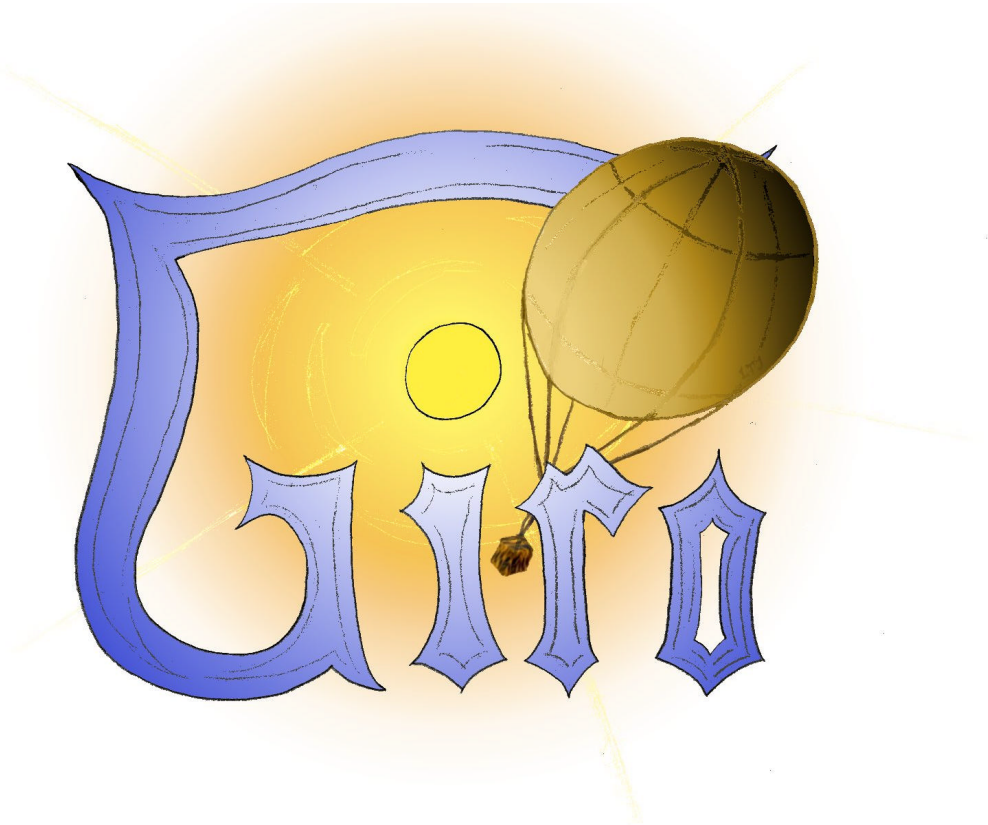
CLES-FACIL

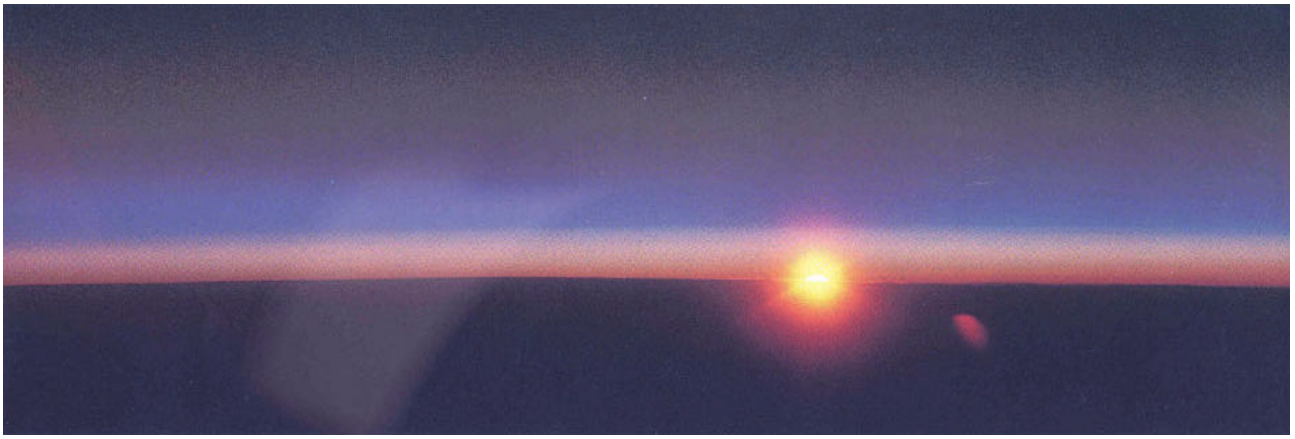
INSA LYON



Compte rendu de Projet 2002-2003

Introduction :

Ce projet a été mené sur une période de 7 mois de Octobre 2002 à Juin 2003 par 7 membres du CLES FACIL. Il a été initialement prévu d'être lancé pour la manifestation étudiante des « 24H de l'INSA », avec un éventuel repli durant les « festiciels de Vaulx en Velin », journée de découvertes scientifiques organisée par l'association nationale Planète Sciences. Mais les difficultés rencontrées lors du projet nous ont finalement poussé à retarder une dernière fois le lancement. Après des améliorations notables et l'obtention d'une finalisation acceptable du projet, c'est à la campagne de lancement nationale de Sissonnes (03), le 1er Août 2003, que le projet de ballon stratosphérique Giro du CLES-FACIL a pris son essor pour les hautes couches de l'atmosphère.



➢ *Ont contribué à la réalisation de ce compte-rendu :*

- **Stéphane Simon**, président 2002-2003 du Cles Facil et chef du projet Giro, pour la partie mécanique et analyse des résultats de l'expérience.
- **Ludovic Bertrand** pour la partie électronique et améliorations envisageables.
- **Olivier Bompis** pour la partie informatique et améliorations envisageables.

Sommaire

Présentation générale

Description du projet

Le ballon est né de l'idée de mettre en pratique une boucle d'asservissement afin de d'agir sur un phénomène mécanique, ici en l'occurrence la rotation de la nacelle du ballon. Par le biais d'une mesure d'un phénomène physique, on a l'opportunité de « donner vie » à un système. Grâce à la mesure d'éclairement de trois capteurs de lumière, nous avons réalisé un asservissement en rotation d'une nacelle supportant une cellule photovoltaïque. Avec cette cellule constamment orientée face au soleil par l'asservissement, il était possible de recharger nos accumulateurs. Nous avons donc formalisé nos idées pour définir les expériences validées par planète Sciences

Définition des expériences :

Optimiser la quantité d'énergie réceptionnée par une cellule photovoltaïque et suivre le ballon pendant son vol.

Expérience Principale :

⇒ *Optimiser la quantité d'énergie servant à la recharge de nos accus et renvoyer les informations des capteurs par télémesure.*

Expériences secondaires :

⇒ *Etre en autonomie énergétique. (Icharge > Iconsomme).*
⇒ *Suivre la position du ballon.*

Solution proposée :

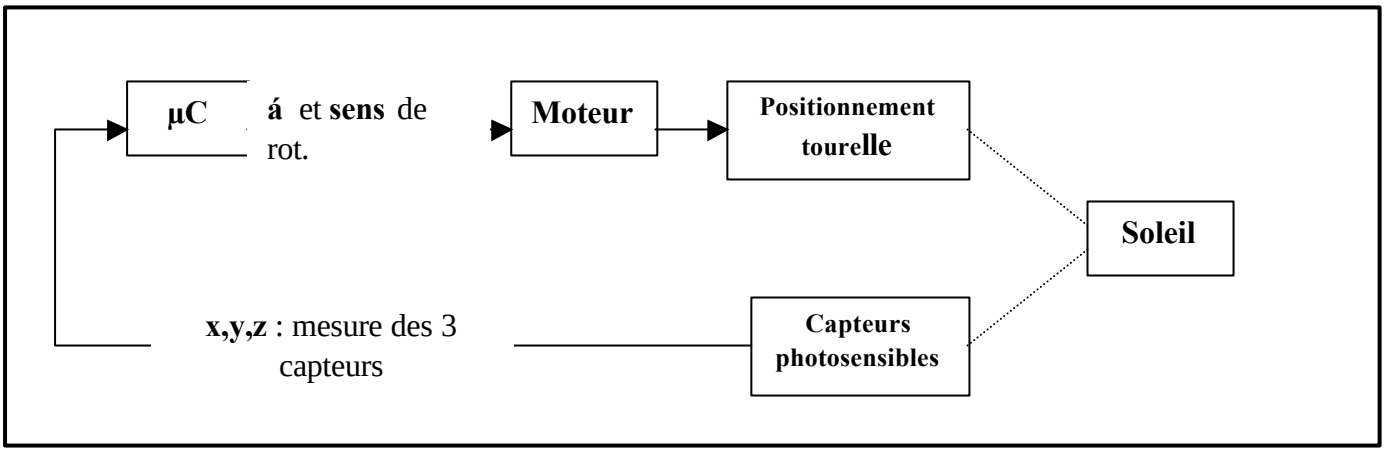
Afin de réaliser ces expériences, nous avons trouvé des solutions techniques répondant aux objectifs précédents :

- Pour optimiser l'énergie fournie par des cellules photovoltaïques, nous les orientons dans la direction du soleil. Nous avons fait l'hypothèse que sur le vol, le ballon resterait suffisamment vertical pour considérer que l'orientation des cellules photovoltaïques se ferait suivant un seul axe de rotation: l'axe vertical suivant la chaîne de vol. On a donc considéré que le principal problème rencontré lors du vol serait la rotation de la nacelle suivant sa verticale (soumise au vent). Il faut donc que le panneau solaire soit le plus possible dans la direction du soleil.

