qui sera suffisant pour conduire les études initiales. Rien n'est figé, mais pour le moment la sortie **D4** est provisoirement utilisée pour visualiser de façon optique les échanges réalisés lors du dialogue.

➢ Réaliser un premier bavardage entre l'Homme et la Machine.

Première manipulation pour tester le dialogue entre les deux machines, nous allons devoir concrétiser l'agencement de la Fig.228 et le mettre en œuvre matériellement. La Fiche n°31 résume les branchements et la façon de réaliser cette première expérience. Toutefois, comme c'est la première fois que vous allez tenter ce type d'expérience, nous allons y aller progressivement. On partira pour cette tentative d'une configuration où aucune des deux cartes n'est reliée au P.C. et l'éditeur de l'**IDE** n'est pas ouvert. En résumé on se trouve en activité hors programmation. Sur la plaquette d'essais le module Arduino, le clavier expérimental **X** et **Y** et la LED rouge sont câblés. L'inverseur d'isolement est également en place avec les lignes **TX** et **RX** branchées.

1) **BASCULER l'inverseur vers IDE sur le P.C.** c'est à dire ouvrir la ligne croisée de dialogue,
2) Brancher la carte Arduino **S** de la sonde sur l'une des lignes USB disponible du P.C,
3) Activer l'**IDE** en ouvrant l'éditeur sur le démonstrateur P19_PGM_Esclave_minimal.ino,
4) Outils > Type de carte > Vérifier 

,
5) Outils > Port > Valider le port octroyé à cette prise USB. Par exemple 

,
6) Téléverser le démonstrateur de la Sonde P19_PGM_Esclave_minimal.ino,
7) Brancher la carte Arduino **P** de la plaque à essais une autre ligne USB disponible du P.C,
8) Activer un clone de l'**IDE** en ouvrant son éditeur sur le deuxième démonstrateur logiciel P18_PGM_Maitre_minimal.ino, car on peut ouvrir plusieurs instances du compilateur simultanément. C'est l'un des points forts de l'environnement d'Arduino.

09) Outils > Type de carte > Vérifier Arduino Nano , *(Rien n'interdit à ce stade d'utiliser une carte UNO par exemple, mais je ne vous le conseille pas. En effet, sur une NANO on ne dispose que de 30720 octets pour le programme, alors que sur UNO il y en a 32256. Autant développer sur la carte cible pour éviter des surprises. De plus la carte UNO ne dispose pas des broches **A6** et **A7**.)*

10) Outils > Port > Valider le port octroyé à cette deuxième prise USB. Par exemple ✓ COM13 ,

11) Téléverser le démonstrateur sur la Sonde P18_PGM_Maitre_minimal.ino,

> *À ce stade nous avons deux compilateurs sur le P.C. chacun pouvant modifier le programme source qu'il traite dans son éditeur de texte. Chaque fois que l'on téléverse le code objet, c'est sur la bonne carte, car il y a automatiquement aiguillage sur la bonne prise USB. On peut maintenant que les deux cartes **P** et **S** sont munies des Sketch associés procéder à des essais :*

12) BASCULER l'inverseur vers **Dialogue TX/RX** c'est à dire établir la liaison croisée de dialogue,

13) En fonction des démonstrateurs téléversés procédez aux essais de validation.

14) **BASCULER l'inverseur vers IDE sur le P.C. pour reprendre l'édition des logiciels**.

Facilité apportée par l'agencement d'une carte Arduino, ***si sur l'une des deux fenêtres du compilateur on valide le Moniteur série, l'écran du P.C. affiche les textes des messages envoyés par le programme en cours d'exécution***. C'est parfois bien commode pour lever un doute. Concernant ce premier essai on valide intégralement la chaîne de dialogue puisque l'on donne une consigne sur l'un des deux boutons poussoir du banc de test, les deux machines dialoguent et le Maître rend compte sur l'écran local qui ici se résume à une LED rouge. La boucle est bouclée ...
Allons voir dans l'éditeur du programme P18_PGM_Maitre_minimal.ino le listage du premier démonstrateur qui va assurer les échanges en **mode alternat** sur la longue ligne du ***cordon ombilical***. Et oui, vous vous en doutiez bien qu'à un moment ou à un autre il faudrait le ***brancher ce cordon de deux mètres de longueur***. Pour valider avec fiabilité les programmes qui vont être développés, il importe de se placer dans les conditions "définitives". Ainsi on pourra déterminer jusqu'à quelle vitesse de transmission maximale on peut de façon fiable déclencher les échanges. Plus cette vitesse sera élevée, moins les durées pour échanger les données prendront du temps.
Ça alors, pour dialoguer sur les broches **D0** et **D1** d'Arduino on continue bêtement à utiliser les instructions banales telles que Serial.begin(115200), Serial.print("texte"), Serial.available() et Serial.read(). Nous n'avons pas à écrire nos propres routines. Magnifique !
Et oui, **D0** et **D1** sont aussi nommées **TX** et **RX** car elles sont directement associées aux lignes de donnée de la prise USB par une électronique installée sur les cartes Arduino. Mise à part la **nécessité d'ouvrir les lignes de dialogue sur D0 et D1 pour téléverser un programme** et les rétablir pour le tester, il n'y a rien d'autre à faire. C'est proprement génial, car le programme complet peut servir directement pour effectuer des essais ... et comme on va le voir ça fonctionne merveilleusement bien. ***Zavez vu ?*** Dans le démonstrateur les octets cavalent à 115200 bauds, autant dire que l'on ne perdra pas plus de temps sur le cordon ombilical de deux mètres de longueur qu'en dialoguant avec le moniteur série du P.C. Le concept Arduino est carrément fabuleux.

➢ Le syndrome de la carpe.

Incident agassif au possible, il peut inexorablement arriver que sur une fausse manœuvre un couple de programmes parfaitement au point devienne complètement muet. Inutile de vous escrimer sur le bouton de RESET, quand tout est bloqué, s'acharner sur ce pauvre poussoir ne servira strictement à rien. Cet incident arrivera avec certitude chaque fois que vous tenterez de téverser un programme dans l'une des deux cartes et que l'inverseur est resté sur **Dialogue TX/RX**. C'est imparable, la seule façon de débloquer le système consiste à suivre la procédure donnée en bas de la Fiche n°31 et nommée pour la circonstance Reprise d'une synchronisation avec l'IDE.
Ce type d'aventure constitue le prix à payer pour bénéficier de l'immence facilité apportée par l'environnement de programmation pour faire usage d'une ligne série. Il faudra donc s'armer de rigueur, et chaque fois que l'on aura terminé des essais **BASCULER l'inverseur vers IDE sur le P.C.** pour reprendre l'édition des logiciels. Globalement cette précaution va rapidement devenir une habitude. Reste qu'il suffit que l'une des deux cartes peut facilement faire perdre la synchronisation sur la ligne de dialogue si son programme interne comporte une quelconque erreur.

➢ Ça va dropper !

Maintenant que nous avons vérifié que la cadence de tir sur **TX** et **RX** sera considérable, il nous reste à maitriser des échanges automatiques à des rythmes très élevés et vérifier si on peut le faire raisonnablement sans perte de synchronisation et de façon fiable. Dans ce but on téléverse P30_Programme_COMPLET_T5.ino sur la carte esclave **S** de la sonde. *(Et oui, je vous l'avais dit que l'on pouvait user des logiciels sans modification ... à condition toutefois que sur la carte du pupitre **P** on téléverse un programme adapté.)* Puis, sur la carte du pupitre **P** on télécharge le démonstrateur P20_Maitre_pour_dialogue_rapide.ino qui va se charger de causer de manière entièrement automatique avec JEKERT. Quand les deux machines sont munies des logiciels en question, vous établissez la liaison ombilicale, un petit RESET sur **P** et vogue la galère.
Chaque alternat inverse l'état de la LED rouge. Si conformément aux remarques données dans le démonstrateur P20_Maitre_pour_dialogue_rapide.ino vous avez branché un bruiteur sur la sortie **D6**, chaque fois que les machines auront réalisé 1000 alternats, c'est à dire 500 allumages des phares et du LASER et 500 extinctions de ces deux périphériques, un BIP sonore sera généré.

- À donf le bla bla bla, ça cause un max !

Examinons le contenu de la boucle de base du programme Maître :

```
  void loop() {
① Serial.print("a*"); Traiter_ACR(); delay(3);
② Serial.print("l*"); Traiter_ACR(); delay(3);
③ Nb_alternats++;
④ if (Nb_alternats == 1000) {BIP(); Nb_alternats = 0;}}
```

Fig.229

Sa structure est d'une banalité évidente. En ① le programme transmet sur la ligne série du cordon ombilical une Consigne pour alterner l'activation des phares. Puis, immédiatement après avoir transmis la Consigne, le logiciel se met à l'Écoute pour réceptionner l'ACR.
Étonnant, on raconte que l'on va chercher à dialoguer le plus rapidement possible, et bêtement on paralyse l'ATmega328 pendant 3mS ! Puis, après ce gaspillage de temps la ligne ② transmet une Consigne pour alterner l'activation du LASER. Rapidement il repasse à l'Écoute pour réceptionner l'ACR. Et nouvelle consommation de temps de 3mS à se tourner les Octets !
Pas grand chose à dire pour la suite. En ③ on incrémente le compteur de bavardages, puis en ④ on génère le BIP sonore tous les 1000 échanges du type "fais ça" / "C'est fait".
Pourquoi ces deux instructions delay(3) ?

➢ La synchronisation.

Imaginez-vous à l'école. *(Berkkk, on va travailler !)* Le Maître Parle sans interruption. Vous Écoutez avec attention, cependant le temps pour tout écrire n'est pas suffisant et la main prend du retard sur les oreilles. Arrivera le moment où quelques mots vont se perdre. Du coup, la phrase qui suit n'est pas complète et vous ne comprenez plus tout à fait ce que raconte le Maître. Vous avez perdu la synchronisation. C'est exactement ce qui peut se produire lorsque deux machines dialoguent. Passez les deux temporisation delay(3) en remarque avec // ou effacez ces instructions dans P20_Maitre_pour_dialogue_rapide.ino et relancez le bavardage. La sonde va immédiatement se fâcher tout rouge et générer à la pelle des BIPs d'alertes sonores. Sa mémoire tampon réservée pour la ligne série est saturée car le Maître cause trop rapidement. L'Esclave n'a pas le temps de tout réaliser. Quand il revient à l'Écoute une partie du message a été perdue car le tampon mémoire de la ligne série est gavé. Le message étant incorrect **S** fait alors entendre sa désapprobation.
Analysons le listage de la Fig.229 en supposant que l'on n'a pas codé les deux temporisations. En ligne ②, dès que le démonstrateur à reçu et analysé l'ACR issu de l'Esclave, en quelques microsecondes il traite les ligne ③ et ④ puis immédiatement se remet à Parler en ①. Pendant cette courte durée que laissent ③ et ④ l'Esclave doit traiter tout ce qui se trouve dans sa boucle de base entre Accuser réception et Attendre_une_chaine c'est à dire l'intégralité des instructions de ③ à ⑩ suivies de ① et ②. Pour beaucoup la ligne se résume à un test négatif et passage à la suite. Par contre, en ⑥ l'instruction s'avère plus gourmande en µS car elle procède à une mesure sur l'entrée analogique du microcontrôleur avec interpolation des valeurs, et un test négatif active un else.
Toutes ces instructions exigent plus de temps processeur sur l'Esclave que ③ et ④ sur le Maître qui de ce fait parle à une cadence trop rapide. Si on ne ralentit pas son débit "verbal"

① Traiter uneRequette_MPU6050.
② Traiter Listage_Gyro_en_continu.
//------------------- Traite une consigne arrivée sur USB ----------
Attendre_une_chaine. Enregistrer un message sur RX.
Analyser le message. Traiter la Consigne. Accuser réception.
//---
③ Traiter ACR_configuration_sur_USB.
④ Faire clignoter la LED de la boucle de base active.
⑤ Traiter les LED témoins d'état de la Sonde.
⑥ Traiter les fonctions analogiques.
⑦ Traiter le LASER.
⑧ Traiter les mouvements de Torsion.
⑨ Piloter_un_moteur_en_manuel.
⑩ Gérer la stabilisation Gyroscopique.

Fig.230

il y aura perte d'une partie de certaines Consignes et génération d'erreurs. Pour éviter une telle perte de synchronisation qui pourrait aller jusqu'au blocage complet des dialogues, il importe de temporiser le Maître donc générer un petit délai AVANT de Parler.
Dans le démonstrateur on constate que des loupés se produisent si on limite à delay(2). Il sera impératif de se montrer prudent, car dans cette manipulation aucune des LEDs d'état n'est allumée. Tous les tests sont négatifs. Si certaines fonctions comme la Torsion ou la Stabilisation Gyroscopiques imposent du traitement en temps réel, le délai d'attente devra forcément être augmenté. Se pose alors la question de savoir quelle valeur de "temps gaspillé" adopter.
Nouvelle approche conversationnelle : Pour contourner radicalement ce problème, une approche différente consiste à réorganiser la structure de la boucle de base du programme Esclave. Cette boucle sera conçue avec une chronologie légèrement différente :

Attendre_une_chaine. Enregistrer un message sur RX.
Analyser le message. Traiter la Consigne.
Traiter toutes les fonctions de la boucle de base.
Accuser réception.

Avec cette nouvelle structure, quel que soit le temps passé à traiter les fonctions de la boucle de base, après avoir accusé réception l'Esclave repasse immédiatement à l'Écoute.

STRATÉGIE DE DÉVELOPPEMENT

Repartir de zéro pourrait sembler pessimiste, voir frustrant. C'est pourtant la façon la plus rationnelle de reconstruire entièrement l'édifice logiciel. La structure de la boucle du programme fonctionnant sur JEKERT sera différente. Nous allons réduire au minimum les informations retournées sur **TX** puisque la mise en forme sera traitée sur le programme Maître. Par ailleurs, il n'est plus utile d'envoyer des consignes sous forme de texte faciles à mémoriser puisque l'interface Homme/Machine sera sur la Raquette de commande. Aussi, à bien y regarder nous allons concrètement agencer ***deux programmes entièrement nouveaux***. Chercher à récupérer la dernière version P30_Programme_COMPLET_T5.ino et l'adapter serait du "bricolage" avec un risque d'effet "boule de neige" trop important, avec le danger de conserver des commandes qui ne sont plus pragmatiques, de conserver des variables devenues inutiles etc. Il faut entièrement revoir l'analyseur syntaxique, on va remplacer le potentiomètre par un codeur Rotatif etc.
La meilleure approche à mon sens consiste à ***construire progressivement et simultanément les deux logiciels associés***. Pour le programme Esclave on récupèrera les séquences au point, mais une à une avec chaque fois les essais de validation qui s'imposent sur les deux machines, ainsi que la mise en place des affichages sur l'écran de contrôle du pupitre de commande. On se doute que la mémoire non volatile sur **S** contiendra moins de textes, son implantation sera certainement différente. En préambule à tous ces chamboulements, il semble judicieux de concevoir globalement ce que pourra être le pupitre de commande car sa gestion par **P** en dépend directement.
Bref, on a pas mal de pain sur la planche ...

Méthode pour développer deux programmes couplés.

Remettre entièrement en cause un gros logiciel qui globalement a fait ses preuves ne consiste pas de repartir entièrement depuis zéro. Si l'on désire bénéficier un maximum des fruits précédents, il importe de procéder avec rigueur, on évitera ainsi des pertes de temps considérable et surtout on progressera résolument. On se doute que pour "diviser pour régier" on va produire un grand nombre de démonstrateurs. Aussi, pour ne pas s'égarer dans fichier, tout ce qui va suivre sera rangé à part dans le dossier <Sonde et Raquette>. Tout ce qui précède concernant la faisabilité d'un dialogue Machine/Machine est mis à part dans <Dialogue_entre_Machines>. Voici globalement comment on va procéder pour récupérer le code déjà au point et pour différencier les divers démonstrateurs qui vont se succéder :

1) Copier P30-T5 dans A_VIDER.ino qui ***servira à puiser les séquences et les initialisations déjà bien au point***. *(Certaines seront modifiées pour améliorations ou surtout simplifications.)* Chaque ligne de cote prise en compte est enlevée. Ainsi à la fin on verra ce qui reste et qui est devenu inutile et surtout on discernera éventuellement des détails oubliés qui ont leur importance.
2) Bien que sur la Sonde le BUZZER n'est plus utile puisque les avertissements seront "sur Terre", BIP() est conservée puisque le matériel est disponible. Ainsi, on peut s'en servir pour traceur auditivement des points de passage durant le développement du démonstrateur et ainsi vérifier si le programme invoque certaines séquences ou non. *(C'est un outil de programmation.)*
3) On va développer des démonstrateurs progressifs :
   P21**v**_Démonstrateur_Raquette.ino pour les logiciels "fragmentaires. (1)
   P22**v**_Démonstrateur_Sonde.ino pour les logiciels "fragmentaires. (1)
4) P40_PGM_RAQUETTE_MAITRE.ino recevra au fur et à mesure du développement des démonstrateurs les compléments mis au point. C'est donc le programme complet définitif.
5) P50_PGM_ESCLAVE_SONDE.ino recevra au fur et à mesure du développement des démonstrateurs les compléments mis au point. C'est donc le programme complet définitif.