Boucle d'Asservissement :

- Nous avons donc déterminé la position relative soleil / panneau solaire grâce à 3 cellules photovoltaïques liées au panneau solaire visant dans 3 directions à 120° chacune.
- Un microcontrôleur s'occupe des calculs pour déterminer la position du soleil par rapport aux capteurs et donne une consigne de commande sur le sens et la vitesse du moteur.

- Le moteur, commandé par le microcontrôleur, oriente le panneau solaire. On a donc le schéma d'asservissement suivant :



Réception des résultats :

- Pour tout lancé de ballon, on prend le risque de ne jamais retrouver la nacelle et les expériences embarquées. Nous avons donc choisi de transmettre les informations des mesures et des corrections apportées avec l'asservissement par une liaison radio (une *télémesure*) utilisant l'émetteur de Planète Sciences *kiwi millénium*. Une expérience secondaire consistait à suivre la position du ballon par à un *Global Positioning System*. Pour des raisons techniques, et par manque de temps, cette fonctionnalité n'a pas été implantée dans notre nacelle.

Présentation de l'équipe

- Pour réaliser cette aventure nous étions une équipe de 6. La plus part d'entre nous étions novice et avions peu d'expérience pour la réalisation de ballon sonde. Nous nous sommes répartis la réalisation de la nacelle de la manière suivante :

✓ Deux personnes pour la mécanique de la Nacelle :
Josef Lajda.
Stéphane Simon.

✓ Deux personnes consacrées à l'électronique :
Floriand Raynaud.
Ludovic Bertrand.

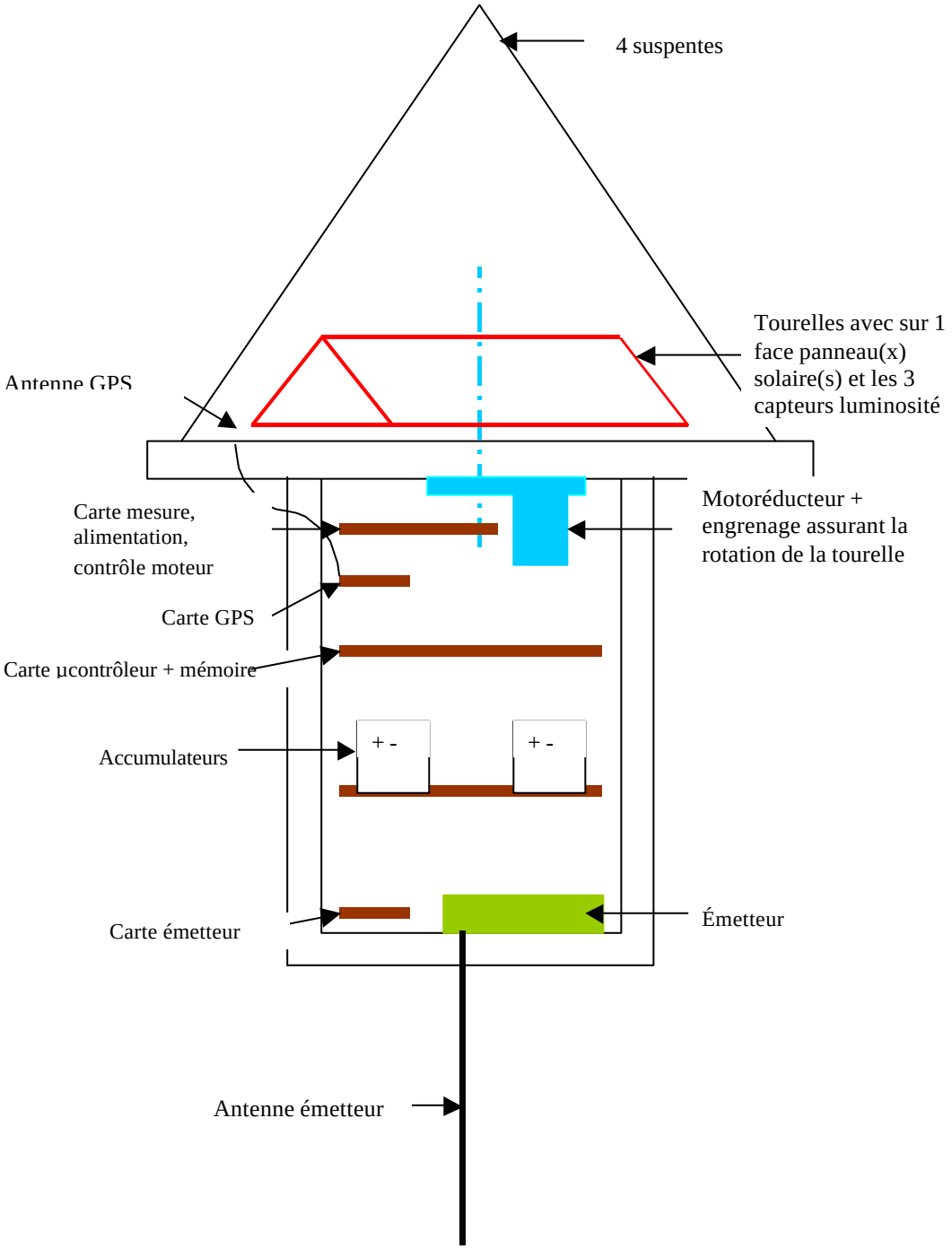
✓ Deux personnes consacrées à la réalisation de la carte microcontrôleur et sa programmation :
Ovidiu Dobre.
Olivier Bompis.

Cette répartition correspond aussi aux domaines de formation de nos études.