(1) : "**v**" sera une lettre indiquant la version du programme. Les lettres **A**, **B**, **C** etc correspondent à l'ordre chronologique de développement. Si deux démonstrateurs sont menés de front ils ont la même lettre de version. Dans tous les cas pour chaque démonstrateur est précisé en remarque le programme associé avec lequel il doit être testé.

NOTE : Tous les programmes P21 sont relatifs au pupitre de commande, tous les démonstrateurs P21 concernent le logiciel qui se charge sur JEKERT.

Évalué globalement, le pupitre de commande sera équipé d'un clavier **CLv** à seize touches. La répartition géométrique sur le coffret n'est pas définitivement définie, le besoin par contre montre que seize boutons poussoir ne seront pas de trop. Ce nombre n'est pas le fruit du hasard. Il correspond à une optimisation fonctionnelle, ainsi qu'à une optimisation dans le cas d'un traitement multiplexé sur un petit clavier matricé du commerce servant aux essais. Pour piloter le LASER, la TORSION, le niveau des éclairages nous utilisions sur le prototype un potentiomètre. Aussi, personnellement je n'aime pas trop ces dispositifs qui dans le temps se mettent à "cracher". Par ailleurs, balader des signaux analogiques de tension faible sur de longues lignes filaires les rend très sensibles aux parasites électromagnétiques environnants. Aussi ce dernier sera remplacé par un Codeur Numérique **Cn**. Enfin nous visualiserons les données sur un écran graphique **Ecr**. La Fig.231 résume assez bien les divers périphériques qui devront être gérés par le logiciel P21**v**. Nous allons en toute logique devoir développer trois démonstrateurs **A**, **B** et **C**. On va réserver trois chapitres dédiés à ces divers démonstrateurs.

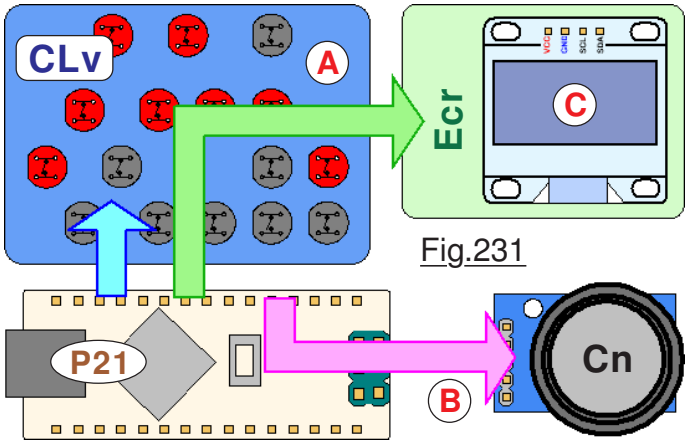


Fig.231

48) 27/12/2017 : Gestion d'un clavier multiplexé *(MJD 58114)*

Laissant un peu la bride sur le cou à l'équipe des programmeurs, on se rend ce jour en salle électronique S12 dans laquelle nous avons négligé un peu les personnels. Il est temps d'aller les voir pour évaluer l'avancement de leurs travaux. Quand on arrive, il sont en train de commencer à assembler une magnifique console dont plusieurs modèles seront installés un peu partout dans les stations de poursuite autour du globe, ainsi qu'à Toulouse au centre principal de contrôle. Ce jour Julien 58114 ils sont en train de déballer de magnifiques claviers à intégrer sur les pupitres de maîtrise.

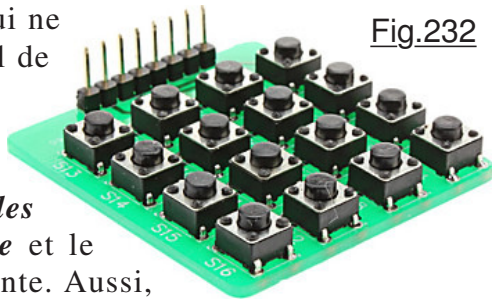


Fig.232

Lorsqu'on s'amuse à programmer de petites applications qui ne comportent que deux ou trois touches, il est assez naturel de ne pas se compliquer la vie et des les prendre en compte comme montré sur la Fig.228 en réalisant un diviseur potentiométrique dont la tension est mesurée sur une entrée analogique. Plus on place de touches dans le diviseur de tension, ***plus l'écart entre les niveaux représentatifs*** des contacts électriques ***devient faible*** et le risque qu'un parasite fausse la mesure en cours du CAN augmente. Aussi, jusqu'à cinq boutons poussoir par entrée on peut considérer que c'est raisonnable, mais guère plus.

➢ *Clavier organisé en matrice.*

Consultez la Fiche n°32 relative à un petit clavier du commerce dont la Fig.232 montre l'allure. Il est typiquement organisé en matrice. Pour simplifier, quand on ne fait pas référence aux mathématiques, ***une matrice est un damier organisé en lignes et en colonnes***. Un jeu d'échec est une matrice 8 x 8. La bataille navale, un écran graphique comme celui que nous allons utiliser, un caractère texte affiché sur un écran LCD textuel, tous ces éléments sont des matrices. La Fig.233 montre le schéma électrique du petit clavier commercial qui est vendu sans les composants de mise en œuvre, laissant à l'utilisateur le soin de l'agencer comme il l'entend. Souvent on se contente d'affecter quatre sorties pour les colonnes et quatre entrées pour les lignes, c'est la façon banale et "standard" d'exploiter un tel dispositif. La technique qui consiste à utiliser cinq résistances de plus et quatre diodes augmente un peu la complexité du circuit électronique. Elle fait économiser trois E/S sur le microcontrôleur et simplifie un peu l'exploitation logicielle. Toutefois elle n'est utilisable que si le processeur utilisé dispose de convertisseurs AN, grand avantage de l'ATmega328.Naturellement il est possible de concevoir des matrices 3 x 3 pour 9 touches, 5 x 4 pour 20 touches etc. Le petit clavier de la Fig.232 est bien commode pour conduire des expérimentations car il regroupe sous un faible volume un grand nombre de touches. Matricé 4 x 4 il comporte 16 boutons poussoir qui sont géométriquement également agencée en quatre lignes de quatre colonnes. Pour développer, c'est utilisable, mais pour la qualité opérationnelle, les touches seront disposées par des regroupements fonctionnels. On concevra un clavier "maison" et ce n'est qu'électriquement que les boutons poussoir seront comme sur la Fig.233 organisés en matrice.

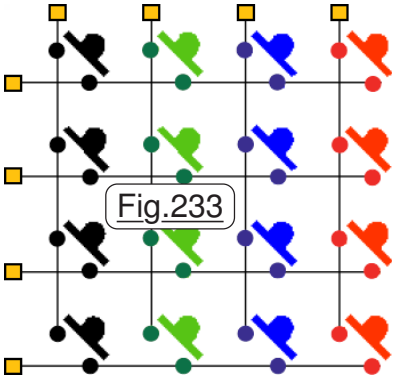


Fig.233

➢ *Circuit imprimé expérimental.*

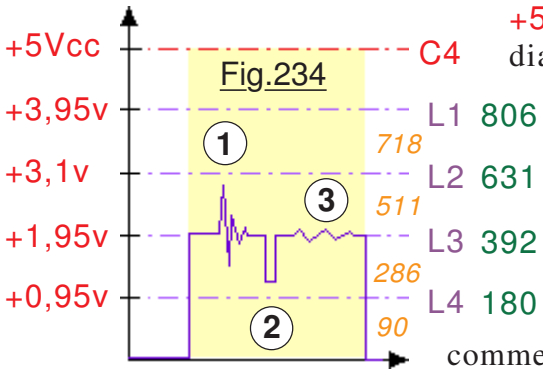
Faisant partie intégrale du laboratoire "Développement Arduino", le petit clavier dont il est question dans le chapitre qui précède est associé à un tout petit circuit imprimé qui se place en gigogne directement sur le connecteur HE14 du module commercial. Sois dit en passant, avoir soudé un connecteur vertical sur leur produit est à mon avis une très mauvaise idée. Un connecteur HE14 coudé se branchant latéralement serait bien plus commode. De ce fait ce module devient pratiquement inutilisable dans un quelconque coffret. Dommage qu'il était livré tout soudé, dans le cas contraire un connecteur bien plus adapté aurait été installé. Bien qu'un circuit imprimé sera spécialement dédié à la Raquette de commande, à titre d'information je vous livre sur la Fiche n°33 le petit circuit de complément expérimental. Pour mieux comprendre comment cette interface électronique est réalisée et surtout enfichée sur le clavier, les Image 55.JPG et Image 56.JPG sont disponibles dans le dossier <Galerie d'Images> qui accompagne ce **TOME 5** du didacticiel.

➢ Principe du multiplexage : Le balayage.

Prendre la Fiche n°32 car nous allons raisonner sur la Fig.2 tant électroniquement qu'informatiquement. Si les diodes **D** n'étaient pas insérées dans le circuit, les résistances de ligne **R3** seraient en parallèle, et n'en formeraient physiquement qu'une seule. Le circuit ne fonctionnerait pas. Quand ces diodes **D** ne sont pas conductrice, elles se comportent comme des isolants électriques et l'entrée **A6** se trouve "en l'air". Soumis à un haute impédance, le fil de liaison forme antenne radio et capte des parasites hertziens. La résistance de 47kΩ a donc pour rôle de maintenir **A6** à zéro volt tant qu'aucun B.P. n'est enfoncé. *(Sa valeur est choisie assez importante pour ne pas trop changer celle des résistances placées sur les lignes, car elle se trouve en parallèle. Il faut toutefois ne pas prendre beaucoup plus que 47kΩ car plus l'impédance est élevée, plus l'entrée sera sensible aux parasites radios.)* Tout multiplexage est basé sur l'idée de balayage. Pour tester le clavier, le principe consiste à "regarder" par exemple si sur la colonne **C1** une touche est enfoncée. Si ce n'est pas le cas, alors on teste **C2**, **C3** et **C4**. Si aucune colonne de touches n'a été sollicitée, le programme en déduit que le clavier est au repos. La séquence est terminée et le logiciel poursuit infiniment sa boucle de base pour traiter d'autres séquences. *(On peut effectuer cette exploration des colonnes dans un ordre quelconque, de même que l'on peut balayer les lignes et mesurer les tensions sur les colonnes, toutes les combinaisons sont possibles.)* Imaginons par exemple que cette exploration du clavier arrive sur **C4**. Dans ce but, comme montré sur la Fig.2 **D10** passe au niveau "1". Supposons qu'au moment de la mesure nous ayons appuyé sur la touche **S12** par exemple. Le +5Vcc disponible sur **D10** est divisé par deux par la résistance de 1kΩ **R2** et la résistance de 1kΩ **R3**. La diode **D** prélève ses 0,5V environ et **R1** fait diminuer un peu la tension qui sur **A6** avoisine alors +1,95Vcc. La "sortie" du convertisseur CAN fournit une valeur numérique de 392. Le programme sait alors que l'une des touches de la ligne **L3** a été sollicitée. Comme c'est lui qui active **D10**, il sait aussi que c'est sur **C4** que se trouve le B.P. cliqué. Il en déduit que c'est **L3**/**C4**, donc la touche **S12**.

➢ L'immunité aux parasites hertziens.

L'anti parasitage doit dans la pratique compléter la théorie abordée dans le chapitre précédent. Examinons en détail la conception de l'interface du clavier, sachant que deux sortes de pollutions viennent se superposer. La première a pour origine l'environnement électromagnétique. Les résistances ont été astucieusement choisies pour n'employer que des composants ordinaires de valeurs standards, qui dans le montage vont diviser la tension +5Vcc en cinq zones "identiques". Chaque palier est séparé de ≈1Vcc. Supposons que l'on teste la ligne **L3**. La sortie **D10** passe à +5Vcc début de la zone jaune sur la Fig.234 qui propose un diagramme temporel de la valeur de la tension présente sur l'entrée de numérisation **A6**. *(Ligne ④ du programme listé sur la Fig.236.)* Le convertisseur analogique numérique pour effectuer son travail exige la durée coloriée en jaune. Si tout était parfait, la tension mesurée serait de +1,95v bien constante. Dans la réalité, durant cette période il peut se produire une grande variété de perturbations. Par exemple en permanence se superpose "du bruit en herbe" comme symbolisé en **3**. En **1** on a coupé un moteur électrique voisin provoquant un parasite de type oscillation amortie, sans compter qu'en **2** un éclair orageux à induit sur le réseau électrique une surtension qui atténuée se retrouve en sortie des alimentations régulées de nos électroniques. Bref, le +1,95v est en permanence fluctuant. Les valeurs numérisées des seuils théoriques sont indiquées en vert sur la Fig.234 correspondant à chaque ligne du clavier. Si vous regardez attentivement la ligne ® du programme, vous constaterez que les tests sont effectués au "milieu" de chaque plage. Par exemple toute numérisation comprise entre 286 et 511 sera considérée dans la batterie de comparaisons comme appartenant à la ligne **L3**. Toute numérisation comprise entre 511 et 718 sera considérée dans la suite de comparaisons comme appartenant à la ligne **L2**. Le logiciel introduit par cette technique une immunité aux parasites de ±0,5 volt ce qui dans la pratique s'avère suffisant et fiable. Notez au passage que si l'on augmente le nombre de touches par colonne les écarts entre seuils diminuent, donc l'immunité logicielle. C'est la raison pour laquelle il reste recommandé de ne pas dépasser cinq B.P. par colonne.



➢ L'aspect logiciel en C++.

Outre les parasites issus de l'environnement électromagnétique, on se heurte à un autre problème d'origine mécanique. Quand on entend le clic caractéristique d'un bouton poussoir à bascule, la théorie suppose que le contact électrique est établi de façon immédiate et stable comme représenté sur la Fig.235 en sur **1**. Laissez tomber une balle de votre hauteur sur un sol dur. Étant donné quelle est élastique, elle rebondit et remonte presque aussi haut. Un peu moins car la déformation due au choc consomme un peu d'énergie. Puis elle retombe et rebondit encore, toujours un peu moins haut. Chaque rebond va prendre un peu moins de temps car à chaque fois la vitesse ascensionnelle la fait monter moins haut. Cette balle va ricocher de plus en plus rapidement jusqu'à avoir consommé toute l'énergie "de hauteur initiale". Si au lieu d'être en caoutchouc elle était en acier, le phénomène serait identique, sauf que le "rendement" étant meilleur la perte à chaque choc serait moindre et le nombre de rebonds plus important. Pour un contact électrique il se produit un phénomène tout à fait analogue. Nous entendons un unique "clic". Dans la réalité, comme en **2** sur la Fig.235 se produisent une kyrielle de rebonds mécaniques. Chaque saut coupe le contact, et comme le microcontrôleur peut boucler rapidement une séquence, il va croire que l'on a appuyé plusieurs fois sur la touche. Sur un bon interrupteur on compte environ 6 à 8 commutations. Sur un B.P. ordinaire on peut en totaliser entre 30 et 50 !

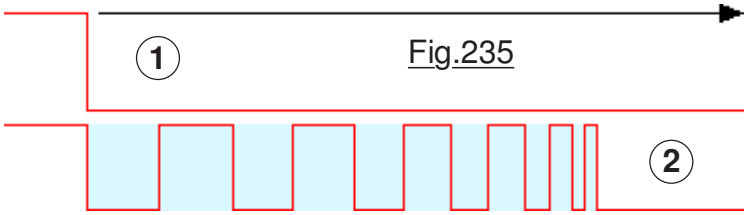


Fig.235

Pour parer ce genre de problème on peut placer un condensateur de 10nF à 50nF aux bornes du contact. Un composant, c'est concevable, mais pour 16 le coût et la place exigée sur le circuit imprimé sont exagérés. Aussi, on peut alors procéder à de l'antirebonds par logiciel. C'est le premier démonstrateur P21**A**_Démonstrateur_Raquette.ino qui établit une relation conversationnelle entre les deux cartes Arduino. Seize touches : Seize fonctions comprises entre "24*" et "39*" pour tester le clavier et commencer à mettre en place notre stratégie logicielle. La Fiche n°35 donne la liste des codes retenus pour envoyer les Consignes à la Sonde. Examinons le contenu du démonstrateur :

```
   void Tester_le_clavier() {
①   BP = 0; // Indique une non utilisation du clavier.
②  Teste_Colonne(C1); Teste_Colonne(C2); Teste_Colonne(C3); Teste_Colonne(C4);}

③ void Teste_Colonne(byte Colonne) {
④   digitalWrite(Colonne, HIGH); // Passe la colonne au +5Vcc.
⑤   CNA = analogRead(Mesure_U_clavier);
⑥   if (CNA > 90) // Codage du BP sollicité seulement si une touche est active.
⑦     {BP = Colonne + 6;
⑧      if (CNA>286) BP = BP - 4; if (CNA>511) BP = BP - 4; if (CNA>718) BP = BP - 4;
⑨      while (analogRead(Mesure_U_clavier) > 90) delay (5); delay (10);}
⑩   digitalWrite(Colonne, LOW);} // Repasse la colonne à zéro volts.
```

Fig.236

En ① le code BP prend la valeur zéro qu'il conservera en sortie de Tester_le_clavier() si aucune touche n'est activée lors des quatre mesures effectuées en ligne ⑤. En ligne ② on teste successivement les quatre colonnes même si l'une d'elle répond positivement. *(Un swich sase a été testé et conduit à un encombrement identique. Comme la cascade des if est plus parlante elle a été conservée.)* Pour tester une colonne, on fait monter sa sortie Colonne à +5Vcc en ④. En ⑤ on mesure la valeur de la tension, parasites inclus. En ⑥ on ne modifiera la valeur de **BP** que si une touche dans la colonne testée est enfoncée. La constante C1 désigne la sortie testée **D7** et vaut 7. Donc 7 + 6 en ligne ⑦ donne la valeur 13 à **BP**. En fonction de la tension mesurée, on aura entre zéro fois et trois fois la soustraction de 4 unités. Pour la colonne de **D7** on obtient alors 13, 9, 5 et 1 qui sont les valeurs correspondant aux touches **S13**, **S9**, **S5** et **S1**. Ce n'est pas un hasard si les quatre sorties utilisées pour multiplexer ont été choisies consécutives. Ainsi le codage est particulièrement économe en code objet. En ⑨ le while va boucler tant que l'on détecte l'état "activé". On temporise 5mS pour laisser passer le rebond et on recommence. Quand il n'y a plus de rebond, on réalise un délai de sécurité de 10mS. Enfin en ⑩ la vérification sur la colonne est achevée, on fait

repasser la sortie à zéro. En toute logique ⑩ devrait être placée juste après la ligne correspondant à la mesure en ⑩. Curieusement si on corrige ce "manque d'élégance du programme, le code objet augmente de 10 octets ! Alors le programme source a été conservé en l'état.

➢ Codage des Consignes envoyées à la sonde.

Partie intégrante du changement de stratégie opéré pour ce **TOME 5**, nous savons qu'il n'est plus nécessaire d'avoir des consignes faciles à mémoriser par l'opérateur. Par exemple sur le démonstrateur la touche **S7** "fige les moteurs", la touche **S8** les rétablit. Sur le clavier définitif, par exemple on réservera une seule touche notée Moteurs OFF. Un premier clic sur cette dernière allumera la LED rouge, un deuxième clic l'éteindra. Sur la Fiche n°35 on constate que pour suspendre les moteurs on a choisi le code "31*" et pour rétablir la motorisation le code "32*". C'est le programme complet P40_PGM_RAQUETTE_MAITRE.ino qui sera chargé d'envoyer des codes différents en fonction de la chronologie d'utilisation de cette touche de type bascule OUI/NON.

- Pourquoi avoir choisi deux codes différents pour une action de même nature ?

Tout simplement pour optimiser le code de P50_PGM_ESCLAVE_SONDE.ino qui n'aura plus à se préoccuper du booléen dans la boucle de base. En effet, dans l'ancien programme, chaque rotation de la boucle de base testait le booléen Moteurs_ON. Donc à chaque itération soit le if invoquait la

```
if (Moteurs_ON) digitalWrite(LED_Moteurs_OFF,LOW);
else digitalWrite(LED_Moteurs_OFF,HIGH);
```

procédure digitalWrite pour allumer, soit le else digitalWrite pour éteindre. Le microcontrôleur passe son temps à allumer ou éteindre des centaines de fois une LED qui est déjà dans l'état logique désiré. Aussi il est bien plus logique de ne le faire qu'une seule fois. Il faut plus de code puisque l'on a deux cas à traiter dans le swich sase. Pour compenser on économise le if et le else. Ce doublement des codes de Consigne sera systématiquement utilisé dans le programme de la Raquette.

Puisque c'est une instruction de type swich sase qui sera chargée d'aiguiller les traitement en fonction du code de la Consigne, on peut adopter ce que l'on désire pour ce dernier. Aussi, pour pouvoir réutiliser la Fiche n°12 durant le développement, les codes "1*" à "23*" correspondent directement aux fonctions des programmes "p01*" à "p23*". Globalement, les autres codes sont attribués au fur et à mesure que le logiciel prend de la consistance.

Les Consignes les plus longues, tout au moins à ce stade des études, font donc trois caractères sentinelle '*' comprise puisqu'il n'y a plus besoin de préciser 'p'. Pour la répétition d'un mouvement élémentaire, il sera toujours possible de conserver une lettre de sélection avec une Consigne du genre "p3d*" ou "p4c*" par exemple. Sachant qu'une chaîne de caractères mémorisée par le compilateur est terminée par un délimiteur, on réserve quatre Octets avec la constante LGR_chaine.

➢ Changement de stratégie pour optimisation.

Anciennement la commande "b*" était réservée pour figer les lumières quand le potentiomètre servait à pointer le LASER, à déplacer en TORSION ou à piloter un moteur en mode manuel. Pour son compte, la commande "n*" neutralisait le potentiomètre quand on changeait de moteur cible en mode manuel. Il sera plus logique d'affecter un indicateur pour préciser sur qui on agit quand le codeur rotatif est en cours d'utilisation. Ainsi les variations du potentiomètre virtuel pourront être dosées en fonction de la cible. LASER et Phares pourront être modifiés séparément sans interaction réciproque. Enfin, le pilotage manuel des moteur pourra être amélioré : Chaque changement de moteur cible commencera par forcer le potentiomètre virtuel à la "consigne du moteur" évitant ainsi les mouvements brusques. Les commandes "b*" et "n*" devenues inutiles libèrent leurs LEDs respectives. En revanche, il devient utile de savoir quand le potentiomètre virtuel est arrivé en butée logicielle puisque le codeur incrémental n'a pas de limite angulaire.

Fig.237

La LED jaune de l'ancien "n*" signalera la saturation dans le sens Positif. La LED blanche de l'ancien "b*" signalera la saturation dans le sens Négatif. Émuler un potentiomètre virtuel libère l'entrée **A0** qui deviendra une sortie pilotant une LED rouge "BUZZER" montrée sur le Fig.237 soudée sur un petit adaptateur HE14 à trois broches se branchant à la place de l'ancien potentiomètre. C'est un témoin visuel "système" servant au développement "silencieux" du logiciel.

49) 28/12/2017 : Gestion d'un codeur rotatif indexé (MJD 58115)

Commençant notre visite de courtoisie par la salle électronique S12, les techniciens ont bien travaillé et déjà plusieurs consoles d'exploitation sont presque terminées, tout au moins pour les boutons et les claviers. Ils sont en train de combler certains trous par de superbes gros boutons rotatifs dont l'utilisation n'est pas encore précisée, car les panneaux esthétiques comportant les inscriptions sont toujours bien emballés dans leurs caisses de livraison. Chaque composant est préalablement vérifié électriquement pour être validé avant sont installation sur la console. Ils avancent bien, ce n'est pas ce service qui va ralentir le projet et bousculer le planning.

➢ Codeur rotatif incrémental.

Exclure le potentiomètre qui servait à pointer le LASER, à augmenter ou diminuer la puissance lumineuse ou diriger la TORSION, **a été justifié pour des raisons techniques**. Composant peu fiable par nature, le remplacer par des technologies plus robustes s'impose naturellement. Ce n'est toutefois pas la seule raison qui incite fortement à le remplacer pas un codeur rotatif incrémental. C'est surtout un critère de qualité opérationnelle d'utilisation de notre pupitre. En effet, le nombre de Consignes dépassait les cinquante dans l'ancienne version. *(23 programmes et environ 31 consignes sous forme d'un seul caractère.)* Il n'est pas question d'envisager un clavier qui comme celui d'un P.C. permet d'envoyer tous ces "textos". Nous allons forcément procéder par des sélections sous formes de "MENUs Déroulants". Hors la rotation d'un potentiomètre fait au maximum 270° et surtout on domine mal sa "progressivité". Alors qu'***avec un codeur incrémental on sait immédiatement que chaque clic d'indexation engendre un pas***. Clic clic clic : On s'est déplacé de trois items dans la liste, en avant ou en arrière en fonction du sens de rotation. Coté convivialité on ne peut pas trouver mieux. Du coup, ce qui est très pratique pour déplacer un index sur un menu écran l'est aussi pour diriger un LASER ou doser un éclairage. Le pupitre de JEKERT sera donc muni d'un tel dispositif si agréable à employer, surtout sur un petit pupitre où il y a peu de place.

➢ Mise en œuvre d'un codeur rotatif incrémental.

Qualifié d'incrémental car bien qu'il puisse tourner sans butée autant que désiré, il ne peut occuper de manière stable que des positions bien précises. Il y a quarante indexations par tour pour le modèle utilisé. Désolé chères lectrices et chers lecteurs, je me suis un tantinet fourvoyé. Le didacticiel est rédigé conjointement au développement du projet. Plusieurs semaines y sont englouties. Hors j'avais complètement oublié que la Fiche n°19 décrivait le composant utilisé. Je viens de m'en apercevoir, effectuant un contrôle d'informations sur les documents déjà publiés. Hors je me suis donné le mal d'en refaire une. Comme cette Fiche n°34 est plus complète et surtout documentée avec plus de rigueur, elle rend caduque la Fiche n°19 que vous pouvez ranger précieusement aux oubliettes. Refermons cette parenthèse.

Généralement les codeurs Angulaires fournissent l'information de position sur plusieurs sorties binaires. Pour résoudre le problème des fausses numérisations par décalages temporels subtils sur les changements d'états de plusieurs sorties, on utilise des codes binaires intelligents tel que le code Gray par exemple. Grosse différence, ***dans notre cas ce n'est pas une position angulaire que l'on désire prendre en compte, mais un incrément de rotation***. Du coup deux sorties sont suffisantes pour pouvoir effectuer du "comptage" et déterminer le sens de rotation. La technologie de ces composants ainsi que le principe d'utilisation sont précisés sur la Fiche n°34.

➢ Approche logicielle : Oublions les interruptions.

Souvent traité par interruptions, c'est la première technique qui avait été utilisée sur le démonstrateur initial P21**B**_Démonstrateur_Raquette.ino qui introduit la gestion du codeur incrémental et celle du gradateur des énergies lumineuses. Dans cette façon de traiter le bouton à encliquetage, nous sommes obligés d'utiliser les broches **D2** et **D3**. Si pour une quelconque raison on désire changer cette affectation, ce sera possible car les interruptions ont été abandonnées. Cette approche qui semblait séduisante s'est avérée catastrophique. Son avantage théorique réside dans la prise en compte aléatoire d'une action sur le codeur incrémental même si le programme est en train de traiter une séquence quelconque. C'est avantage sur le papier n'en est finalement pas un puisque de toute façon le traitement se fait dans la boucle de base, les routines d'interruptions

ne faisaient que détecter une rotation, de positionner un booléen pour prévenir la boucle de base, et d'indiquer le sens de rotation. En contrepartie, ce non bénéfice s'accompagnait d'une foule de gros problèmes. Déjà le fait de traiter directement les deux entrées **D2** et **D3** dans la boucle de base fait économiser plus de 320 octets ce qui n'est pas rien. Le deuxième problème réside dans la spontanéité des interruptions qui réagissent très rapidement. Les rebonds des contacts du codeur engendraient des fausses détections de transitoires binaires, l'antiparasitage se montrant particulièrement difficile à éradiquer, et ce d'autant plus que les codeurs approvisionnés sont de qualité médiocre.

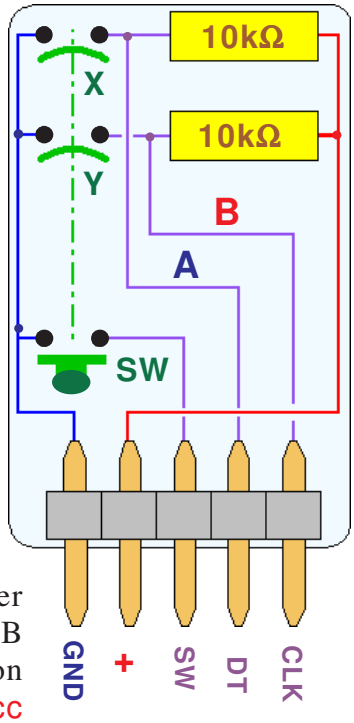


Fig.238A

Pour mémoire la Fig.238A redonne le schéma interne. Les contacts électriques **X** et **Y** étant de type fugitifs par conception de ce composant, on doit trouver sur les sorties **DT** et **CLK** des tensions correspondant à l'état logique "1" puisqu'au repos **A** et **B** ne sont pas en contact avec la masse GND. Faisons une expérience simple. Ajoutons sur les entrées **D2** et **D3** deux diodes électroluminescentes pour visualiser les contacts entre **X** ou **Y** et GND. Le schéma électrique donné sur la Fig.238B est rapidement câblé sur une plaque à essais. La Fiche n°34 en main on déduit qu'au repos les sorties **DT** et **CLK** sont au +5Vcc par les résistances de 10kΩ internes au module. Les deux LED sont éteintes. Quand on réalise un pas en rotation, durant un court instant **X** et **Y** provoquent un niveau "zéro volt" sur **D2** et **D3** ces impulsions étant légèrement décalées. On observe un court éclairement des deux LEDs puis les composants optoélectroniques redeviennent noirs. Comme montré sur la Fig.238C les deux contacts **X** et **Y** peuvent parfois rester en liaison avec GND et ne n'isole pas. Quelquefois ce n'est qu'une sortie qui reste à la masse. Une LED allumée témoigne de ce type d'incident non conforme à la théorie. Entre les rebonds et les contacts erronés, il a été impossible de dominer les interruptions. Soit sur un seul clic elles déclenchaient plusieurs changements d'options, soit le système se bloquait carrément et le codeur rotatif devenait inerte. Les interruptions ont pour toutes ces raisons été abandonnées.

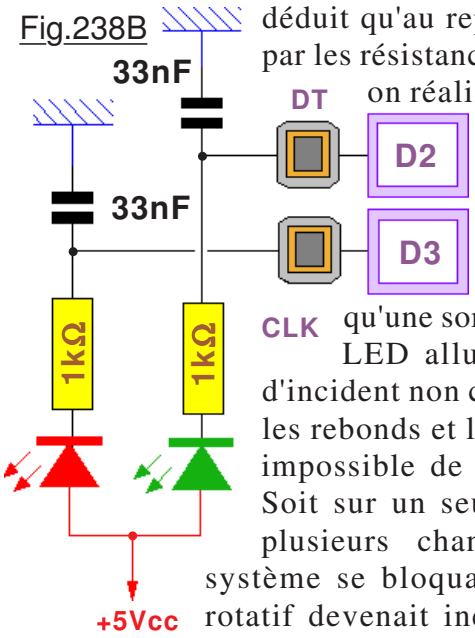


Fig.238B

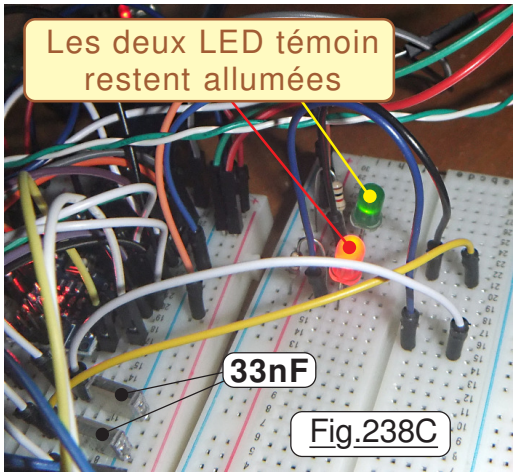


Fig.238C

➢ *Gestion logicielle du codeur KY-040*

C'est la technique décrite en bas de la Fiche n°34 qui sur le listage de la Fig.228D est mise en pratique sur P21**B**_Démonstrateur_Raquette.ino et les versions qui suivent. ***Les transitions montantes sont surveillées dans ce programme*** sur **B** soit l'entrée binaire **D3**. Dans ce but on commence en ① par déterminer l'état de l'entrée Codeur_B. La variable Transition en ⑤ a recopié l'état précédent de Codeur_B. Dans le test de la ligne ② s'il était à LOW et que maintenant il vaut HIGH c'est bien que nous détectons un front montant. Le test étant positif on exécute alors tout ce qui est dans le bloc d'instructions {...}. En lisant l'état actuel du Codeur_A en ③ on détermine de sens de rotation et on positionne le booléen Sens_positif. Dans la procédure *Traite_Codeur_incremental*() en ④ on effectue les instructions concernant le codeur incrémental.

```
①  Etat_B = digitalRead(Codeur_B);
②  if ((Transition == LOW) && (Etat_B == HIGH)) {
③      if (digitalRead(Codeur_A) == LOW) Sens_positif = false;
       else Sens_positif = true;
④      Traite_Codeur_incremental();}
⑤  Transition = Etat_B;}
```
Fig.238D

50) 29/12/2017 : Gestion d'un petit écran graphique (MJD 58116)

Consultation du planning : Il va y avoir pas mal de "taf" en salle électronique S12 et en salle informatique S4. Sur les consoles les techniciens s'affairent, car l'intégralité des boutons et des clavier sont en place. Les belles plaques recouvrent les panneaux mécaniques. Ils ont fière allure ces pupitres d'exploitation avec toutes ces inscriptions. Maintenant les trous pour les écrans graphiques vont être comblés car sur les établis les belles dalles de luminophores sont déballées.