Mécanique

Plan général

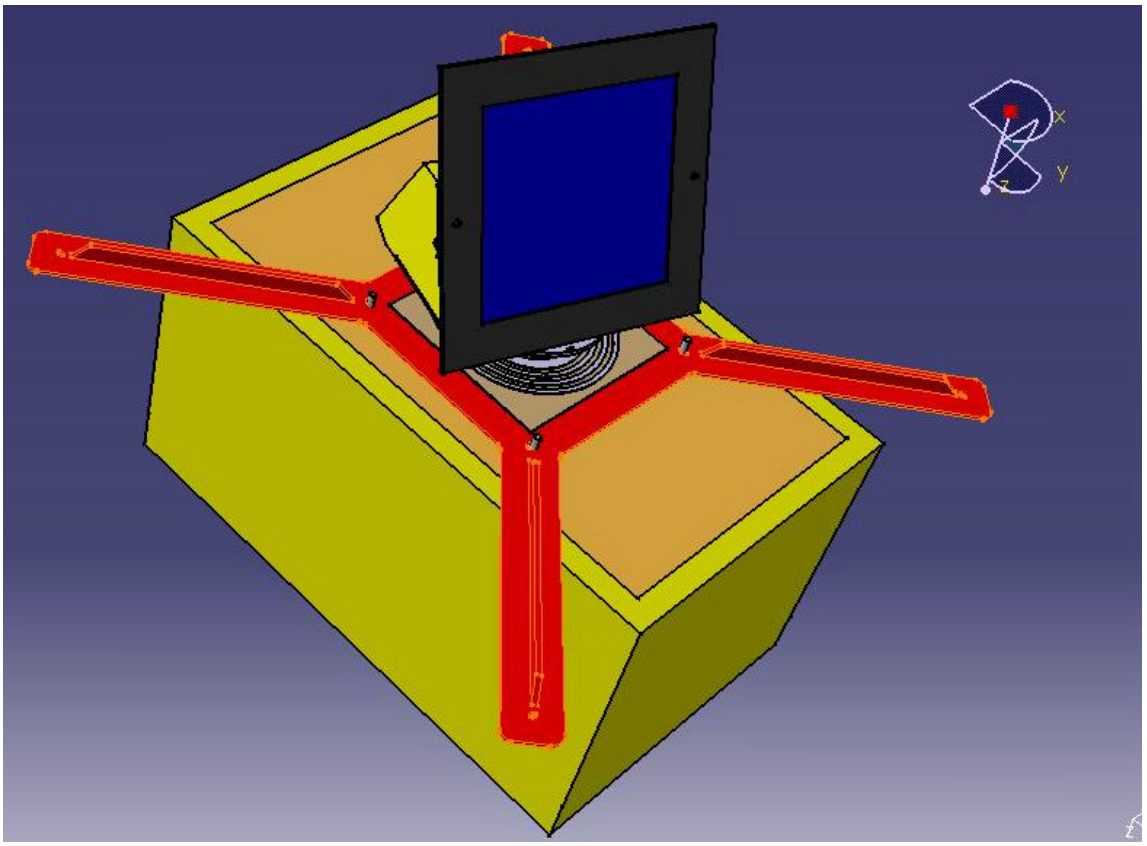
Schéma de la nacelle

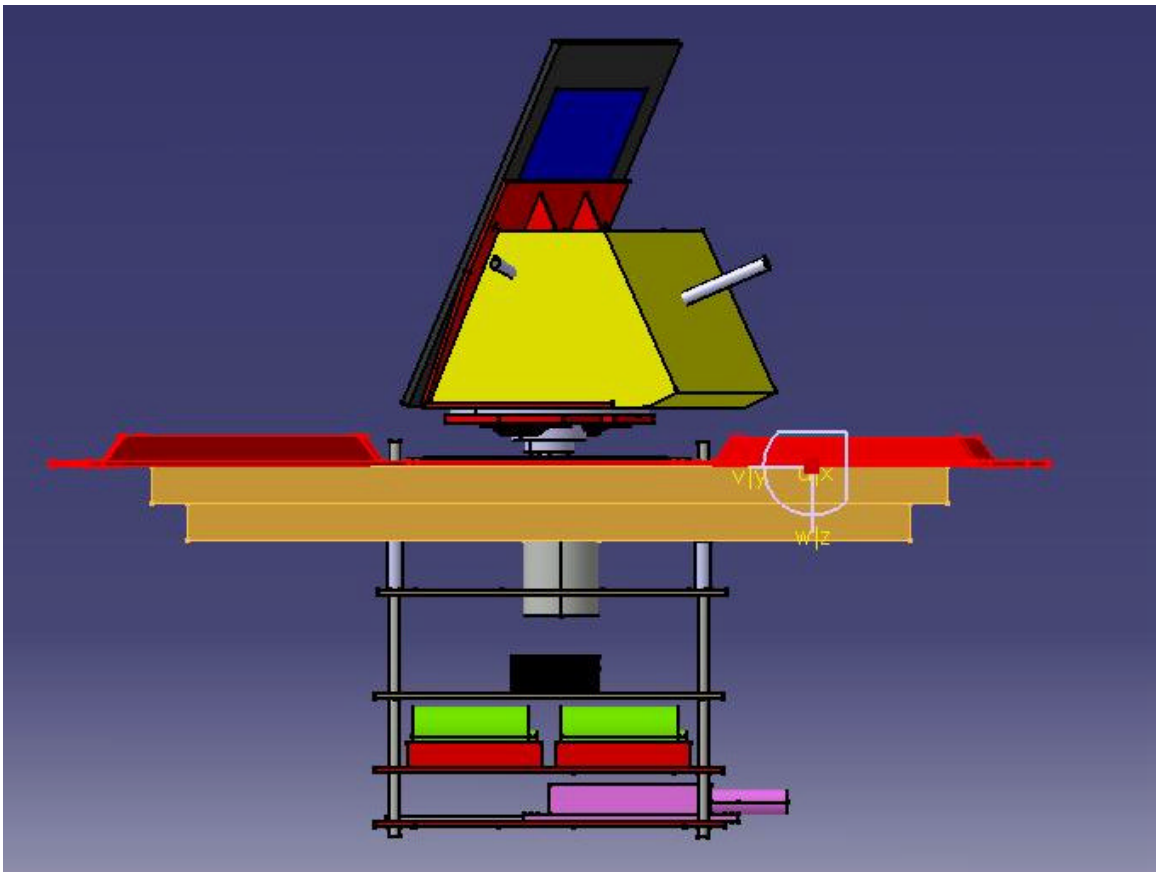


Il a été prévu dans le plan général, la séparation de l'émission du reste de la nacelle en isolant le plus possible l'émetteur des autres composants de la nacelle. Le panneau solaire est placé au dessus de la nacelle pour éviter les zones d'ombre.

Des collecteurs tournants permettent de transmettre les informations, (cellule solaire + capteurs lumière), de la tourelle vers la nacelle.

- *Aperçu de l'ensemble tourelle et nacelle réalisé sous Catia®*





La place de la mécanique dans la nacelle est assez large puisqu'il s'agit de la structure générale de la nacelle, mais aussi de la tourelle, des collecteurs tournant ainsi que le dimensionnement et la fixation des cartes électroniques.

Structure générale

La structure de la nacelle est réalisée en polystyrène expansé. Les dimensions de la nacelle sont petites pour avoir le plus petit espace de confinement afin de garder la chaleur à l'intérieur de celle ci. Elle a été la moins profonde possible, mais suffisamment large pour permettre un accès pour fixer les quelques connecteurs. L'ensemble des cartes électroniques ont le même format et sont fixées au couvercle. Elles sont ainsi facilement retirables de la nacelle. Les cartes sont empilées les unes sur les autres et sont maintenues grâce à 4 tiges filetées disposées dans les coins. Les cartes sont séparées les unes aux autres au moyen d'entretoises.

Moto réducteur

Nous avons choisi un motoréducteur (moteur + réducteur) ayant une vitesse de rotation de 15 tr/min. C'est un moteur à courant continu 12V. Il a été principalement choisi pour des raisons de robustesse grâce à sa résistance à l'effort axial, le méplat existant à extrémité de l'arbre assure une fixation aisée de la tourelle.

Collecteurs tournants

Les collecteurs tournant transmettent 5 informations, 3 capteurs de lumière, 1 courant fourni par le panneau ainsi que la masse commune. Ceux-ci ont été réalisés en laiton, matériau bon conducteur. A partir d'une plaque de laiton nous avons réalisé des lamelles que nous avons ensuite forgées. L'élasticité naturelle de ce matériau est suffisante pour épouser les formes des pistes. En effet les lamelles forment la partie mobile du dispositif, la partie fixe a elle été réalisée grâce à une plaque d'époxy pour circuit imprimé sur laquelle nous avons dessiné des pistes concentriques. L'avantage de cette réalisation est qu'elle est très économique ; L'inconvénient réside dans les pertes par frottement importantes.

→ ***Schéma cf. annexe***

Tourelle

La tourelle supporte le panneau solaire ainsi que les capteurs de lumière. Les capteurs sont disposés et protégés des faibles températures. Nous avons fait attention de bien isoler les capteurs de lumière entre eux. L'inclinaison du panneau solaire a été calculée pour être la plus optimale par rapport au soleil du début d'après midi. Il est important de noter que les capteurs photoélectriques ont la même inclinaison. Nous avons placé devant ceux-ci des tubes en cuivre qui permettent de réduire le plus possible l'angle de visée du capteur afin que celui-ci rende compte de la luminosité dans une direction précise. Il a été notamment très difficile d'isoler de la lumière l'ensemble de la tourelle ; nous avons dû la recouvrir de feutrine noire.

→ ***Plans cf. page ci-dessus***

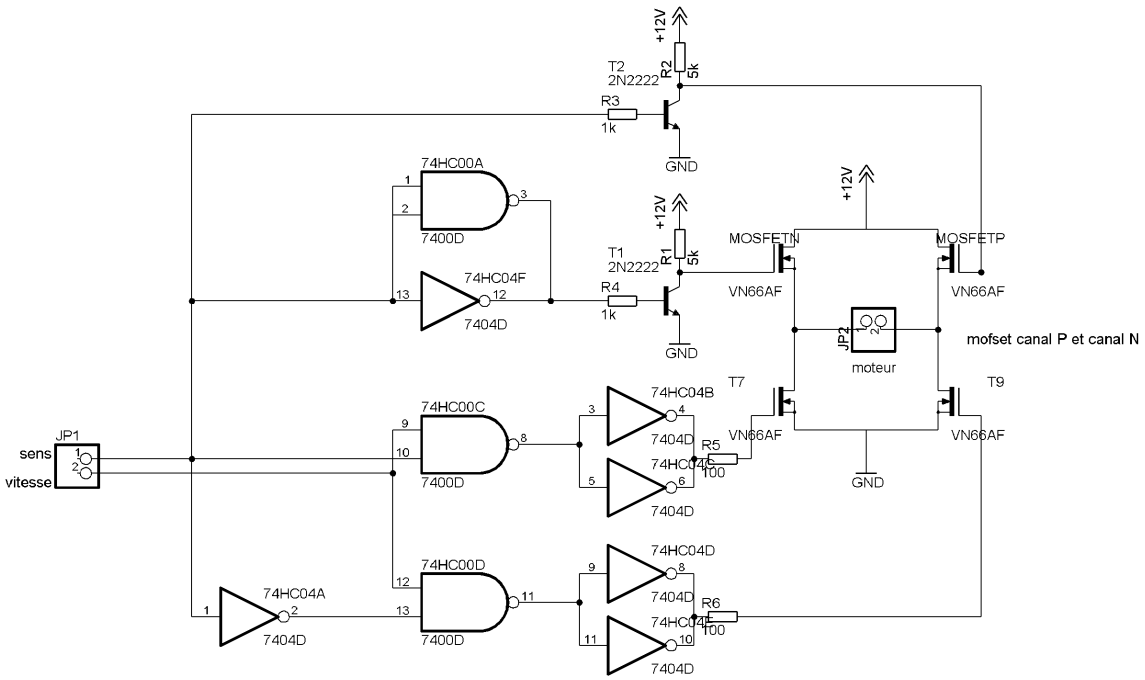
Electronique

Carte commande moteur, alimentation et mesure courant

Commande

La carte de commande moteur comporte les organes suivants : alimentation & régulation, mesure de courant, commande moteur.

La carte commande a été réalisée à l'aide de portes logiques qui pilotent des transistors *MOSFET* canal *P* et canal *N*. Les portes *NAND* et *NOR* sont doublées pour fournir plus de courant afin d'assurer une commutation franche. Le microcontrôleur envoie 2 signaux (0V ou 5V): le sens et la vitesse. Il faut les utiliser correctement pour commander les bons transistors. Le moteur est piloté par un pont en *H*, les deux canal *P* (au dessus du moteur) sont commandés uniquement par le signal "sens", ceux d'en bas par le signal "vitesse".

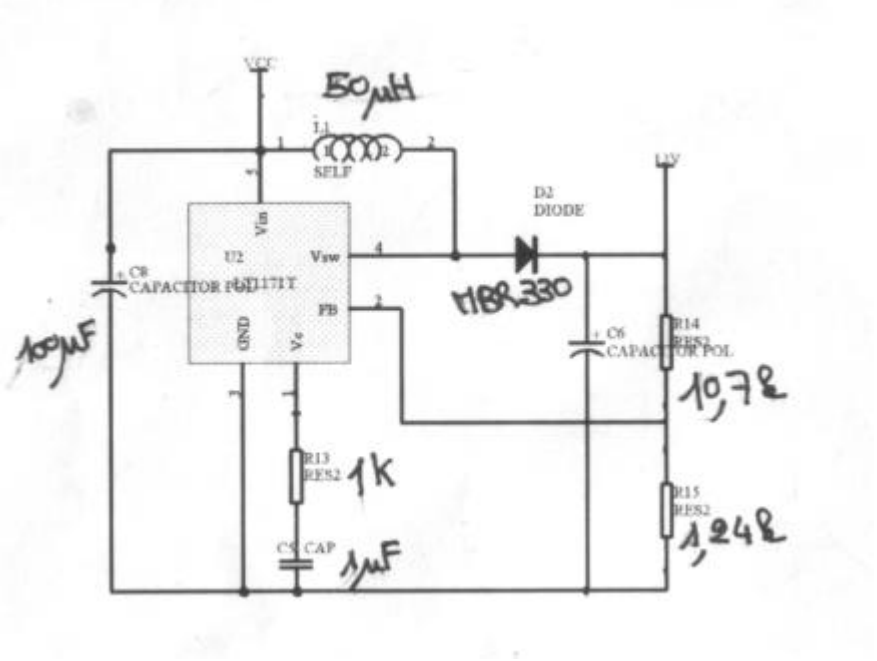


Alimentation

Dans notre nacelle, nous avons besoin d'une tension de +12V pour faire fonctionner le moteur électrique à la vitesse maximale. Cependant les accumulateurs ont une tension nominale de 6V environ. La tension a été augmentée en réalisant une alimentation à découpage basée sur le composant *LT1171*.

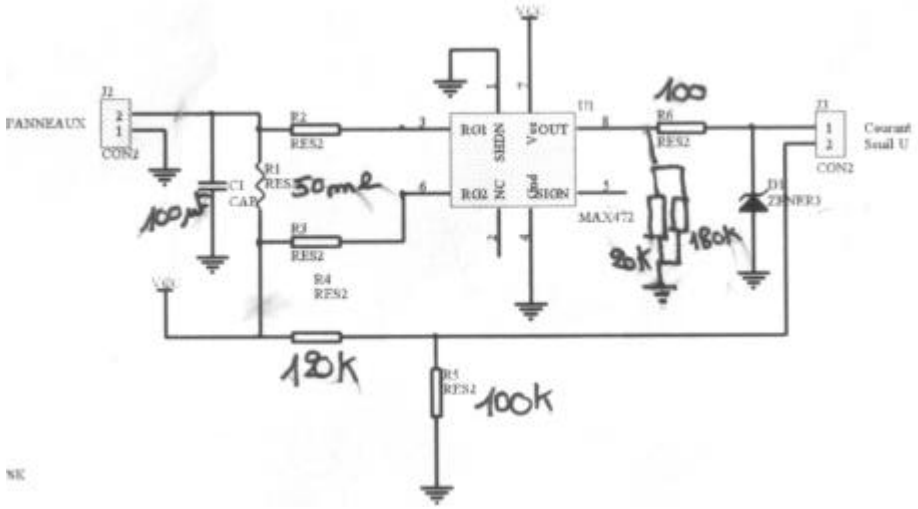
Le 5V, destiné au microcontrôleur, est généré par un régulateur série. Un modèle standard, du type 7805, requiert une différence de potentiel supérieur à +3V entre la sortie et l'entrée pour réguler correctement. La faible tension des accus donne une différence de 6V-5V = 1V.

Il a fallu utiliser un régulateur *low dropout* référence …… dont la différence de tension entrée/sortie est de seulement 500mV.



Mesure de courant

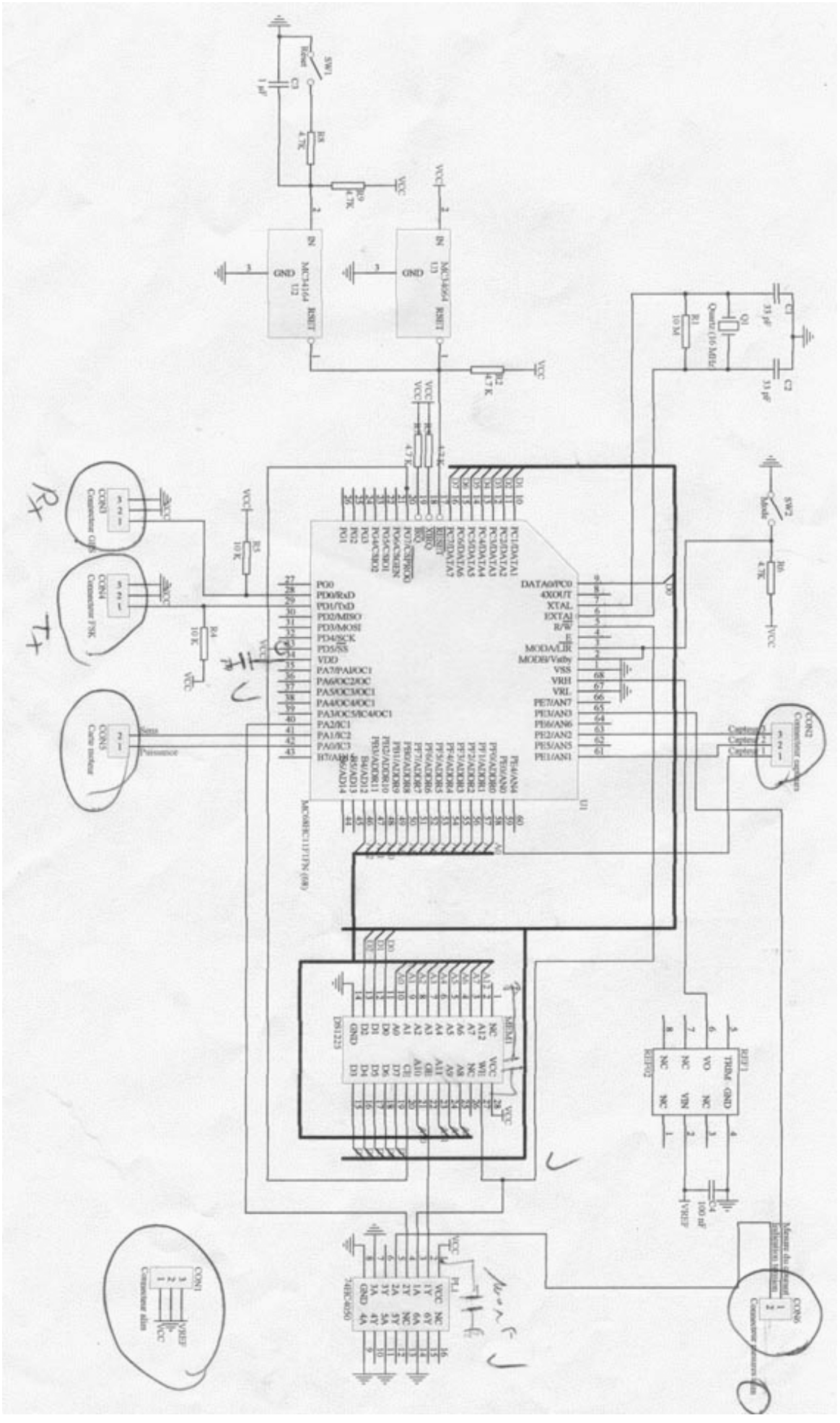
La mesure du courant, fourni par le panneau solaire, est réalisée sur cette même carte par souci de gain de place. Un composant électronique, le *MAX472* fournit une tension variant de 0 à 5V en fonction du courant traversant le Shunt de 50mÙ. Le shunt est intercalé entre les panneaux solaires et l'ensemble du système. De la sorte on mesure le courant total, c'est à dire celui de charge + celui de l'électronique. Le niveau de charge des accumulateurs est surveillé par le microcontrôleur grâce à un pont diviseur situé sur cette carte. (Voir fonctionnement ci-après).