➢ Présentation du tout petit écran graphique OLED Adafruit SSD1306.

Quand à la réception du coli postal on ouvre le paquet et que l'on sort ce minuscule module, on reste dubitatif en considérant ses dimensions vraiment discrètes. L'écran en verre ne mesure que 27 x 15mm. Tant que l'on n'a pas vu ce que donne une si petite image, on ne peut que douter de l'efficience de ce composant. Rapidement on "triture" un programme pour afficher le plus grand cadre possible, pour se rendre compte que ses dimensions ne font plus que 22mm x 16mm. Le doute devient encore plus sérieux. Et pourtant, dès que l'on arrive à afficher du texte, la plus petite police reste parfaitement lisible. "Traçons" ensuite des traits, des rectangles, des cercles, une sinusoïde, tout ce petit monde graphique s'avère parfaitement visible, et immédiatement le doute s'évapore. Un si petit écran remplit totalement sa mission, comme vous pouvez le vérifier sur la Fig.239, le plus petit texte en taille ainsi que les graphes sont très facilement "lisibles". Sur cette photographie, le texte BONJOUR utilise la plus petite police de caractères de 5 x 7 pixels. *(Soit une matrice de texte bien classique.)* La netteté est en outre assez dégradée par le manque de définition de cette photographie, sur laquelle figure en haut à droite un pixel isolé. Il n'y a plus qu'à programmer ...

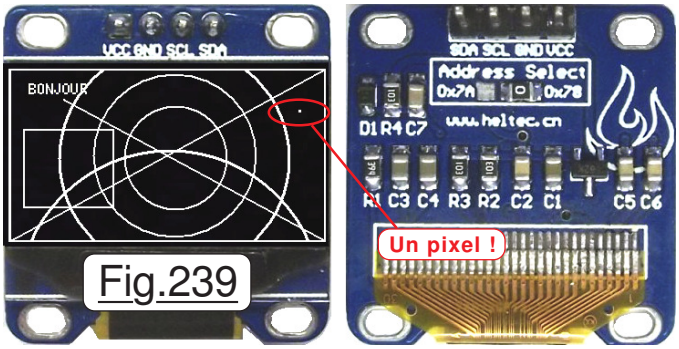


Fig.239

➢ Installer la bibliothèque indispensable.

Programmer un afficheur graphique est particulièrement laborieux, car il faut définir les caractères textuels, tracer des droites, des cercles, limiter ces derniers à convenance, gérer les débordements de définition de l'écran. Heureusement pour nous, quand un composant du commerce s'avère particulièrement séduisant, on trouve parfois une bibliothèque bien pensée créée par un passionné, ou souvent directement par le concepteur du produit commercialisé. Dans notre cas il suffit de télécharger pour l'**IDE** la bibliothèque Adafruit_ssd1306syp.h particulièrement efficiente. Il nous faut installer cette "library", rassurez-vous, ce n'est pas bien compliqué :

1) Téléchargez la bibliothèque dans un répertoire quelconque,
2) Décompacter le fichier ZIP dans ce répertoire.
3) Placer le dossier de la bibliothèque décompressé dans un dossier de votre choix, celui dans lequel vous préserverez toutes les "library" que vous ajouterez à votre environnement **IDE**.
4) Déclarer la nouvelle bibliothèque dans la liste disponible :
   Croquis > Importer bibliothèque.. > Add Library... >
   Indiquer le chemin de votre dossier personnel sur le H.D. *(Précisé en étape 3.)* La bibliothèque est alors ajoutée dans un dossier personnel :
   <C:\Users*VotreOrdi*\Documents\Arduino\libraries>
   ou éventuellement dans
   <C:\Users\user\Documents\Arduino\libraries>.
5) Recharger un programme quelconque. Cliquer sur Croquis puis placer le curseur de la souris sur Importer bibliothèque.. : En **1** on découvre les bibliothèques disponibles par défaut et en **2** les bibliothèques importées dans l'**IDE** par vos soins. Vous devez y trouver Adafruit_ssd1306syp.h comme mis en évidence sur la Fig.240 dans l'encadré rouge.

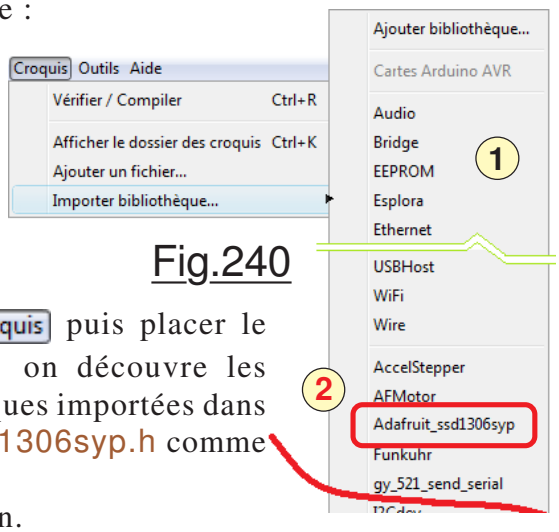


Fig.240

C'est parfait, nous pouvons enfin passer à la programmation.

➢ Faisons connaissance avec l'afficheur OLED SSD1306.

Plusieurs bibliothèques personnalisées sont disponibles sur la toile. Il faut bien faire un choix. Celle utilisée dans les programmes proposés dans ce didacticiel m'a semblé la plus "populaire". Il s'agit de la "library de référence" Adafruit_ssd1306syp.h disponible sur de nombreux sites. Par exemple on peut aller sur : http://www.codeforge.com/read/242813/Adafruit_ssd1306syp.h__html Tous les programmes qui vont suivre vont utiliser ce module logiciel. Pour les comprendre il faudra consulter en permanence la Fiche n°36 qui détaille chaque instruction de cette bibliothèque.
Disposant de ses fonctionnalités, programmer le SSD1306 devient un jeu d'enfant ... Encore que ! En effet, pour tracer tout ce que l'on désire, le code est immédiat pour peu que l'on s'imprime une grille de 64 x 128 carreaux pour repérer les coordonnées. Mais la petite puce fait de la résistance. Le petit écran ne se laisse pas faire et nous oppose dès le départ deux difficultés.

Première embuche : L'approvisionnement peut nous réserver une petite surprise. Ayant mis en service le tout premier de ces composants dans le PICOLABORATOIRE décrit sur : http://www.robot-maker.com/ouvrages/apprendre-a-programmer-arduino-en-samusant/ c'était un écran monochrome qui sur la Fig.241 est présenté en **A**. Ayant commandé la même référence pour avoir un exemplaire de rechange en cas de "pépin", *(Et oui, ça arrive !)* j'ai reçu un modèle un peu différent. Comme montré sur la Fig.241 en **B** il est bicolore, avec une zone orange de 16 lignes en haut, une ligne de séparation sans pixel *(Repérée en rouge.)* et le reste inférieur en bleu. Du coup, avec le même programme on constate qu'en blanc sur la photographie **A** il y a continuité du tracé, alors qu'en bicolore sur l'image **B** il y a une discontinuité entre les deux couleurs.
Renvoyer le produit au fournisseur était une option, mais tout compte fait, la couleur rend l'écran bien plus attrayant. En tenant compte dans les programmes de la ligne sans pixels, on peut concevoir un logiciel qui favorise l'esthétique, les deux couleurs mettant en évidence des informations de type différent. La version en blanc est bien moins visuelle. *(ATTENTION : Les deux photographies présentées ici dégradent considérablement la finesse réelle de l'afficheur, car l'appareil de prise de vues n'est pas prévu pour prendre des clichés en objectif rapproché.)*

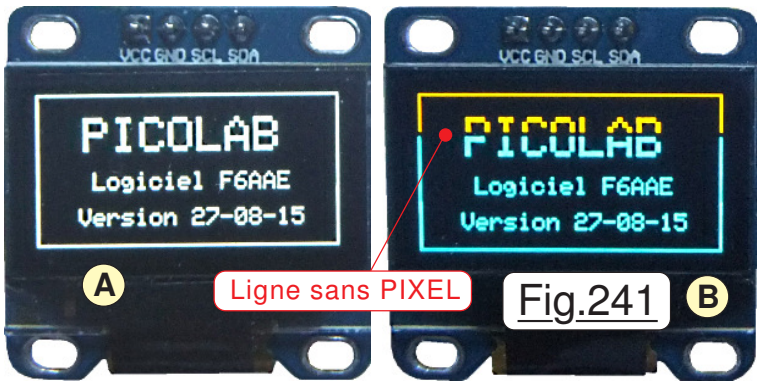


Fig.241

Personnellement je trouve que la couleur est très séduisante, et l'inconvénient de discontinuité faible puisqu'il n'est présent qu'en mode "dessin plein écran". Pour que chacun puisse choisir à sa guise le modèle d'afficheur utilisé, tous les programmes seront conçus sur l'afficheur bicolore. *(Prise en compte de la discontinuité.)* Ainsi les deux versions du composant seront totalement compatibles.

Deuxième difficulté : Fonctionnant en allumant et en éteignant des pixels, l'effacement de l'afficheur total ou partiel prend du temps. Quand on recherchera de la rapidité de rafraichissement sur l'écran, il faudra faire un gros effort logiciel pour que le code soit optimisé en rapidité d'exécution. Y compris dans ces conditions il ne sera pas possible d'obtenir la "fluidité" d'autres écrans du commerce. Rassurez-vous, mis à part quelques cas particuliers, et d'un usage marginal, dans l'ensemble les performances au final restent très "compétitives".

Choix de l'afficheur : Il existe dans le commerce une quantité impressionnante de composants pour afficher sur des matrices textuelles ou graphiques. Des monochromes, des multicolores, sous toutes les tailles et définitions, dont certains sont tactiles. Ayant expérimenté pas mal de ces périphériques, mon choix c'est porté sur l'afficheur **graphique** OLED pour plusieurs raisons. En particulier il est minuscule et convient à merveille pour le petit pupitre de commande. J'affectionne tout particulièrement le bicolore jaune et bleu d'où son adoption.

NOTE : Le didacticiel était rédigé jusqu'à la page 31. Puis, pas à pas j'ai décidé de développer globalement le logiciel. Ainsi, voyant mieux jusqu'où serait possible l'entreprise, les démonstrateurs de la version **C** à la version **P** ont été consciencieusement créés. Chaque version apportait des possibilités nouvelles dont il sera question dans les pages qui suivent. J'avais prévu

de vous faire tester les diverses améliorations en téléchargeant sur la carte NANO les démonstrateurs les uns après les autres ... BERNIQUE !
Ce n'est plus possible, car l'étude des ces programmes a exigé plus de quatre mois. Les modifications ont régulièrement chamboulé l'organisation des textes en EEPROM. Du coup, toutes les versions entre P22**C**_Démonstrateur_Sonde.ino et P22**E**_Démonstrateur_Sonde.ino sont inutilisables car les textes affichés sont incohérents. Ce n'est pas catastrophique. Le didacticiel sera bâti comme si ces démonstrateurs étaient disponibles, sauf que pour expérimenter les améliorations on va ignorer les versions **C**, **D**, **E** et télécharger directement la version **F** ce qui au final vous épargnera quelques manipulations routinières. Le programme complet et définitif nommé P40_PGM_RAQUETTE_MAITRE.ino contient en tête l'intégralité de l'historique. Vous pourrez ainsi, si vous le désirez, savoir combien coûte en termes d'octets de programme chaque amélioration et évolution logicielle. Notre première expérience va donc se faire en téléchargeant P21**F**_Démonstrateur_Raquette.ino et P22**F**_Démonstrateur_Sonde.ino qui ainsi nous font faire un "bond en avant dans le temps".

```
SONDEJEKERTVersion > MOUVEMENTS <>> POSTURES <<
OPTIONS SONDE.DONNEES SONDE.> EXPLOITER. <Repeter
  - DEPLACEMENTS -Couper les   ?Moteurs ON ?
Mouvements RAPIDES ?Stab. Gyroscope ?CLV
Mode APPRENTISSAGE ?
Version sur OFF : MOTEURS actifs Mvt. RAPIDE :
Sauver la POSTURE ?QUITTERBouclier au sol
APPRENTISSAGE : Bleu =Mauve=Rouge=Orang=x
SAV spectre couleur ?AFFOUINON
Energie : 1 a 254PHARES    LASER
Utiliser lexxTestReveiller Gradateur Phares.
LASER.Phares & LASER = 127VEILLEStable Raisonnable
Hauteur MaximaleMoteurs au Neutre OPApprise en EEPROM
AtterrissageDecollageDeposer JEKERTStable Transversal
ATTENTION : Retour par FIN (BERCEAU !)Rot fois
Jaune=Vert = Lumiere totale = hiberne : un balayage ?
TORSION active ?Pilotage MANUEL ?Moteur : JAMBE :
HancheGenouGriffePosture actuelle ?
```

Fig.241C

➢ *Les textes affichés à l'écran sont logés en EEPROM.*

ATTENTION : Les textes sont logés en EEPROM. Aussi, *avant de tester le démonstrateur* il faut dans un premier temps inscrire les diverses chaînes de caractères dans la mémoire non volatile avec P30_Ecrire_les_textes_en_EEPROM.ino. Pour ce faire, vous téléchargez le petit programme dans la carte Arduino. Puis vous faites appel au moniteur de l'**IDE** qui, comme montré sur la Fig.241C présente le contenu actuel de la mémoire non volatile, suivi du listage octet par octet sous forme d'une matrice de 64 lignes de 16 colonnes. Ce qui compte, c'est d'avoir placé les textes en EEPROM avant de téléverser le démonstrateur, car dans le cas contraire, les affichages seront incohérents. Fermons la parenthèse et revenons au didacticiel.

➢ *Protocoles d'exploitation de la sonde sur le terrain.*

Particulièrement délicat à élaborer, l'agencement d'un contexte d'utilisation d'un système quel qu'il soit se heurte à une difficulté particulière liée au fait que les divers MENUS qui permettront le pilotage sont liés à la présentation du pupitre. Hors pour disposer géométriquement ce dernier il faut savoir à quoi serviront les touches, donc avoir présent à l'esprit les protocoles envisagés. Bref, c'est le serpent qui se mord la queue. Dans la conception d'une Raquette de commande c'est de loin la phase la plus délicate car elle sera suivie de longues études logicielles qui seront remises en cause si ce que l'on a imaginé à ce stade aboutit à une NON convivialité et oblige à mettre à la poubelle des heures d'études logicielles. Pour ne pas risquer de gaspiller un temps considérable à des réalisations matérielles qui seraient perdues, on ne passera à la concrétisation que lorsque globalement tout sera défini. Pour l'instant, c'est le petit clavier malcommode expérimental géométriquement "matriciel" qui sera mis à contribution.

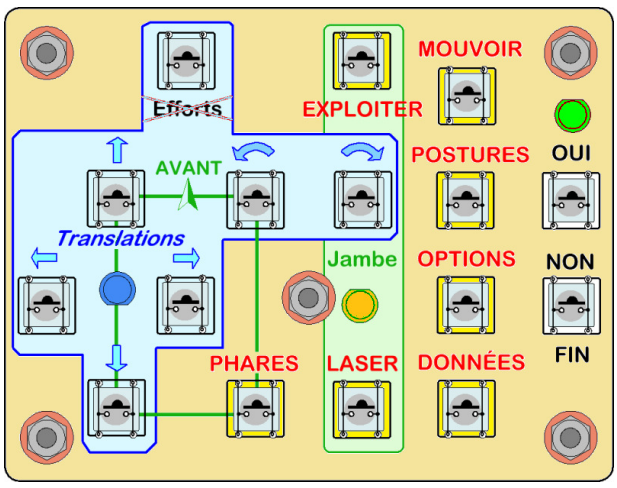


Fig.242

L'étude qui va suivre va s'efforcer de concevoir un pupitre "définitif", mais il est évident que seule l'expérience qui sera acquise au cours du développement logiciel confirmera son bienfondé. On doit donc s'attendre à des déconvenues qui obligeront à revoir entièrement notre copie, sachant que c'est statistiquement dans l'ordre des choses. Bon, il faut bien commencer ...
Une première réflexion fait apparaitre les divers modes fonctionnels suivants :

1) ***Consulter la sonde :*** Configuration système / Données scientifiques. *(Consignes passives.)*
2) ***Configurer les OPTIONS :*** Rapide/Lent, Moteurs OFF/Actifs, VEILLE etc.
3) ***Imposer une POSTURE.*** *(Peut obliger à vérifier la configuration initiale.)*
4) ***MOUVEMENTS :*** Déplacements de base translations/Rotations.
5) ***Mode EXPLOITATION des expériences scientifiques et des facultés d'apprentissage***.

Tributaires des positions des trous disponibles sur les plaques de circuit imprimés prépercées, les positions des divers boutons poussoir du clavier ne pourront pas se voir librement réparties. Les positionnements horizontaux et verticaux seront sur des alignements forcément "au dixième de pouce". Des études préalables pour s'assurer de la faisabilité ont induit le choix de l'accumulateur qui sera inclus dans la raquette pour assurer l'alimentation des deux cartes Arduino. Dans l'espoir de minimiser les dimensions du petit pupitre, les dimensions du clavier sont évaluées pour que ce dernier occupe le dessus de l'accumulateur et en adopte "la surface". La première possibilité représentée sur la Fig.242 est équipée de petits boutons poussoir économiques. On les trouve par paquets de cent pour quelques euros. Le rectangle délimitant le clavier respecte globalement les dimensions de l'accumulateur rechargeable envisagé. Idem pour la Fig.243 sur laquelle sont représentés des boutons poussoir plus couteux. Ils sont plus beaux, mais plus volumineux. Les couleurs sur le dessin ne sont pas celles qui seraient "logiques", mais celles des B.P. actuellement disponibles. Pour l'heure, aucune des deux solutions n'est choisie, c'est l'avenir qui fera pencher la balance d'un coté ou de l'autre. Les inscriptions non plus ne sont pas taillées dans le marbre. Il ne s'agit ici que d'une estimation rapide de ce pourra être le clavier. Cette esquisse va servir toutefois à dégrossir la géométrie du clavier, la disposition des touches et l'élaboration des protocoles d'utilisation du petit pupitre. On se doute qu'à l'expérimentation concrète nous serons certainement amenés à modifier ce qui nous sert de base de départ. Examinons les concepts généraux qui servent à débroussailler le terrain. Nous allons raisonner sur la version "de luxe".

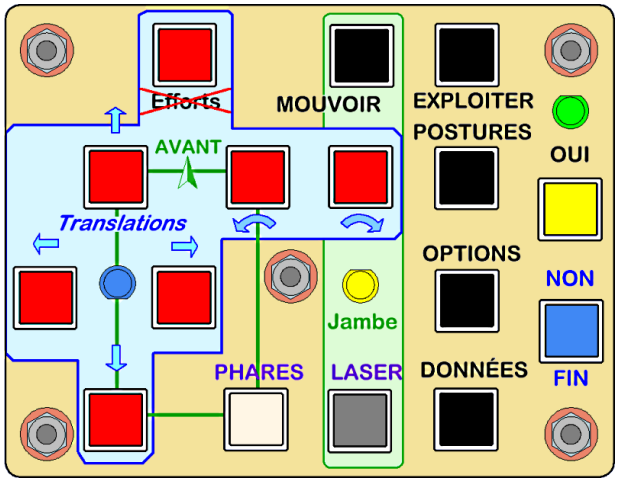


Fig.243

Pour faire court, dans l'état actuel des études primaires, ***les cinq touches noires seront affectées à l'appel des cinq menus de base*** correspondant aux cinq fonctions mentionnées ci-avant. Le fait de cliquer sur l'une de ces touches fait changer de mode. Elles auront cet usage en permanence sauf **MOUVOIR** pour le pilotage manuel des moteurs. **PHARES** et **LASER** servent à allumer ou éteindre ces deux périphériques. leurs fonctions ne changent pas sauf occasionnellement pour **LASER** lors du pilotage manuel des moteurs. Dans ce cas particulier, les trois touches "verticales" de la zone verte servent à définir l'articulation pilotée. Dans ce mode le bouton rotatif actionne les servomoteurs. La sortie de ce type de mode particulier se fait par la touche **FIN**.
Les sept touches rouges seront réservées principalement pour commander les déplacements élémentaires quand l'option **MOUVOIR** sera validée dans les menus. *(Sortie de ce mode par l'une des autres touches noires.)* Quand elles sont valides, la LED bleue s'allume.

Ponctuellement le programme demandera de confirmer ou d'infirmer une action, ou simplement valider l'un des items d'un menu déroulant en cours. Ce sont les deux boutons de droite. Celui du haut correspond au "risque maximum", c'est à dire l'accord. Celui du bas au refus. Notez que cette notion de "danger maximum" est aussi présente pour les touches noires. En partant du bas vers le haut les "conséquences" d'une activation augmentent. Par exemple le menu **DONNÉES** n'engage à rien, si ce n'est obtenir des informations sur l'écran. **OPTION** commence à modifier le comportement de la sonde. On ne modifie que des bascules de type OUI/NON.

POSTURE engage déjà plus fortement, car à partir d'ici la sonde va se mettre en mouvement avec tous les risques que cela engendre. Pour clore ce rapide tour de ce qui est envisagé à ce stade, tourner le codeur rotatif aura des effets qui seront totalement différents en fonction du mode en cours. Nous verrons les détails au fur et à mesure du développement des programmes. Les grandes lignes d'utilisation pratique du clavier étant tracées, il faut les valider ce qui ne peut se faire qu'expérimentalement, donc avec des démonstrateurs codés en C++ dans l'environnement de l'**IDE**.

➢ *Affectation des touches du clavier.*

Pouvant influencer notablement la complexité du code compilé, l'affectation des touches doit privilégier la compacité du code. Par ailleurs, le développement des programmes va exiger un nombre considérable d'essais avec un clavier non ergonomique au point de vue mnémonique. De paire avec l'impact sur la taille du code objet, il importe également de faciliter le travail du programmeur. Pour tenir compte de ses deux critères, la Fig.244 montre les choix qui ont été effectués. Placer les touches de fonction entre **S1** et **S5** permet de simplifier certains traitements , donc de minimiser la taille des démonstrateurs. L'ordre des affectations facilite la mémorisation des touches pour le programmeur car elles sont géométriquement disposées comme sur le clavier "définitif". OUI et NON sur **S7** et **S8** est prévu pour le programmeur. Les touches en "diamant" **S9**, **S11**, **S14** et **S6** seront faciles à retenir pour effectuer les translations. Du coup "Annuler les Efforts" est placé au centre. Pour avoir les rotations l'une à coté de l'autre il ne restait de disponible que **S16** et **S12**. Enfin, ayant ainsi réparti les touches il ne reste plus que **S13** et **S15** qui sont dans l'ordre dévolues aux bascules d'allumage des Phares et du LASER. Ainsi réparties, on optimise en partie le code objet, tout au moins pour le codage des cinq touches de fonction. Pour éviter les effets boule de neige, ***à partir d'ici on ne modifiera plus ces affectations***. Il ne nous reste qu'à espérer que ces choix ne compliqueront pas trop la réalisation du circuit imprimé. Il faut maintenant concrétiser cette stratégie dans P21**F**_Démonstrateur_Raquette.ino et effectuer les essais avec intégration des premiers menus déroulants ce qui n'est jamais une mince affaire. La gestion d'un l'écran graphique autorise bien des plaisirs esthétiques ... au prix parfois de quelques "crises de nerf !

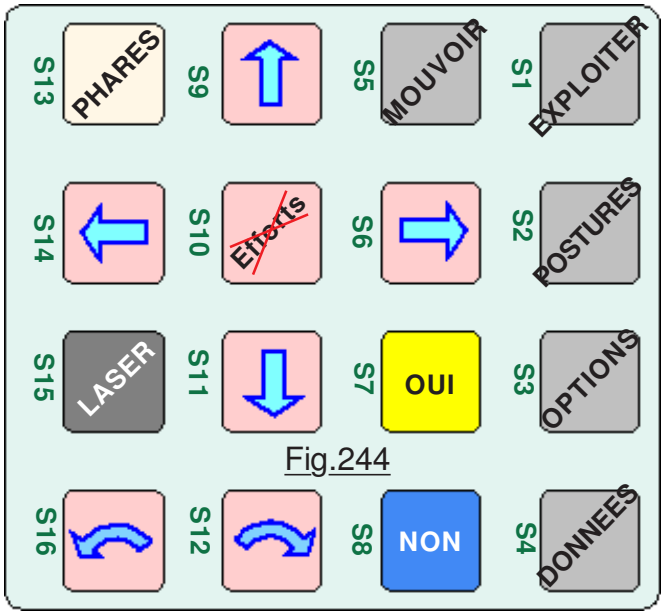


Fig.244

➢ *Le MENU des DONNÉES.*

Pour pouvoir interroger la petite machine sur son état système ou sur les valeurs mesurées par ses différents capteurs, il importe d'avoir la main sur les options. Aussi, avant de développer le menu des DONNEES SONDE historiquement le menu des OPTIONS SONDE a été élaboré en premier. La vérification de l'effet des consignes était réalisée sur les LEDs affectées aux différents paramètres. Le programme P21**F**_Démonstrateur_Raquette.ino présenté dans ce chapitre gère les deux menus. Il s'utilise conjointement avec le démonstrateur précédent P22**F**_Démonstrateur_Sonde.ino qui était prévu pour répondre à certaines consignes des menus "déroulants". Lorsque le programme démarre, il commence par afficher l'écran d'accueil de la Fig.245 et ***attend un clic sur le clavier***. Pour nous en informer, ***la LED CLAVIER verte clignote rapidement***. Dans le cadre jaune est indiquée la version du programme qui anime la Raquette de commande. Dans le cadre bleu en grand la "présentation du système". Dès que l'on appui sur l'une des seize touches la LED cesse de clignoter et l'écran présente alors le "menu déroulant des DONNEES SONDE. *(Bien que son développement ait été effectué après celui des options, c'est le premier menu qui s'active lors de la mise sous tension.)*

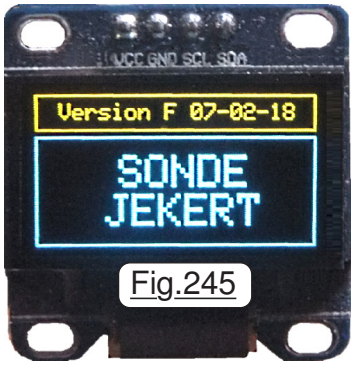


Fig.245

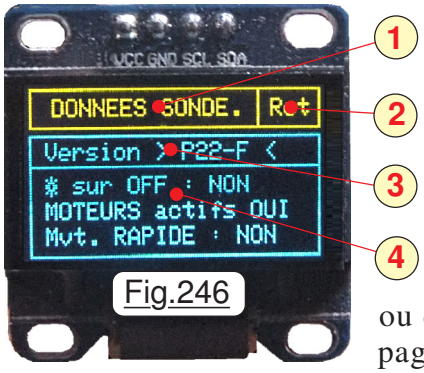


Fig.246

Représenté sur la Fig.246 l'écran des informations est structuré en quatre zones. Dans le rectangle jaune en **1** le titre du MENU en cours. En **2** le périphérique à utiliser pour changer d'item. Rot indique le codeur rotatif incrémental, CLV le clavier. En **3** se trouve une information d'ordre général. Ici c'est la version du programme Esclave qui est indiquée. Enfin en **4** nous trouvons trois informations de la configuration système ou des données disponibles sur les capteurs scientifiques. En tournant le codeur Rotatif dans un sens ou dans son contraire on fait défiler en permutation circulaire les quatre pages-écran disponibles. Les pages défilent dans l'ordre des photographies Fig.246, Fig.247, Fig.249 et la Fig.250 de présentation différente. Concrètement, entre les écrans de la Fig.249 et celui de la Fig.250 vient s'interposer l'affichage représenté sur la Fig 249B mais qui dans la réalité n'a été introduit quavec cette version P22F_Démonstrateur_Sonde.ino qui des démonstrateur. La donnée de CAP sera étudiée de ce fait dans un autre chapitre.

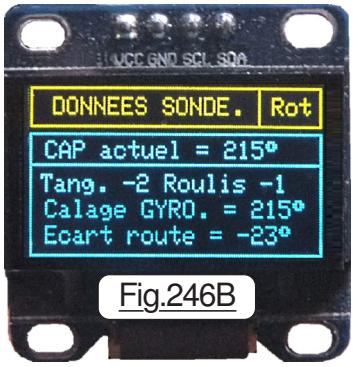


Fig.246B

En **A** est précisée la distance télémétrique de l'obstacle éventuel situé devantla sonde. En **B** sont réunie les trois paramètres météorologiques. Si le capteur d'humidité n'est pas installé sur le connecteur, les deux mesures engendrent une ERReur n°3 ou n°4 en **C** avec en **D** le non affichage du texte HYGROMETRIE = nn %. En **E** est indiquée la tension d'alimentation des servomoteurs avec en **F** trois options système. Le dernier écran répartit de façon différente ses données. Pour des facilités de lecture les quatre informations utiles sont regroupées par deux en **H** dans le rectangle du bas. Du coup la zone d'information annexe **G** étant libre, on y précise la plage de variations possibles pour la valeur des énergies que l'on peut consigner aux deux périphériques à l'aide du gradateur. *(Potentiomètre virtuel qui était non exploitable dans la version **C** du démonstrateur.)* Il importe de noter qu'il aurait été possible de mémoriser les valeurs des booléens d'état dans la Raquette, évitant ainsi d'interroger la sonde ce qui diminuerait le nombre des codes et allègerait d'autant le programme objet de la sonde. Cette approche ne serait pas fiable, car on peut envoyer un ordre à la sonde, et que suite à un incident quelconque il ne soit pas exécuté. *(Problème de transmission ...)* Il est donc impératif que ce soit la sonde qui fournisse les états des divers paramètres.

Fig.247

Fig.248

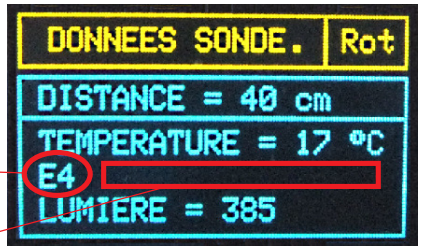


Fig.249

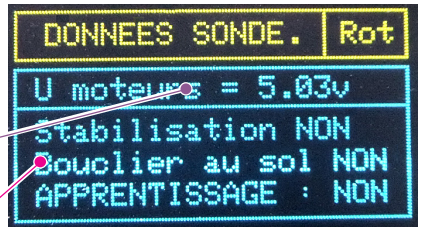
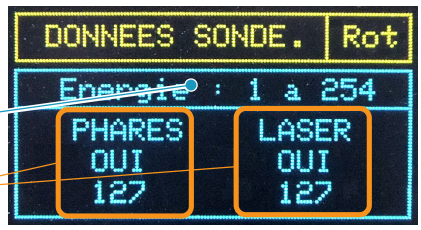


Fig.250



➢ L'ordre des pages écran du menu des DONNEES.

L'agencement d'un menu ne doit pas être considéré à la légère. De sa convivialité dépend directement la qualité opérationnelle "du produit". Chaque fois que l'on valide la fonction DONNEES SONDE il importe d'ouvrir la page la plus "utile". Dans cette étude c'est celle qui correspond à la gestion des moteurs et de l'énergie. Ensuite, entrant dans ce menu, les items qui sont voisins doivent privilégier des pages "importantes en exploitation". Si on tourne dans le sens positif, *(Sens horaire.)* on ouvre la page des mesures scientifiques. Une rotation dans le sens négatif ouvre les informations des périphériques consommant beaucoup d'énergie, c'est à dire les Phares et le LASER. C'est donc celle des options et informations sur la stabilisation et l'APPRENTISSAGE qui se trouve la plus "éloignée en rotation" de la fenêtre d'entrée. Notez au passage que les affichages ralentissent considérablement le processeur, car le composant OLED étant graphique doit entièrement générer des dessins qui pour vous sont des caractères de texte. Il faut aussi effacer

des zones ce qui impose de blanchir beaucoup de points. Aussi, quand vous tournez le codeur ne soyez pas trop rapides, laissez le temps à l'affichage de finir de tracer les informations concernées avant de tourner fébrilement d'un nouveau cran. Notez au passage que le symbole  visible sur la Fig.146 et qui sera rencontré sur plusieurs écrans représente un Soleil. Il symbolise l'alimentation en énergie de la ligne qui peut être disjonctée et qui allume en particulier les phares, le LASER et les nombreuses LEDs situées sur la sonde.

➢ Le MENU des OPTIONS.

Concrètement, certaines sont vérifiables par deux possibilités. Par exemple l'allumage des Phares et du LASER sont observables visuellement et par interrogation de la sonde. Moteurs figés la LED rouge confirme l'information disponible avec le menu DONNEES SONDE. Quand le programme sera au point, la crédibilité consistera à enlever le petit "strap" à languette pour éteindre toutes les LEDs système. Il faudra alors faire confiance aux télémesures, le réalisme est à ce prix.