Carte microcontrôleur

Celle-ci comporte, outre le microcontrôleur, une mémoire SRAM de 64 Ko à pile intégrée. A cela, on ajoute quelques éléments essentiels: Quartz, bouton reset et capacités de découplage. Les trois signaux de mesure de luminosité parviennent sur les entrées analogiques du calculateur ainsi que la mesure de courant fournie par les panneaux. Le niveau de tension (signal analogique) aboutit à une porte logique qui autorise la commande ou l'arrêt du moteur suivant le niveau des accus.

Si Vcc < 5,5V le seuil de la porte (2,5V) est franchi, le microcontrôleur stoppe le moteur afin d'économiser l'énergie. Le moteur redémarre lorsque la tension des accus dépasse 5,5V.

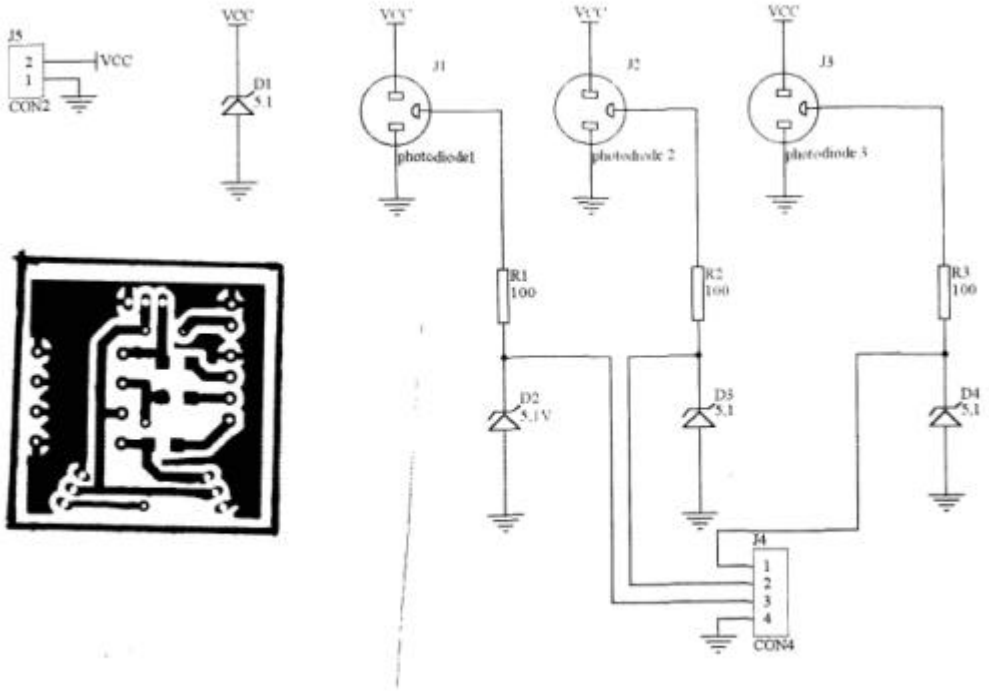


Carte FSK

Une carte interface à été prévue pour l'adaptation des signaux entre le microcontrôleur et l'émetteur *kiwi millénium*, le composant XR2206 de chez EXAR a été employé d'après la documentation fournie par Planète Sciences.

Carte mesure luminosité

Contrairement aux autres cartes, celle-ci a été placée dans la tourelle. Elle comporte les trois photodiodes *TSL 252* orientées à 120° les unes des autres. Ces photodiodes sont alimentées directement par le panneau solaire. Un diode *Zener* limite la tension a 5,1V.
Nous protégeons aussi les entrées du microcontrôleur en plaçant une *Zener* à chaque sortie des capteurs. La consommation en courant de cette carte est très faible et négligeable vis a vis du courant fourni par le panneau solaire. La mesure de courant faite dans la nacelle n'est donc que très légèrement faussée.

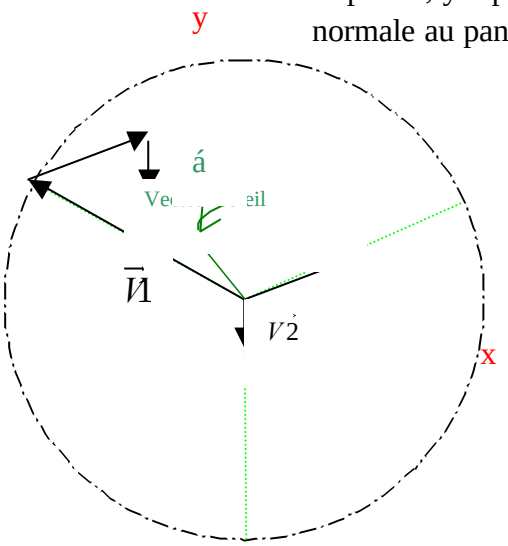


Informatique

Programmation

La programmation du 68HC11 a été réalisée en langage C. Le programme est ensuite compilé en instruction machine pour le microcontrôleur. Le programme est décomposé en plusieurs séquences telles que l'acquisition des mesures des capteurs lumières, ou la gestion de l'asservissement.

Les données des capteurs doivent être traitées pour donner l'information de l'angle que forme le panneau solaire par rapport au soleil. Un petit schéma explique facilement la méthode utilisée.



Repère x, y repère local de la tourelle, la normale au panneau est le vecteur {0,1}

Le vecteur soleil correspond à la direction du soleil, c'est à dire la somme vectorielle des vecteurs $\vec{V1},\vec{V2},\vec{V3}$, alors que $\left|\vec{V1}\right|,\left|\vec{V2}\right|,\left|\vec{V3}\right|$ sont les mesures envoyées par les capteurs de lumière. L'angle que fait la normale au panneau solaire et le soleil, (á), est l'angle qu'il faut minimiser. Il y a donc une partie du programme qui calcul cet angle alpha et qui donne un signe si alpha est positif ou négatif *cf. fonction Calculalpha()*. Dans un premier temps le programme devait envoyer une commande de vitesse au moteur qui était proportionnelle à l'angle á. Or on s'est rendu compte et malheureusement trop tard que le 68HC11 était incapable de fournir un signal de fréquence suffisamment élevée pour générer la PWM qui devait commander le moteur. Nous avons trouvé une solution de rechange qui consiste à commander le moteur en « tout ou rien » : Soit il fonctionnait à plein régime, soit il ne fonctionnait pas. Il a fallut définir un angle á minimum pour commander la rotation de la tourelle car on s'est aperçu que lorsque l'angle était trop faible le moteur oscillait. Ceci induisait une surconsommation due aux pointes de courant élevées lors du démarrage. Nous avons donc choisit de ne pas faire tourner la nacelle en dessous d'un certain angle alpha sachant que les pertes sur le panneau solaire ne sont pas très élevées. Le programme réalise aussi la formation de trames de données qu'il envoie au modulateur FSK. On remarquera que l'asservissement et l'émission de trames sont deux opérations totalement séparées. Le calculateur émet seulement quelques trames par rapport à tous les calculs réalisés.

- Description du programme implanté (cf. code source en annexe) :

Notre programme s'appuie sur deux fichiers. Un fichier d'interface, ***lux.h***, qui contient les adresses des registres que nous manipulerons, et un fichier, ***lux.c***, qui contient le programme proprement dit.

Par la suite nous ferons des commentaires que sur le fichier .C contenant le programme implanté.

Le point d'entrée du programme est représenté par la fonction ***main()*** qui va appeler successivement les procédures ***InitProg()***, et ***Asserv()***. La procédure ***InitProg()*** est chargée d'initialiser le microcontrôleur en vue des traitements futurs à réaliser. Citons par exemple l'initialisation du *Convertisseur Analogique Numérique*, la définition du «*double buffer* » permettant l'envoi de trames, et la constitution de la première trame qui sera envoyée en guise d'initialisation. Pour plus d'informations sur la procédure d'initialisation, on pourra se référer au code source du programme entièrement commenté.