Beaucoup plus épurés que pour le menu des données, les écrans pour les OPTIONS SONDE ne présentent qu'un seul item à la fois. Dans le rectangle jaune, Rot précise que l'on changera d'option avec le codeur rotatif. Sur la Fig.251 est précisé en **1** l'option en cours de saisie et en **2** son état actuel. ***La LED CLAVIER verte clignote rapidement*** précisant que le programme P21**F**_Démonstrateur_Raquette.ino attend une saisie au clavier. La logique veut que l'on clique sur la touche OUI ou sur le bouton poussoir NON. Ces deux touches auront l'effet escompté. Toutes les autres touches du clavier seront ignorées et n'auront aucun effet mis à part l'arrêt du clignotement tant que le bouton poussoir reste enfoncé. Immédiatement après avoir sollicité la touche **S7** ou la touche **S8** l'état actuel du booléen concerné est mis à jour en **2**. *(Par exemple, ici la ligne des énergies n'est pas disjonctée.)* Tourner le codeur incrémental dans un sens ou dans l'autre fait changer d'item dans la liste des options. La permutation circulaire est naturellement fonction du sens de rotation. Les cinq items du menu des options sont présentés par ordre croissant sur les Fig.251, Fig.252, Fig.253, Fig.254 et Fig.255 dont la qualité des images laisse parfois à désirer. Plusieurs origines en sont la cause. L'écran est petit et les images sont prises en macrophotographie, mon appareil numérique n'étant pas idéal dans ce domaine. Obligation de prendre au flash ce qui fait ressortir toutes les poussières. Par ailleurs l'écran OLED est multiplexé, c'est à dire que ces PIXELs sont balayés rapidement et non lumineux de façon constante. Par exemple on constate bien sur la Fig.255 que le texte bleu est tout juste lumineux en bas de la ligne du centre alors qu'il y a surexposition sur la ligne du bas. Je vous prie donc de bien vouloir accepter ces images parfois assez peu visibles. Et encore, elles sont considérablement retouchées à la main pour ajuster le cadrage horizontal, optimiser la lumière et la concentration et surtout une foule de petites tâches blanches générées par les poussières sont enlevées.

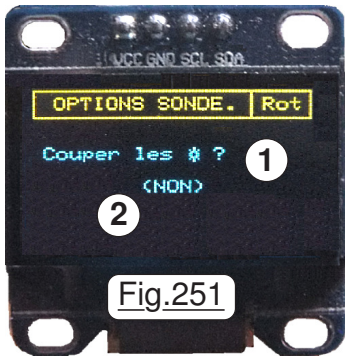


Fig.251

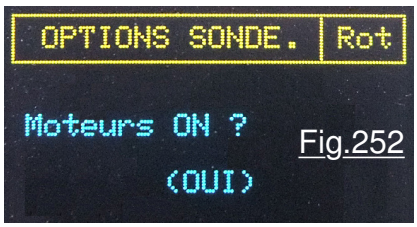


Fig.252

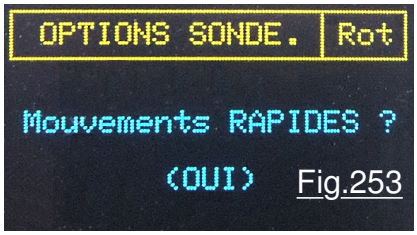


Fig.253

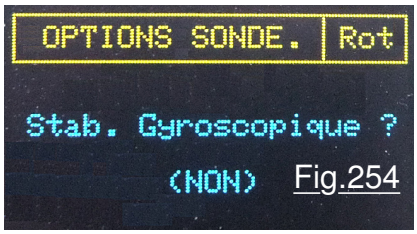


Fig.254

Fig.255

➢ Quelques éclaircissements sur le démonstrateur.

Contrairement à ce laisserait entendre le chapitre sur L'ordre des pages écran du menu des DONNEES, l'afficheur OLED n'est pas le seul à ralentir la vitesse d'exécution du programme. N'oubliez-pas que tourner le codeur rotatif ou cliquer sur OUI ou NON impose au programme de dialoguer avec le calculateur de bord. Par exemple OUI ou NON impose d'envoyer le code Consigne correspondant à l'item d'option affiché et d'attendre l'accusé de réception. Dans ce chapitre nous allons passer en revue deux ou trois petits détails logiciels.

Retour sur les interruptions : Cherchant à mettre au point les séquences des menus, le codeur rotatif s'est montré particulièrement pointilleux, son fonctionnement s'avérait aléatoire accompagné de très nombreux blocages. Quand on le tournait dans un sens ou dans l'autre il ne se passait rien. Il manquait dans la procédure void **setup**() les instructions suivantes :

A_set = false; A_set = false; et pinMode(Codeur_A, INPUT); pinMode(Codeur_B, INPUT);

Curieusement elles n'étaient pas dans le démonstrateur précédent qui pourtant semblait fonctionner correctement. Il est vrai qu'utilisé sur le gradateur lumineux, si des "clics" ont été perdus c'était bien moins visible qu'un item affiché à l'écran qui refuse de changer.

Début de la séquence qui traite le clavier :

```
① Tester_le_clavier();
② if (BP > 0) {
③   if (BP < 6) {Mode_en_cours = BP; Num_des_DONNEES = 1; Num_des_options = 5;
④                 Affiche_Ecran_MENU();} // Boutons des MENUS à validité permanente.
⑤   if (BP == 13) {PHARES = !PHARES;
⑥                  if (PHARES) Envoyer_consigne(27); else Envoyer_consigne(28);}
⑦ ..... suite .....
```

La ligne ① teste si une touche est enfoncée. Si c'est le cas elle retourne sa valeur dans BP. Si aucune touche n'est cliquée, elle retourne la valeur 0 et dans ② le test est négatif, tout ce qui suit dans le bloc du if est ignoré. En ③ toutes les valeurs de BP comprises entre 1 et 5 seront prises en compte. C'est la raison pour laquelle on a placé les "fonctions MENUs" entre S1 et S5, un seul test détermine alors d'un seul coup toutes les touches de "fonctions MENUs". Dans ce cas, avant d'invoquer le menu déroulant correspondant avec en ④ l'instruction Affiche_Ecran_MENU() on ***impose la page sur laquelle il sera ouvert***. Les autres touches ont un codage assez banal. Par exemple en ⑤ on inverse le booléen PHARES qui fait office de bascule de type OUI/NON. Puis, en fonction de son état on transmet sur **TX** en ⑥ la valeur de consigne qui allume ou éteint les phares.

Un petit détail dans la procédure du MENU_des_OPTIONS :

```
  void MENU_des_OPTIONS() {
① ..... Incrémente ou décrémente Num_des_options .....
② switch (Num_des_options) {
     case 1 : {Aff_TEXTE_EEPROM(114, 14); break;}  // Mode sommeil.
     ..... suite .....
     case 5 :  Aff_TEXTE_EEPROM(181, 20);}        // Mode apprentissage.
③ BIT_Option = Num_des_options; BIT_Option--; if (BIT_Option == 4) BIT_Option = 7;
④ Affiche_etat_option_en_cours();}
```

La prosédure en ① commence par incrémenter ou décrémenter Num_des_options. Puis avec le switch / case en ② elle procède à l'affichage de l'item pointé dans la liste. La ligne ③ a pour but d'affecter à la variable BIT_Option une valeur correspondant dans Chaine_Memorisee à la position du "0" ou du "1" représentatif du booléen correspondant au case. Pour en saisir l'écriture, il importe de résumer ce que fait en ④ la procédure Affiche_etat_option_en_cours :

Elle commence par envoyer la Consigne n°44 à l'Esclave sur la sonde qui retourne l'état des booléens système dans la variable Chaine_Memorisee de l'ACR. Ils sont dans l'ordre suivant :

Sommeil	Moteurs	Rapide	Stabilisation	Bouclier au sol	Phares	LASER	Apprentissage
0	1	2	3	4	5	6	7

La variable Chaine_Memorisee est donc composée de huit caractères de type booléens.

Chaque booléen est donné sous la forme du chiffre "0" ou "1" avec les conventions de représentation habituelles. En orange sont indiqués l'indice de position dans la chaîne de caractères.

Détaillons les instructions de la ligne ③ :

```
BIT_Option = Num_des_options; BIT_Option--; if (BIT_Option == 4) BIT_Option = 7;
           ⑤                  ⑥                ⑦
```

case	ordre booléen	Instruc ⑤	Instruc ⑥
1	0	1	0
2	1	2	1
3	2	3	2
4	3	4	3
5	7	5	4

Fig.256

Le tableau de la Fig.256 résume l'évolution des acteurs agissant sur BIT_Option lors du déroulement du programme. L'instruction de la ligne ② laisse Num_des_options égal à l'ordre du case qui traite l'affichage des données. En ⑤ on affecte cette valeur à BIT_Option. En décrémentant de 1 dans l'instruction ⑥ on constate que mis à part la valeur 4 les autres valeurs sont correctes. Elle est corrigée en ⑦.

Pourquoi ce traitement particulier alors qu'un classique switch / case montré sur le listage de la Fig.257 serait plus parlant à lecture du programme ? La réponse réside dans ce leitmotiv permanent du programmeur : Optimisation du code. L'utilisation de la ligne ③ fait économiser trente quatre octets ce qui n'est pas rien. N'oubliez jamais que pour atteindre un but dans un logiciel,

Fig.257

```
switch (Num_des_options) {
  case 1 : {BIT_Option = 0; break;} // Mode sommeil.
  case 2 : {BIT_Option = 1; break;} // Moteurs OFF.
  case 3 : {BIT_Option = 2; break;} // Mouvements rapides.
  case 4 : {BIT_Option = 3; break;} // Stabilisation Gyroscopique.
  case 5 :  BIT_Option = 7;}        // Mode apprentissage.
```

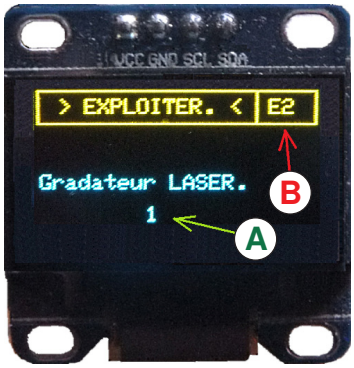
il existe souvent de nombreuses façons différentes de l'atteindre. L'art de programmer c'est choisir la plus élégante en fonction de critères hiérarchisés. Tout en haut on trouvera la minimisation du code objet, puis la réduction de la taille des variables dynamique, enfin la lisibilité du programme source. Choisir avec précision de type des variables et l'ordre des données dans les tableaux ou dans les chaînes de caractères fait partie intégrante de cette stratégie omniprésente.

51) 5/01/2018 : La carte des menus (MJD 58123)

Finir l'année 2017 a fortement éprouvé les personnels de la NDRMSE, un repos s'avérait impératif pour ne pas trop abuser de leur dévouement à la cause. Aussi, six jours de repos ont été octroyés pour toutes les personnes travaillant sur le site. Ça fait du bien, on retrouve avec plaisir la frénésie des lieux. Les lumières ont été allumées, les ordinateurs remis en service, et dans S4 les informaticiens soudent à nouveau leur regards sur les écrans saturés d'informations abscondes. Et dire qu'en ce moment, si loin dans le vide sidéral, la petite exploratrice hiberne d'un profond sommeil.

➢ Changement de stratégie.

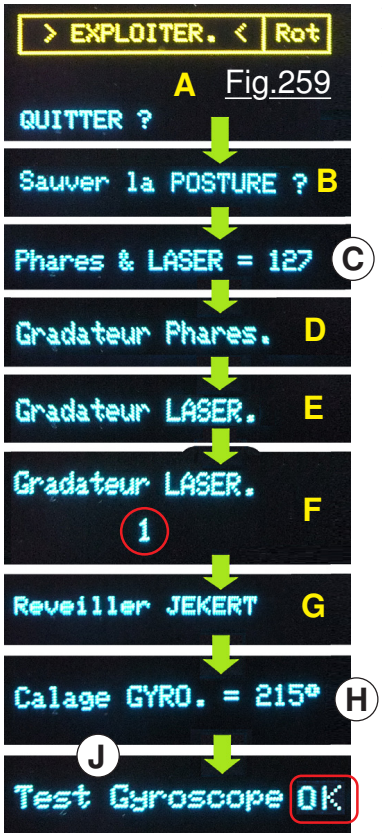
Fig.258



Caractérisant une nouvelle façon plus agréable d'utiliser le petit écran, le démonstrateur P21**F**_Démonstrateur_Raquette.ino associé à P22**F**_Démonstrateur_Sonde.ino marque une évolution qui va dans le sens d'une plus grande convivialité opérationnelle. Si une erreur se produit, sa référence est maintenant indiquée dans le petit cadre jaune en haut à droite de l'écran. Par exemple sur la Fig.258 nous somme en train de diminuer la luminosité des phares. Arrivé à la valeur 1 comme on peut le voir en **A**, si l'on persiste à tourner le codeur rotatif dans le sens négatif, un bip sonore nous avertit que le codeur est en butée logicielle. C'est en **B** que l'on peut alors avoir l'information sur la nature de l'erreur commise. Le texte d'erreur s'efface et sera remplacé par Rot ou CLV dès que le bouton rotatif change d'indexage. Notez au passage qu'avec cette version du démonstrateur, la LED ajoutée sur le dispositif de la Fig.237 s'allume pour indiquer que les moteurs sont en vitesse lente.

➢ Le menu EXPLOITATION.

Logiquement, avec celui des déplacements ce sera en principe le plus utilisé. Dans cette version du démonstrateur on n'émule que huit fonctions, par la suite on va en ajouter encore plusieurs. L'ouverture du menu EXPLOITATION se fait sur l'item QUITTER. On pourrait penser que ce n'est pas très logique, et qu'au contraire, ce sont les fonctions de mise en service de JECKERT qui devraient se trouver en tête de liste. En fait, comme nous ne sommes qu'à la phase développement logiciel, pouvoir rapidement replacer la sonde dans une configuration propre est prioritaire.



Dans la version ultime du programme on commencera la liste par une fonction plus opérationnelle. Sur la Fig.259 nous avons les différents écrans qui se succèdent quand on tourne le codeur rotatif dans le sens horaire. Nous savons que QUITTER ? en **A** replace la sonde sur son bouclier puis coupe les énergies. L'item en **B** sauvegarde la posture actuelle en EEPROM. Cette fonction est utile lorsque l'on a passé pas mal de temps à créer une configuration originale et que l'on désire la retrouver par la suite à tout moment. Sur un RESET, les phares et le LASER sont initialisés au minimum d'énergie. Par l'option **C** du menu > EXPLOITER < on force les deux niveaux énergétique à 127, c'est à dire la moitié de la puissance maximale possible. La fonction suivante en **D** propose le gradateur pour les phares, suivi en **E** du gradateur pour le LASER. Que ce soit pour les phares ou pour ajuster la puissance du LASER, quand on valide la fonction avec le bouton poussoir **OUI**, immédiatement, comme dans le cercle rouge sur l'image **F**, est indiquée la valeur actuelle. On se trouve alors en ***Mode gradateur***. Lorsque l'on tourne le codeur dans un sens ou dans l'autre, la valeur de la puissance change avec effet immédiat sur les périphériques matériels. Tenter des valeurs inférieures à 1 ou plus grandes que 254 génère un BIP sonore d'avertissement et affiche E1 ou E2 dans le cadre jaune. Si la sonde n'est pas en hibernation, les LED blanche ou jaune dédiées s'allument alors sur la sonde. Pour ne pas avoir à trop tourner le bouton du gradateur, les incrémentations et les diminutions de niveau d'énergie se font par pas de 18 unités. Une bonne pratique consiste à forcer les niveaux à 127 si l'on désire rapidement des éclairages notables. C'est la touche **FIN** qui fait sortir du ***Mode gradateur***. Pour que sur le pupitre nous puissions en avoir confirmation, la valeur numérique de la puissance actuelle est alors effacée de l'écran sur la Raquette de commande.
La fonction Reveiller JEKERT en **G** fait sortir la sonde du mode hibernation qui lui avait été imposé par la commande QUITTER.

QUITTER réalise les actions suivantes :
- Place la Sonde au sol sur le bouclier. *(Active les moteurs si ils sont sur OFF.)*
- Force les moteurs sur OFF, et la puissance à 1 sur les phares et sur le LASER.
- Désactive le mode APPRENTISSAGE si actif et éteint la LED rouge sur la sonde.
- Coupe le disjoncteur des énergies, ainsi que les phares et le LASER.
- Désactive le mode TORSION manuelle ainsi que la stabilisation gyroscopique.

Reveiller JEKERT n'est pas la réciproque et ne réalise que les actions suivantes :
- Réarme le disjoncteur pour rétablir les énergies.
- Réactive la motorisation sur ON.
- Impose la configuration Stable_Transversal et Libère les efforts.

La fonction Calage GYRO en **H** impose la ***Référence interne*** correspondant à l'orientation actuelle. L'item indique en complément le CAP magnétique actuel. Si on valide cette option avec le bouton poussoir **OUI**, l'IMU6050 est réinitialisée et éventuellement le CAP est affiché à la nouvelle valeur. *(La version **F** du démonstrateur présente un petit "bug" : La valeur ancienne n'est pas effacée, les deux valeurs se superposent. Ce petit détail est corrigé dans les versions qui suivent.)*
Quand la centrale gyroscopique a été recalée, l'écart de route qui est affiché dans les données de navigation sera calculé à partir de cette ***Référence interne***. Tout changement d'orientation en Lacet sera alors détecté et indiqué avec une précision d'un degré angulaire. Nous verrons plus avant que l'information gyroscopique est plus précise que l'utilisation du compas de route, mais se trouve affectée par la rotation terrestre ...
Enfin, la dernière fonction Calage GYRO proposée en **J** provoque une vérification de la centrale IMU6050. Si on valide avec la touche **OUI**, après un court délai l'accusé de réception OK s'affiche à la suite du texte Calage GYRO comme montré dans l'encadré rouge.

➢ Le "menu" MOUVEMENTS.

Impératif pour pouvoir exploiter sur le terrain la petite machine, le menu pour télécommander des déplacements est assez particulier. À franchement parler, ce n'est pas un menu mais une fonction purement opérationnelle. Du reste, quand on tourne le codeur rotatif il n'y a pas de changement d'item sur l'écran de maitrise de la lointaine exploratrice. C'est la seule option de pilotage pour laquelle dans le petit cadre jaune il y a affichage en **1** de CLV au lieu de Rot. Pour bien nous informer que dans ce mode c'est le clavier qu'il faut utiliser, la LED verte clignote. Considérons sur la Fig.260 la page écran relative à cette fonction. Le titre > MOUVEMENTS < en **2** précise sans ambigüité que l'on va imposer à la sonde de se déplacer. Pour le cas où vous n'auriez pas totalement pressenti le danger, le logiciel insiste en **3** avec l'avertissement ATTENTION - DEPLACEMENTS -. De plus, pour le cas où ce n'est pas encore assez clair, la LED rouge du "BIP" des alertes sonores clignote en permanence signalant donc un DANGER potentiel. Pourquoi une telle obstination ?
On ne pilote pas du tout une sonde comme une automobile. Entre le moment où l'on télécommande une consigne et celui où la sonde la réalise puis rend compte, il peut se produire un délai jusqu'à une heure pour recevoir l'ACR. C'est un peu comme si vous conduisiez votre automobile en ouvrant un court instant les yeux, en analysant la route, en corrigeant au volant puis en fermant les mirettes durant une heure sans savoir ce qui se passe ! Aussi, pour ne pas risquer de faire tomber la machine dans un ravin, on analyse avec finesse les images issues de la caméra de bord. Puis on décide soit de changer de CAP, soit de faire un déplacement élémentaire. Si le terrain est plat, on peut se risquer à enchainer plusieurs étapes, mais on ne cherchera jamais à couvrir une longue distance d'un coup, c'est contraire aux protocoles astronautiques.
Pour des raisons de réalisme, on s'en tiendra à de telles procédures. Néanmoins, il vous sera facile de titiller frénétiquement les boutons de déplacement envisagés sur la Fig.243 pour déplacer de façon notable et à vue directe le petit insecte mécanique.

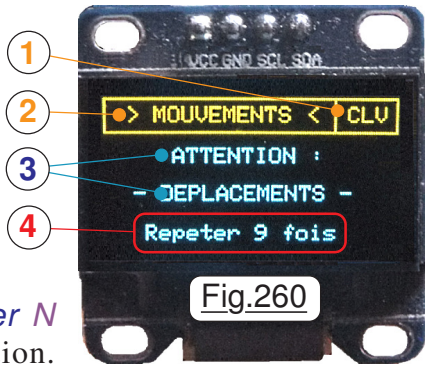


Fig.260

Quand on tourne le codeur rotatif, le nombre *N* indiqué par *Repeter N fois* change entre *1* et *9* dans un sens qui dépend de celui de la rotation. Lorsque vous déclenchez un déplacement élémentaire, en n'appuyant qu'une seule fois sur la touche désirée, la machine va automatiquement enchaîner *N* fois le déplacement demandé au robot. Aussi, vous allez rapidement constater que *9* fois c'est beaucoup et vous prendrez assez vite la bonne habitude de replacer à *1* le facteur de répétition, sauf cas particulier. Par exemple j'utilise la répétition de *9* pour effectuer les test d'autonomie sur un accumulateur, ce genre de chose ... encore que dans ce cas il vaut mieux faire apprendre à la machine un long programme comportant tout une suite de mouvements qui se succèdent. La sonde étant alors sur un berceau on déclenche le programme qui anime les moteurs sans avoir à surveiller JEKERT. Notez au passage que sur un RESET le facteur de répétition est forcé à *1*. En revanche, si vous modifiez cette valeur, elle sera conservée dans la mémoire du système quand vous quittez les déplacements. Lorsque vous revenez dans la fonction > MOUVEMENTS <, l'ancien facteur ajusté pour la répétition est retrouvé.
Quand on impose un déplacement, *(Avec répétition ou non.)* immédiatement les deux LEDs cessent de clignoter et le codeur rotatif devient inerte. On a l'impression que le système s'est bloqué. Ce n'est en rien un aléa de programmation. Le pupitre a envoyé à la sonde une directive. Après avoir parlé, n'oublions pas que nous sommes en alternat, le logiciel passe à l'écoute et attend un accusé de réception issu venant de Mars. Tant que la ligne **RX** reste muette, le programme espère avec patience. Lorsque JEKERT a entièrement effectué le ou les déplacements élémentaires, alors seulement elle transmet sur **TX** son accusé de réception. La raquette de commande redevient disponible ce qui visuellement se voit par le clignotement des LEDs qui reprend sa cadence de métronome. Aussi, il peut se montrer agassif de déclencher un déplacement alors que l'on a oublié de replacer à *1* le facteur de répétition. C'est la raison pour laquelle personnellement je ne l'utilise que très peu.
Dans ce mode on peut librement allumer ou éteindre les PHARES et le LASER, mais pas en changer la puissance avec le gradateur. *(N'oubliez-pas non plus que piloter un allumage ne sera effectif que si le disjoncteur n'est pas déclenché.)* La seule façon de sortir consiste à appuyer sur l'une des toutes de MENU ... par exemple celle des POSTURES que l'on va aborder maintenant.

➢ Le menu des POSTURES.

Bien qu'engendrant des mouvements sur la machine, changer de posture ne génère pas de risque de voir JEKERT tomber dans un ravin ou heurter un obstacle. Les changements de configuration ne provoquent pas de déplacement à proprement parler, ils se font "sur place". Ces adoptions d'attitudes ne constituent pas des déplacements d'exploitation, ce sont des configurations particulières qui ne sont justifiées que dans des cas singuliers peu fréquents. C'est la raison pour laquelle ces commandes sont regroupées à part dans le menu >> POSTURES << pour ne pas trop encombrer la fonction déjà bien chargée des > MOUVEMENTS <. La Fig.261 présente l'écran d'accueil du menu qui propose en premier la configuration VEILLE. Dans la pratique c'est la consigne qui impose à JEKERT de poser son bouclier sur le sol pour libérer les efforts dans les jambes. Cette désignation est très mal choisie, car on peut confondre avec l'hibernation qui consiste à couper les énergies sur le bus des "lumières". Dans la version finale on remplacera ce texte par un item plus évocateur.

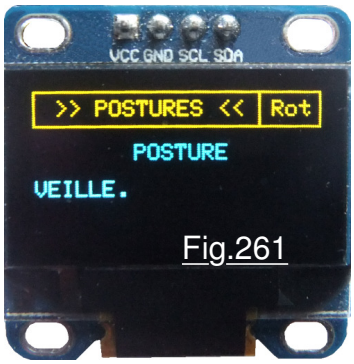


Fig.261

Fig.262

L'écran suivant, quand on tourne le codeur rotatif dans le sens horaire, et visible sur la Fig.262, est plus "dynamique" en ce sens qu'il s'accompagne d'un BIP sonore. Cette alerte a pour but d'attirer l'attention de l'opérateur sur la ligne du bas en **X** qui nous instruit sur l'importance de faire suivre cette consigne, si on la valide par **OUI**, par un retour à la configuration standard en utilisant la touche **FIN**. ***En effet, certaines postures particulières imposent un retour à une configuration de base par un mouvement coordonné spécifique.*** Les écrans sont dans ce cas complétés par cette ligne **X** assortie d'un BIT d'alerte sonore. La posture Stable Raisonnable correspond à la plus grande surface de sustentation possible, attitude que l'on pourra adopter lorsque la sonde est sur un terrain en forte pente pour en assurer une meilleure stabilité. L'écran de la Fig.263 correspond à l'adoption de la Hauteur Maximale pour laquelle les Tibias et les Griffes sont en orientations verticales. Il s'agit également d'une posture qui doit être suivie d'une touche **FIN** pour revenir à la normale par un mouvement coordonné spécifique. Puis, tournant le bouton toujours dans le sens horaire, les écrans de la Fig.264 vont se succéder dans l'ordre des photographies. Les configurations **1** et **2** sont sans particularité. Faire attention toutefois à POSTURE Apprise en EEPROM qui suppose que dans la mémoire du microcontrôleur on a logé au moins une fois une configuration cohérente. Si l'EEPROM est vierge et n'a été chargées que par les postures de base et les textes, la zone réservée à tout ce qui concerne l'apprentissage ne contient que des $FF. Si vous déclenchez l'option **2** les servomoteurs vont diverger.

Fig.263

Fig.264

Quand on affiche l'écran **3** un BIP d'alerte se fait entendre pour attirer notre attention sur le fait que la sonde doit impérativement se trouver sur son (BERCEAU !). Si elle est au sol quand vous validez cette configuration ... les moteurs ne vont pas apprécier. Le programme teste pour vérifier que le micro-contacteur de présence "du sol" est bien activé. Si ce n'est pas le cas un E7 va remplacer Rot, accompagné d'un autre BIP d'alerte. Même punition en **5** si Deposer JEKERT est validée et que le contact avec le sol n'est pas effectif. Les deux postures **4** et **6** n'appellent pas de commentaires particuliers. Il nous arrivera assez souvent de confirmer la POSTURE Stable Transversal, comme pour VEILLE du reste. Pour faciliter l'accès rapide à ces deux configurations fondamentales, VEILLE est directement en entrée dans l'écran d'accueil de >> POSTURES <<, et **6** s'obtient alors par un simple indexage en sens anti-horaire sur le codeur incrémental.

➢ Compte rendu relatif à la version F des démonstrateurs.

Comme déjà expliqué très en amont de ce chapitre, les versions **C**, **D** et **E** des démonstrateurs ont été écartés car inutilisables avec la table des textes actuelle. À titre d'information, ce chapitre explicite les fonctionnalités qui ont été émulées par P21**F**_Démonstrateur_Raquette.ino et P22**F**_Démonstrateur_Sonde.ino qui ont servi à manipuler pour le dernier paragraphe :

* Installation de la centrale inertielle et nouvelle stratégie d'utilisation. Sur RESET il n'y a plus d'arrêt si l'initialisation est négative. À tout moment on peut tester la communication avec le code consigne 58. Si la centrale ne réagit pas à des changements d'orientation, on peut suspecter un problème. Dans ce cas on teste manuellement la communication. Le test engendre une erreur E8 si la centrale n'est pas opérationnelle. Toute tentative d'utiliser l'IMU devra être précédée d'un diagnostic de validation de la centrale. La calibration gyroscopique doit se faire au moins quinze à vingt secondes après avoir testé la liaison pour que la température IMU soit stabilisée.
* Des textes ont été passés de la Raquette à la Sonde pour gagner de la place sur le logiciel du pupitre. Cette stratégie est adoptée car pour l'heure elle semble favorable "à la répartition des encombrements".
* L'écran des données de navigation est effectif dans cette nouvelle version. *(Données gyroscopiques.)*
* Certaines directives imposent automatiquement le mode "Moteurs Lents" : Réveiller la sonde, RESET de la sonde, Configurer en Stable raisonnable, Poser au sol ...
* QUITTER procèdes aux diverses action décrites dans le chapitre qui précède, ainsi que la fonction Reveiller JEKERT qui est codée dans ce démonstrateur.

Les menus sont globalement en place, il importe maintenant de compléter le logiciel du pupitre.

52) 8/01/2018 : En voir de toutes les couleurs (MJD 58126)

Globalement, les consoles de maîtrise du suivi de JEKERT commencent à fonctionner correctement. Les procédures les plus importantes ont passé les tests de validation. Manifestement, en S4 les ingénieurs logiciels commencent à se détendre, ils sont visiblement moins soucieux. Il reste encore beaucoup de code à écrire. Ceci dit, si le planning reste très chargé, tous savent que dans l'état actuel la sonde pourra débarquer en toute sécurité. Le calendrier sera respecté au point de vue "balistique". Ce n'est pas une raison pour lambiner. Aussi, on n'entend pratiquement que le ronron apaisant des ordinateurs et les cliquetis frénétiques sur les claviers. Pour ce chapitre, ce sont P21**G**_Démonstrateur_Raquette.ino et P22**G**_Démonstrateur_Sonde.ino qui doivent être téléversés sur les deux cartes NANO Arduino. Il est évident que la route est encore très longue avant d'aboutir à un programme complet. Les embuches seront nombreuses, les essais innombrables et laborieux. Il importe de prévoir des outils d'aides logicielles pour faciliter l'analyse :

➢ Quelques outils logiciels pour aider le programmeur.

Développer un programme qui exploitera les possibilités graphique de l'afficheur OLED est manifestement un projet très ambitieux. On va forcément se "cogner à du dur de dur", avec à la clef un comportement du programme strictement incompréhensible. Dans une telle situation, il faut que le programmeur puisse placer des "points de test" qui déclenchent une alerte quand le processeur déroule des instructions qui viennent d'être ajoutées dans le Source et qui manifestement ne produisent pas l'effet escompté. La première question à se poser est :

- Le programme passe t'il vraiment dans la procédure ajoutée au programme ?

Ce n'est pas évident, surtout si la subroutine est invoquée suite à une combinaison un peu complexe de booléens. Une parenthèse mal placée, un ET à la place d'un OU ... qui ne s'est jamais fourvoyé dans de telles instructions ? Pour savoir si le programme emprunte bien un chemin particulier, il suffit d'introduire provisoirement l'instruction BIP(); **de la Sonde**. Si le buzzer reste silencieux et que la procédure devrait se voir activée ... c'est qu'il y a un problème en amont, la séquence n'est pas en cause. Si il y a un BIP, alors le diagnostic s'oriente vers un questionnement du genre :

- La procédure est bien invoquée, quelle est alors la valeur de la variable X en entrée / sortie ?

Pour suivre les variations d'une variable dans une séquence de code quelconque, il suffit d'en faire afficher la valeur sur l'écran OLED, peu importe l'endroit sur sa matrice de pixels. L'idée consiste dans ce cas à ajouter au programme une subroutine spécifique à laquelle on fait appel à des emplacements stratégiques dans le programme. Comme ce type d'action sera

rencontré souvent, autant rédiger une procédure "propre". À partir de la version **G** des démonstrateurs nous aurons à notre disposition la routine de servitude :

```
//------------------------------------ Outil de mise au point ----------------------------------
void Pister_un_PB() { // @@@@@@@@@@@@@@@@@@@@@@@@@@
  Effacer_zone(64,4,10); display.setCursor(64,4);
  display.print(Variable pistée); display.update(); delay(1000);}
//------------------------------------------------------------------------------------------------
```

L'instruction display.update(); est indispensable pour que la donnée soit affichée à l'écran. Le délai d'une seconde n'est utile que si l'on fait appel plusieurs fois à cette procédure. Il faut nous laisser le temps de voir les diverses valeurs évoluer durant le déroulement du programme. Enfin, comme pour plusieurs autres lignes stratégiques dans le programme, la remarque composée d'une ribambelle de @@@@@@@@@@@@@@@@ permet facilement de retrouver la séquence avec une recherche de cette chaîne de caractères dans Édition > Trouver ...

- Nom d'un Octet mal placé, j'en ai ma dose de cheminer dans les menus jusqu'à la nouvelle option à tester, je vais finir par destroyer mon beau codeur rotatif !

Pour gagner un temps considérable, on impose durant le développement d'ouvrir le menu en cours de mise au point directement sur l'Item à tester. Dans ce but, il suffit de modifier la constante adéquate dans les déclarations. Lorsque l'on désire revenir à la page d'accueil normale, on rétablit la constante et le tour est joué :

```
#define Num_initial_donnees = 1                 //@@@@@ (1) @@@@@@@@@@@
#define Num_initial_options = 5                 //@@@@@ (5) @@@@@@@@@@@
#define Num_initial_Exploitation = 4            //@@@@@ (6) @@@@@@@@@@@
#define Num_initial_Num_des_postures = 9  //@@@@@ (9) @@@@@@@@@@@
```

Par exemple, la valeur 4 est provisoirement substitué au 6. Pour pouvoir rétablir facilement le fonctionnement prévu en standard, dans les lignes de remarque sont précisées les valeurs prévues pour la version ultime. Nous avons les outils nécessaires pour poursuivre nos études. Notons au passage que cette version **G** fait appel à un changement de gestion des fonctions QUITTER et Reveiller JEKERT : On quitte les dialogues entre sonde et pupitre par la consigne (61) qui positionne le booléen Hibernation à true. Reveiller JEKERT par la consigne (55) le repasse à false.