La procédure ***Asserv()*** va contenir la boucle infinie, typique de la programmation microprocesseur embarqué, qui régira les cadences de traitements. Elle appellera la procédure ***AcqTrame()*** pour la conversion numérique des valeurs des capteurs en entrée. Après avoir convertit ces valeurs, sous certaines conditions (on évite la valeur 255 réservée au fanion de trame), la fonction ***CalculAlpha()*** permettra de déterminer l'angle de correction à fournir pour orienter la tourelle correctement par rapport au soleil. La valeur de cet angle permet à la procédure ***RapportPWM()*** de déterminer si le moteur doit tourner (α > **MARGE_ANGLE**). Une deuxième procédure, ***PWM()***, donnera le sens de rotation, (0 ou 1 logique), pour générer le signal « sens » de contrôle moteur.

Après avoir donné un bref aperçu des fonctions intervenant dans l'opération d'asservissement, on peut expliquer la procédure ***LanceEmission()*** qui effectue la gestion du double buffer (tableau 2x7 cases d'entiers) pour l'envoi sur le port série du microcontrôleur. Grâce à ce double buffer, on peut constituer une trame de 7 octets avec les mesures en cours pendant que le deuxième buffer est « finalisé » par l'ajout de l'octet de détection d'erreurs, (cf. procédure ***Checksum()***), en position finale. On peut donc envoyer cette deuxième trame complète tout en continuant d'enregistrer les données courantes. (***Rem*** : A noter que sur la fusée ELA, il y avait l'utilisation d'un triple buffer pour la gestion de l'émission et de l'enregistrement des trames en mémoire annexe). Enfin, la procédure ***HandlerSCI()*** gère l'interruption système apparaissant lors du vidage du registre de la liaison série ou lors de son remplissage par des données en entrées. Ici, nous initialisons le registre, et nous écrivons octet par octet les informations contenues dans notre buffer d'envoi jusqu'à émission de la trame complète. A noter la déclaration particulière de cet *handler d'interruption* dans le fichier d'interface ***« lux.h »*** et la clause particulière qui apparaît dans *le bootstrap* pour chaque interruption gérée.

Pour plus d'informations sur le programme utilisé, on pourra se référer sur le code source commenté. Pour toutes questions : Ecrire à olivier.bompis@insa-lyon.fr .

Télémesure

Carte FSK

La carte dite FSK *(Frequence Shift Keying)* a pour rôle de transformer le signal numérique (0V ou 5V) provenant du microcontrôleur en un signal modulé à l'émetteur. Les caractéristiques de la FSK sont la vitesse de transmission des information, la porteuse, et les fréquences f1 et f0 qui correspondent pour des niveaux logiques 1 et 0. La fonction de modulation est réalisée grâce à une composant très connu, le *XR2206*, qu'on utilise pour générer un signal sinusoïdal. Sur cette carte on peut y régler l'amplitude du signal, la fréquence de la sinusoïde dite porteuse ainsi que les fréquences f1 et f0 grâce à des potentiomètres.

Protocole de transmission

Le protocole de transmission utilisé était cadencé à la vitesse de transmission de 1200 bauds (symboles/seconde = bit/seconde pour un codage binaire). Pour la réception des trames, il est nécessaire de reconnaître les octets émis. Ceux-ci sont codés sur 8 bits, (0-256 $=2^8$), sans bit de parité. Les fréquences sont celles prévues pour le kiwi millénium. La trame (motif d'envoi des données) respecte un certain format nécessaire à la bonne compréhension et lecture des données.
Voici le format :

FF	Capteur 1	Capteur 2	Capteur 3	Mesure courant	Seuil tension	Checksum
1 octet	1 octet [0-254]	1 octet [0-254]	1 octet [0-254]	1 octet [0-254]	1 octet [0-1]	1 octet [0-254]
Fanion de début de trame	Mesure du capteur luminosité 1	Mesure du capteur luminosité 2	Mesure du capteur luminosité 3	Mesure du courant fourni par le panneau	Indique si la nacelle est en vie	Octet de vérification Somme des octets précédents modulos 255

Exploitation et conclusion des résultats

Le lâcher du ballon stratosphérique a été effectué le vendredi 1er Août vers 17h. Cependant l'enveloppe du ballon n'étant pas suffisamment gonflée celui ci se posa près de 200m de son point de départ, la nacelle s'accrochant à un arbre. On réussi tout de même à le récupérer pour lui remettre un peu d'hélium. C'est assez exceptionnel et il y a de nombreuses photos qui attestent la véracité de nos dires. Le ballon a ensuite été regonflé et lâché une deuxième fois. Le premier décollage nous a permis de vérifier que le ballon fonctionnait correctement et les résultats fournis par ce premier vol sont exactement ceux escomptés (cf. fichier Excel « premier lâché » en annexe). On arrivait à suivre correctement le soleil, du sol on voyait pleinement la nacelle en action. Lors du deuxième vol, qui dura environ 3 heures, les résultats sont beaucoup moins concluants (cf. fichier Excel « deuxième lâché » en annexe). La nacelle n'aura pas réussi à s'orienter dans la direction du soleil. En conséquence le courant généré par le panneau solaire est beaucoup moins important que prévu

Nous allons d'abord essayer d'interpréter les résultats à partir des informations des trames de télémesure. Nous reconstituons tout d'abord les trames correctement en supprimant les trames incomplètes mais je garde cependant les trames dites erronées, celles dont le checksum ne correspond pas avec les informations fournies. Je reconstitue ensuite l'angle entre le panneau solaire et le soleil, je trace ensuite l'angle et l'intensité du courant fourni par le panneau au cours du 1er vol (cf. fichier Excel « premier lâché » en annexe). A partir du graphique on détermine le lâcher et la fin de vol. On reconstitue un bilan énergétique en connaissant la consommation de la nacelle. On sait que lorsque l'angle soleil/nacelle est supérieur à 20° le microcontrôleur fera tourner la tourelle, le moteur fonctionnera et la consommation totale de la nacelle sera de **600mA**. La consommation lorsque le moteur ne fonctionne pas est de **280mA**. On calcule donc le bilan énergétique (énergie fournie par le panneau solaire moins consommation de la nacelle lors de chaque trame durant toute la phase de vol). Il y a 4181 trames de prise en compte j'obtient donc statistiquement le bilan énergétique moyen sur l'ensemble des trames qui est de **-189 mA**.

Le deuxième vol n'a pas pu répondu à nos attentes, on dirait presque que le panneau solaire s'est orienté dans la direction inverse du soleil. Enfin l'énergie solaire fournie est très en dessous de ce qui était prévisible, un exemple classique parmi les 80000 trames reçues est illustré (cf. fichier Excel « deuxième lâché » en annexe). On remarquera que le panneau solaire est plus souvent proche des 180° que des 0°. La mesure de courant est très faible. La nacelle fonctionna correctement jusqu'à environ 10000 trames puis le ballon perdit ensuite en capacité et la nacelle coupa l'alimentation du moteur de plus en plus souvent arrivant jusqu'à l'extinction totale de celui-ci. Lorsque l'on réalise un bilan énergétique, le ballon totalement éteint, celui-ci est sur une moyenne de 10000 trames de **–235 mA**. Quand l'asservissement n'a pas fonctionné celui-ci était de **-549mA**.

Conclusion des résultats

Vraisemblablement l'asservissement n'a pas été à la hauteur de nos espérances, et ce en divers point. Mais le premier lâcher nous aura permis de vérifier qu'asservir un panneau solaire permet de gagner jusqu'à 20% d'autonomie. Par contre lorsque l'asservissement ne remplit pas sa tâche correctement la perte est de 130% par rapport à système non asservi (sans panneau solaire…).

Améliorations envisageables

Ce qui a été bien :

Nous avons opté pour une disposition de la tourelle sur le dessus de la nacelle et de maintenir la nacelle grâce à quatre suspentes. Ce choix fut judicieux et la tourelle ne fut pas à l'ombre de la chaîne de vol car celle ci étant bien plus haute. L'intégration, plus facile qu'une fusée expérimentale, fit preuve d'ingéniosité en terme d'accès rapide aux cartes inférieures. En effet les cartes sont superposées les unes aux autres et facilement retirables grâce à des entretoises et sont reliées au couvercle. Cela nous permet d'accéder plus facilement à chaque carte.

Améliorations possibles :

- Un énorme point critique de notre nacelle est la vitesse de rotation trop faible de la tourelle. Les choix techniques du moteur étaient justifiés du fait de notre conception: Il était difficile de trouver un moteur avec un fort couple et une vitesse de rotation élevée. Il manquait cependant des données précises sur la vitesse de rotation supposée de la nacelle, celle-ci pouvant varier suivant les conditions climatiques, en particulier avec la présence de vents.