➤ *Changement de stratégie.*

Forcément, les nombreuses campagnes de tests font émerger des difficultés d'exploitation. Aussi, et ce n'est pas la première fois du reste, il faut changer les protocoles, revoir l'implantation du pupitre etc. Pour cette version du démonstrateur, plusieurs virages s'imposent. Un changement important concerne l'affichage du spectre colorimétrique qui fait l'objet du chapitre suivant. De plus, on se rend inévitablement compte que l'usage du Clavier / Codeur rotatif devient confus. Quatre LEDs vont assister l'opérateur : Une Verte qui clignote et signale une attente clavier pour passer à la suite. En impulsif elle témoigne de l'appui sur une touche quelconque. Une LED rouge signale une erreur, *("BIP optique".)* ou que le B.R. fonctionne en gradateur lumineux ou que le menu > MOUVEMENTS < est actif agissant sur la motorisation. Une LED jaune signale qu'une fin de fonction doit être impérativement effectuée avec la touche **FIN**. Enfin une LED bleue non clignotante signale le mode > MOUVEMENTS <. Les couleurs du rouge au bleu sont fonction de la "dangerosité" de la configuration signalée par ces témoins lumineux.

> Résumé : Si la LED jaune clignote, c'est juste que la fin d'une fonction doit se faire avec la touche **FIN** du clavier. Si en plus "> Retour par FIN" et BIP sont déclenchés, la touche **FIN** est impérative pour éviter un problème en motorisation. Donc ne pas changer d'item ou de MENU sans avoir au préalable **FIN**i proprement la fonction en cours.

Certaines fonctions sont "muettes". Quand on les valide avec **OUI** il ne se passe rien sur l'écran OLED et l'on peut douter de leur prise en compte par la sonde. Pour lever le doute, ces dernières génèrent le faux code d'erreur E99 qui signale que la consigne a été effectuée avec succès. Cette information comme tout accusé de réception de type erreur issue de la sonde provoque un BIP d'alerte. Les fonctions concernées par les fausses E99 sont les suivantes :

• Calage du gyroscope. *(Car si le CAP magnétique affiché ne change pas lors de cette manipulation, rien n'évolue à l'écran.)*
• Phares et LASER forcés à 127.
• Sauvegarder la posture actuelle en EEPROM.

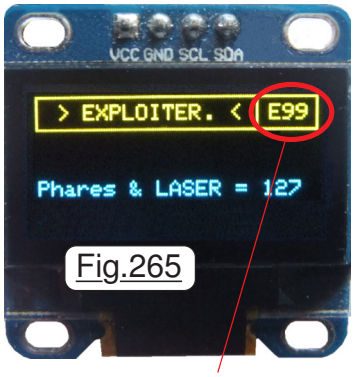


Fig.265

E99 est une "fausse erreur"

➢ Traitement du spectre colorimétrique.

Enregistrer la lumière totale ainsi que six couleurs différentes imposent un écran particulier pour afficher toutes ces données. Le démonstrateur **G** met en place les routines pour pouvoir dans le menu > EXPLOITER. < enregistrer un spectre couleur, *(Voir Fig.266 **A**)* ou le faire afficher. *(Commande de la Fig.266**B**)* La Fig.267 présente la façon dont sont listées les diverses données d'un enregistrement spectrométrique. Un petit point de détail qui a son importance doit être souligné : Quand la Jambe **A** vient vers l'avant pour que le dernier secteur couleur verte soit au dessus de la cellule photorésistante, la ligne d'alimentation des moteurs du Genou et du Pied peut venir au dessus du capteur de lumière. De ce fait la mesure serait faussée et les rapports de sensibilité entre les divers filtres ne seraient plus respectés. Pour éliminer ce petit problème on réunit toutes les lignes des sorties **S1** à **S5** en un seul toron comme montré sur les photographies de la Fig. 268 et sur Image 57.JPG conduisant à un parfait dégagement vers le haut. Remarquant que les informations issues d'un spectre colorimétrique constituent des valeurs "scientifiques", on devrait assez logiquement les obtenir dans le menu des DONNEES SONDE. Après diverses tentatives pour évaluer la commodité réelle des solutions adoptées, il ressort que placer cette page dans le menu > EXPLOITER. < est plus judicieux dans la mesure où cet item suit celui de la SAuVegarde du spectre colorimétrique.

Fig.266

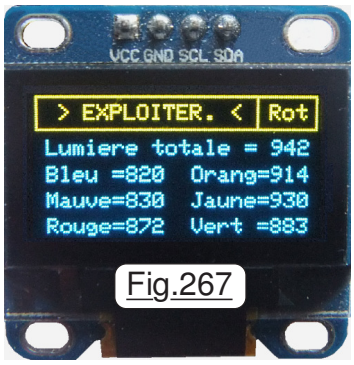


Fig.267

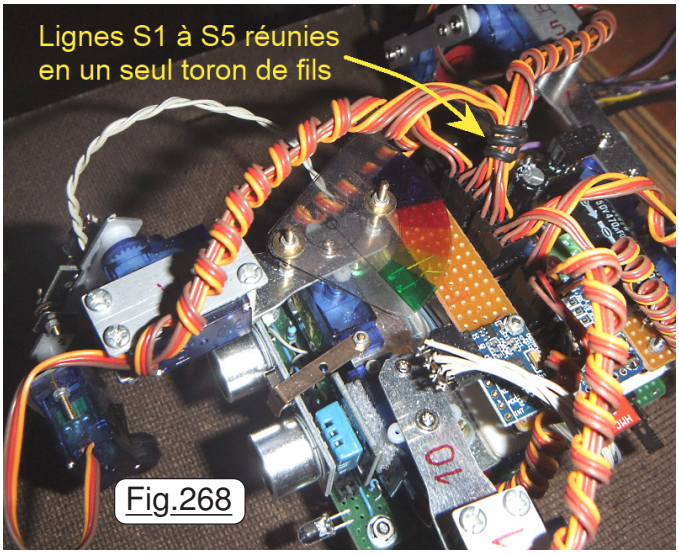


Fig.268

➢ Fonctions ajoutées à la version G.

Étant en phase de développement, il importe d'avancer avec prudence, c'est à dire que chaque version sauvegardée doit faire évoluer le programme, mais raisonnablement. Ainsi, si pour une quelconque raison une "catastrophe" se produit et qu'il faut repartir sur de nouvelles bases, revenir en arrière est plus facile. Toutefois, l'excès contraire doit aussi se voir éviter, chaque étape devant apporter un nombre suffisant d'améliorations. Conciliant prudence et mutation notable, la Fig.269 présente les nouvelles fonctions du menu > EXPLOITER. < qui maintenant ouvre sur la page Reveiller JEKERT. Puis, Test Gyroscope suit et se trouve actuellement avant Calage GYRO car dans l'optique d'une reprise d'activités, cet ordre est plus cartésien. Arrivent alors les fonctions "éclairages". C'est à leur suite que l'on trouve les fonctions listées sur la Fig.269 qui commencent par Utiliser le LASER. Comme il faut prendre en compte les problèmes de sécurité pour les personnes qui éventuellement se trouvent dans la pièce où se trémousse la sonde, consultez le **Manuel d'utilisation** *(1)* à la page **➢ Protocole d'utilisation du LASER**. *(ATTENTION : Le livret a été rédigé avec la version "définitive" du logiciel. La version actuelle **G** n'est pas forcément 100% compatible.)* Les valeurs des balayages angulaires possibles sont détaillées dans le chapitre ***Gestion du pointage LASER***

Fig.269

(1) Cet deux livrets sont fournis avec les fichiers du TOME 6.

Fig.270

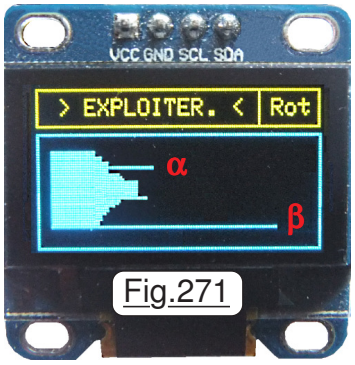


Fig.271

disponible dans le livret de maintenance nommé DOSSIER TECHNIQUE. *(1)* Après l'usage du LASER on arrive à l'émulation de la fonction TORSION active. Le codeur incrémental engendre du Lacet relatif dans une direction qui est fonction du sens de rotation.
ATTENTION : On peut aller dans une configuration avec interférence de certains éléments des Jambes sur des organes du châssis. Il n'y a pas de détection des collisions, donc surveiller visuellement et surtout ne pas exagérer la déviation latérale.
Les commandes pour réaliser un panoramique télémétrique sont présentées sur la Fig.270 respectivement en **A** pour l'enregistrement en EEPROM, et en **B** pour sa visualisation. L'approche pour effectuer la saisie des échantillons *(Résumée sur la Fig.272)* est un peu différente de celle du développement initial inspiré par des manipulations mettant en œuvre un potentiomètre matériel. Les valeurs retournées par le

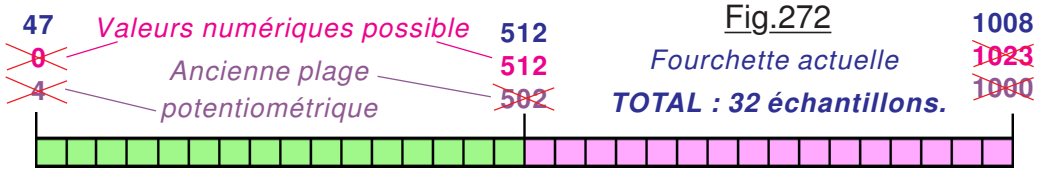


Fig.272

convertisseur analogique numérique s'étendaient alors de [4 à 1000]. La valeur pour le centrage avait été de ce fait fixée à 502. Pour balayer la totalité du spectre, on procédait par des incréments de 15 unités. Avec l'approche numérique, on peut à notre convenance utiliser la pleine plage binaire [0 à 1023]. Il devient alors naturel de placer le centrage à la moyenne de 512. En observant l'affichage sous forme graphique montré sur la Fig.271 on peut déduire que la définition verticale du cadre bleu ne laisse à notre disposition que quarante trois lignes de pixels. Trente deux enregistrements optimisent l'aspect programmation et sont suffisants vu la médiocre ouverture angulaire des transducteurs ultrasons et la présence de parasites comme ceux présents en **α** et **β** par exemple. On optimise dans ce cas l'amplitude du balayage par des pas de 31 unités. Répartis à partir du centrage, les valeurs extrêmes seront donc comprises dans la plage [47 à 1008] bornes comprises.

53) 11/01/2018 : Crash informatique à la NDRMSE *(MJD 58129)*

Arrivant à l'ouverture de la salle informatique, comme tous les matins depuis quelques semaines, il est manifeste pour moi qu'un incident d'actualité perturbe le personnel. Les ingénieurs logiciel sont tous regroupés à la machine à café, ce qui est rare, les bavardages vont bon train.

- Bonjour les gars, zavez-l'air bien excités ce matin, qu'arrive t'il dans la chaumière ?
- ZARIA a claché cette nuit. L'orage à fichu la foudre sur le complexe.
- Ouais, et directement sur le paratonnerre qui n'a pas suffit.
- Le serveur a un peu trinqué, heureusement qu'il est protégé par des éclateurs à gaz. Les électroneux ont réinitialisé, effectué le rechargement des sauvegardes, ZARIA refonctionne.
- Et alors, où est le prob ?
- Ben hier c'était l'anniversaire de Pierre, on a débauché à 18h, du coup on a oublié d'effectuer la sauvegarde. Normalement c'est automatique, mais pas de bol, ZARIA était en maintenance de niveau quatre, sa ligne de transfert sur le serveur de sauvegarde était provisoirement suspendue.
- Pas tragique, ceci dit la version H est perdue. Heureusement que J inclus toutes ses modifs.

➢ ***L'histoire se répète.***

Phénomène bien connu des historiens, ce n'est pas parce que l'on a décortiqué en détails les guerres passées que l'on peut empêcher des nouvelles. Ce n'est pas parce que l'on sait maintenant pourquoi de grandes civilisations brillantes ont disparues, que l'on décide de ménager la Terre sur laquelle repose notre futur. L'évolution logicielle de JEKERT est un peu à cette image. L'optimisation à outrance du code a conduit à modifier les textes logés en EEPROM. À partir de la version **J** il faut utiliser P35_Ecrire_les_textes_en_EEPROM.ino pour inscrire les bavardages d'OLED dans la mémoire non volatile du microcontrôleur. Conséquence : Cette version **H** n'est plus compatible et ses affichages écran sont erronés ... tant pis, on oublie et on téléverse **J**.

➢ Version définitive des textes logés en EEPROM.

Rassurez-vous, c'est l'ultime version qui ne sera plus remise en cause. Quand vous allez téléverser P35_Ecrire_les_textes_en_EEPROM.ino sur la carte NANO Arduino et valider l'affichage sur l'écran du contenu inscrit en EEPROM, vous constaterez que cette fois la totalité de cette ressource matérielle est consommée. Les 1024 octets sont occupés. Hors, le but consiste à décharger le programme des chaînes de caractères qui consomment de la mémoire dynamique. On peut alors affirmer sans grand risque de se tromper que :

- ***Il n'est plus possible de gagner un seul octet en EEPROM***,
- ***Donc modifier son contenu ne présentera aucun intérêt***, saut si l'on désire remplacer un texte par un autre et dans ce cas ***la substitution devra présenter une taille identique***,
- Sauf petite modification locale consistant à remplacer le texte d'un Item, à partir de maintenant les nouvelles chaînes de caractères devront résider dans le programme ... au détriment de la mémoire dynamique. ***IL FAUDRA IMPÉRATIVEMENT MINIMISER les nouveaux textes***.

(La version 'I' des démonstrateurs n'a jamais été envisagée, le l pouvant être confondu avec le '1'.)

Téléversez P35_Ecrire_les_textes_en_EEPROM.ino sur l'ATmega328 et validez le Moniteur de l'**IDE** pour visualiser le contenu de son EEPROM. Le résultat ressemble à la copie d'écran montré sur la Fig.272 sur laquelle le premier octet de la dernière ligne est "repéré en rose". Son adresse est 1008. On constate que le dernier caractère est bien en 1023, la mémoire EEPROM du processeur de la raquette est entièrement saturée. On voit aussi que l'on a passé en mémoire non volatile le mot OK de deux caractères ce qui semble un tantinet ridicule. Il faut bien se rendre compte que pour faire afficher un texte, on doit passer deux paramètres à la procédure de service Aff_TEXTE_EEPROM(1008,2). On s'attendrait à ce que la pénalisation en taille du programme rende déraisonnable cette approche informatique. Pourtant, quand on effectue des tests on constate que :

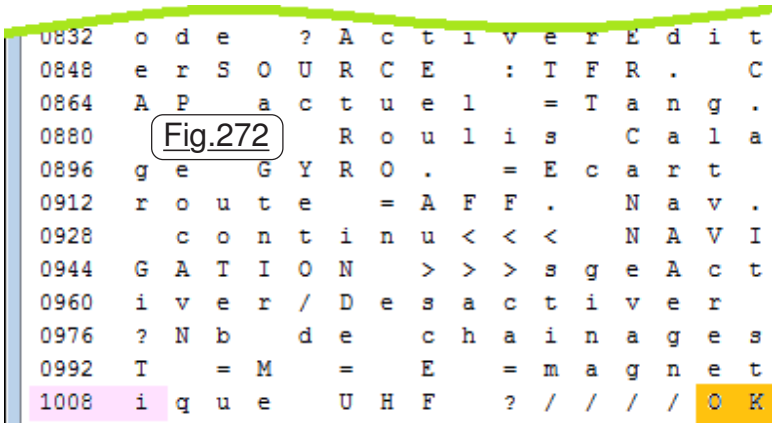


Fig.272

- L'instruction Aff_TEXTE_EEPROM(1008,2) consomme exactement 8 octets dans le programme,
- L'instruction display.print("OK") en exige 10 dans le programme, et surtout diminue la zone de mémoire dynamique de deux emplacements plus le $00 de fin de chaîne.

CONCLUSION : Durant la programmation, on constate que placer une chaîne de caractères, même très courte, en EEPROM est rentable. Il est donc raisonnable de chercher à saturer cette dernière par des textes quand le logiciel est "bavard" comme c'est le cas pour la raquette.
Le développement a été conduit jusqu'à son "aboutissement" engendrant dans les versions ultimes un problème de COLLISION entre la PILE et le TAS. *(Un chapitre y sera réservé.)* De ce fait, tenter de gagner ne serait-ce qu'un seul octet en mémoire dymamique s'est avéré IMPÉRATIF.

Scratch... Boum !

Chères lectrices, chers lecteurs, j'ai déjà précisé dans mes écrits que mon logiciel de P.A.O. accuse les années. Il ne tolère pas des fichiers trop volumineux. Avec cette page 41 il a marqué ses limites et refuse d'aller plus loin.

Désolé de me voir obligé de terminer ici le TOME 5 alors que l'aspect logiciel est loin d'avoir été entièrement abordé. Ce n'est pas bien grave. On va gentiment refermer cet ouvrage, la suite sera contenue sans autre contrariété dans le TOME 6.

Je vous souhaite à toutes et à tous agréable lecture et ... à bientôt dans le TOME 6.

Amicalement : Nulentout.