- On peut aussi améliorer quelque peu l'intégration, en particulier la connectique entre les différentes cartes électroniques: Certains connecteurs se trouvaient au milieu des cartes, d'autres occupaient un côté sur les quatre disponibles.

- Il manque une hystérésis sur le seuil de tension destiné à la coupure moteur. Conséquence: Passé le seuil inférieur de 5,5V le moteur est coupé, la consommation étant plus faible la tensions des accus remonte et repasse le seuil ce qui réactive le moteur, etc.… On se retrouve avec un système oscillant à très basse fréquence.

Deux solutions sont applicables:

- Utiliser un *AOP* à la place de la porte logique câblé en comparateur à fenêtre, seuil bas = 5,5V et seuil haut = 6V
- Conserver la porte mais introduire un retard de quelques dizaines de seconde dans le programme afin de laisser le temps aux accumulateurs d'atteindre une tension suffisante avant de redémarrer le moteur.

➢ Au niveau de la conception de la carte processeur, le reset automatique à la mise sous tension implanté n'a pas fonctionné: L'utilisation du *MC 34164* est à revoir.

➢ En terme d'améliorations sur le programme embarqué, nous avons oublié le précieux « *watchdog* » du microcontrôleur, destiné à redémarrer le programme en cas de plantage. Il aurait souhaitable de mieux définir nos besoins en terme de vitesse de traitements et de choisir le microcontrôleur 68hc11 ad hoc pour réaliser un signal PWM à la fréquence requise (*Rem* : Certains types de 68hc11 réalisent la PWM, donc avant de partir tête baissée…). Enfin, une optimisation sur certaines procédures (*CalculAlpha()* ou sur la fonction *Arctan()*) aurait souhaitable pour accélérer la vitesse de traitement qui est critique sur un système asservi.

Le mot de la fin…

Après de nombreux déboires, (lâcher prévu en mai puis repoussé par 2 fois) la nacelle Giro aura finalement volé. Elle a été lâchée en fonctionnant correctement, il ne s'agit pas d'un projet fini à la hâte. Les tests réalisés au sol ont montré que ce projet avait eu la capacité de mener à bien les objectifs principaux fixés. Malgré les conditions d'ensoleillement idéal le vol nous a montré que nous avions émis des hypothèses trop restreintes ne nous amenant pas aux résultats escomptés. La viabilité économique d'un tel système pour un ballon sonde est quasiment nul mais au-delà de cet aspect, il s'agissait aussi d'asservir un système (panneau solaire) dans un environnement totalement aléatoire tant sur le mouvement de la nacelle que sur le ballon engendrée par la rotation de la tourelle. Nous venons néanmoins de poser une première «pierre» pour ceux qui désireront recommencer plus tard une expérience du même genre.

Annexes

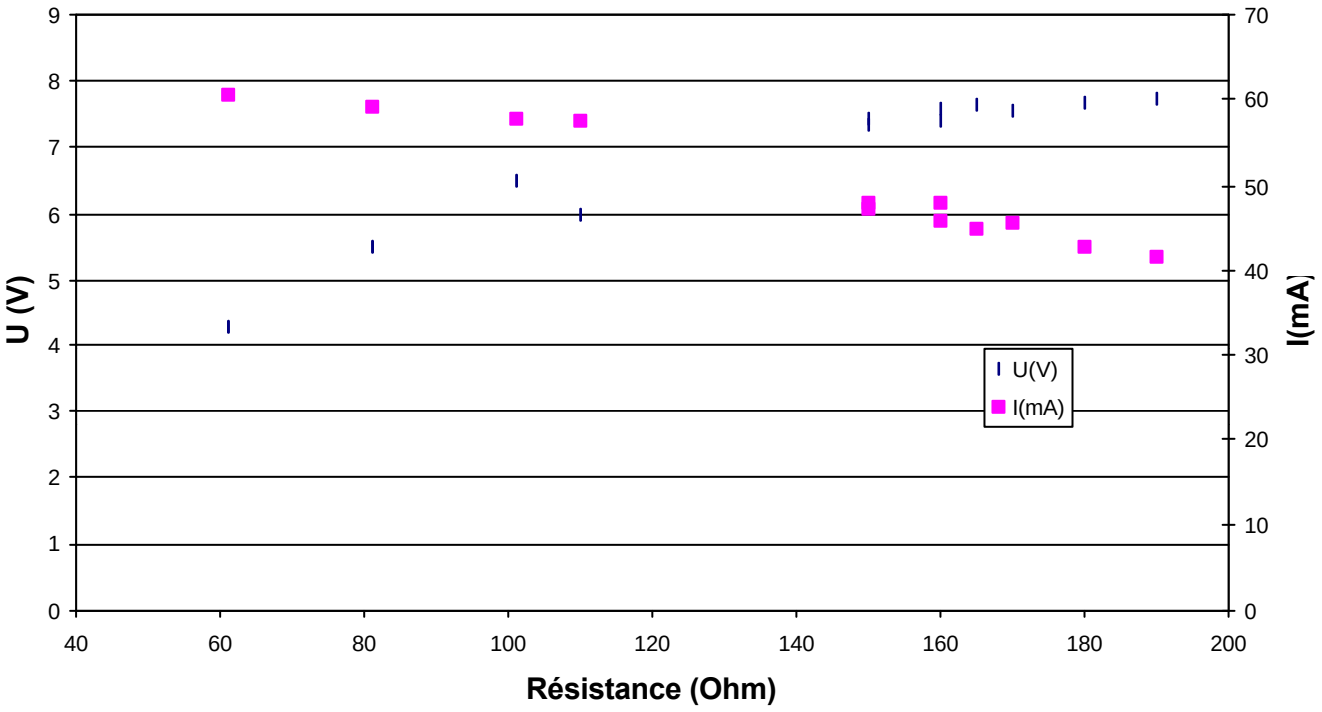
Annexe Etalonnage du panneau solaire

Il s'agissait d'abord de déterminer le point de fonctionnement du panneau solaire permettant d'avoir une tension de 7,5V à ses bornes, Pour cela nous avons procédé à la première expérience qui consistait à faire varier la charge aux bornes du panneau solaire. Nous avons ainsi mesuré pour différentes valeurs de résistance, l'intensité et la tension fournies par le générateur.

Diamètre du détecteur	0,82	cm
Surface du détecteur	0,528	cm²

Luminosité 1000W/m²

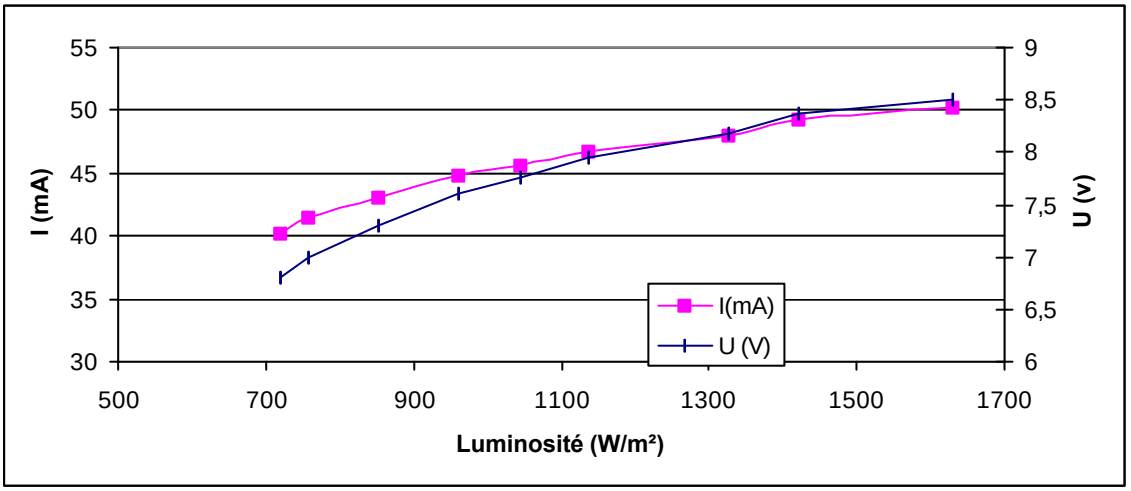
R (ohms)	U (V)	I (mA)	P (W)
61	4,3	60,5	0,260
81	5,49	59,2	0,325
101	6,52	57,6	0,376
110	6	57,5	0,345
150	7,43	48	0,357
150	7,32	47,3	0,346
160	7,58	45,9	0,348
160	7,4	47,9	0,354
165	7,63	45	0,343
170	7,53	45,7	0,344
180	7,67	42,9	0,329
190	7,72	41,7	0,322

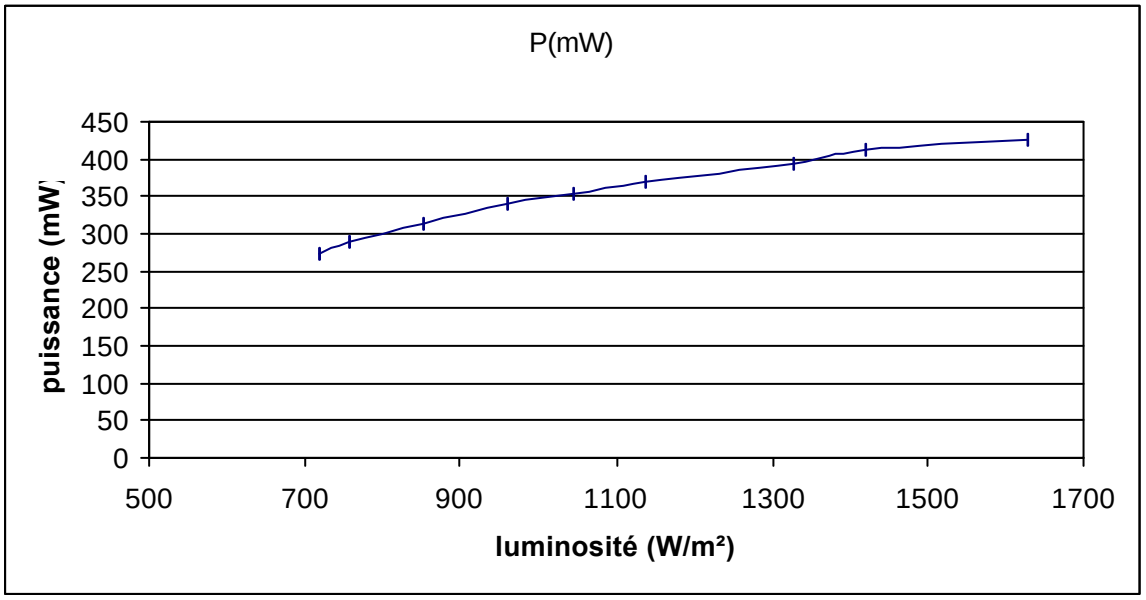


R=165ohm

Eclairement (mW)	luminosité (W/m²)	U (V)	I (mA)	P (mW)
38	720	6,8	40,2	273
40	757	7	41,4	290
45	852	7,3	43	314
50,7	960	7,6	44,8	340
55,2	1045	7,75	45,6	353
60	1136	7,94	46,7	371
70	1326	8,18	48	393
75	1420	8,36	49,2	411
86	1628	8,5	50,2	427

Cette première expérience nous permet de montrer qu'il est nécessaire d'avoir une charge de 165 ohms aux bornes du panneau solaire afin d'obtenir la tension désirée.
Lors de notre deuxième expérience nous avons essayé de caractériser l'intensité et la tension fournie par le panneau solaire suivant différents éclairements.





Annexe programme en C

```
#include "lux.h"

/******************** Programme Asservissement Ballon LUX ************************/
/********************** Bompis Olivier - Dobre Ovidiu ********************************Revu par Yann Gouy****/

//----------------Prototypes de fonctions locales---------
static unsigned char Arctan(int X);
static void Checksum(void);

//-------------- Variables Globales ----------------------

volatile unsigned char alpha;
volatile unsigned char sens;

static const unsigned char bufApres[2]={1,0};
const unsigned short tan[23]={
            0,  1,  2,  3,  3,  4,  5,  6,  7,  8,
            9, 10, 12, 13, 15, 18, 21, 26, 33, 43,
     57, 95, 286};

static const unsigned char idBuf[2] = {0, 7};                   // A CHANGER SI TAILLE TRAME CHANGE
static volatile unsigned char buf[2 * FIN_BUFFER];              // Buffers d'émission

static volatile unsigned char bufrec;                           // Numéro du buffer en cours de réception

static unsigned char bufenv;                                    // Numéro du buffer en cours d'envoi
static char nbOctetEnv;                                         // Position dans le buffer d'envoi
static char trameEnvoye;                                        // Indique si la trame a été envoyé

static unsigned volatile short adresse1 = 0;                    // Variables pour valeurs capteurs quantifiées.
static unsigned volatile short adresse2 = 0;
static unsigned volatile short adresse3 = 0;

//------------ Debut du module lux.c ----------------------

//--------- Fonctions pour la mise en place de l'asservissement ------------
```

void InitProg (void)

```
                                                                // Initialisations des regs système, du CAN, de la SCI,
                                                                // des deux buffers de transmission, des IT s.
{

    /* ---- initialisation des registres système ---- */

    // port série : SCI
    *baud = 0x34;                                               // Config émission 1200 bauds 0x34
    *sccr1= 0x00;
    *sccr2= 0x0C;                                               // l'IT sur la SCI n'est pas encore activée

    // port A
    *DDRA = 0x03;                                               // Config du port A numérique
                                                                // broches 1 & 2 en sortie et 3 en entrée
    *portA = 0x00;                                              // Aucun signal en sortie

    // CAN
    *option |= 0x90;                                            // Config du CAN pour le clock sur E system


    /* ---- vecteurs d'interruption ---- */

    // !!!! ecrire dans EEPROM dans la table d'IT
    // pour SCI : 0xFFD6 <- 0x200 et pour TOC2 0xFFE6 <- 0x204
    *(char *)0x200=0x7e;                                                // opcode JMP
    *(void **)0x201=(void *)handlerSCI;                                 // IT sur SCI
    *(char *)0x203=0x01;                                                // opcode JMP

    *(char *)0x204=0x7e;                                                // opcode JMP
    *(void **)0x205=(void *)handlerPWM;                                 // IT sur TOC2
    *(char *)0x207=0x01;                                                // opcode JMP
```

```
    /* ---- RAZ des premiers octets des deux buffers d'envoi ---- */

    buf[0]=0xFF;        // Fanion de debut de trame
    buf[7]=0xFF;        // Fanion de debut de trame

    /* ---- initialisation de la premiere trame (avant language) ---- */

    buf[POS_CAPTEURS]='G';                  // Les champs capteurs C1 C2 C3 avec GIR pour la première trame.
    buf[POS_CAPTEURS + 1]='I';
    buf[POS_CAPTEURS + 2]='R';

    bufenv = 0;                             // le premier buffer sera envoyé en début
    bufrec = 1;                             // le deuxième buffer servira pour le remplissage

    trameEnvoye = 0;
    nbOctetEnv = 1;

    Checksum();                             // calcul du checksum pour la première trame


    /* ---- activation des vecteurs d'interruptions (SCI & Timers) ---- */

    *sccr2 = 0x8C;                          // on active l'IT sur SCI: Envoi de la premiere trame

    // *PACTL |= 0x10;
    // *TMSK1 |= 0x40;                      // activation IT OutputCompare2 – REM: L'utilisation du timer pour générer une
                                            // PWM n'a finalement pas été retenue pour l'asservissement de la tourelle. Je laisse
                                            // les inits des regs du Timer à titre de documentation….
    // *TMSK2 |= 0x00;
}
```

static void Checksum ()

```
                                            // Calcul octet de parité pour vérifier
                                            // l'intégrité de la trame au sol
                                            // il sera stocké dans le <bufenv>
{
    unsigned char check = 0;
    unsigned char debutBuf = idBuf[bufenv];
    unsigned char i;

    for (i=1; i<FIN_BUFFER-1; i++)
            check = check + buf[debutBuf + i];

    if (check==255)
                    check=0;

    buf[debutBuf + POS_CHECKSUM] = check;
}
```

static void AcqTrame ()

```
// Acquisition capteurs, courant, portA
// les données seront stockées dans <bufrec>
{
    unsigned short add1 = 0;
    unsigned short add2 = 0;
    unsigned short add3 = 0;
    unsigned short add4 = 0;
    unsigned char debutBuf = idBuf[bufrec];

    *adctl = 0x10;                          // Lancement conversion AN

    while ( (*adctl & 128) == 0) {}         // Attente conversion

        add1 = (adresse1 - *adr1);
        add2 = (adresse2 - *adr2);
            add3 = (adresse3 - *adr3);
            add4 = *adr4;

            if ( add1 > 25 ) {              // Moyennage des valeurs quantifiees
                adresse1 = *adr1;
            }
            if ( add2 > 25 ) {
                adresse2 = *adr2;
            }
```

```
            if ( add3 > 25 ) {
               adresse3 = *adr3;
            }

            if ( add4 == 255 ) {                    // Enleve le caractere FF des champs
               add4 = 254;                          // des valeurs capteurs dans la trame
            }
            if ( adresse1 == 255) {
               adresse1 = 254;
            }
            if ( adresse2 == 255) {
               adresse2 = 254;
            }
            if ( adresse3 == 255) {
               adresse3 = 254;
            }

            buf[debutBuf] = 0xFF;                           // Fanion ouverture.

            buf[debutBuf + POS_CAPTEURS] = adresse1;        // Valeurs capteurs lumiere.
            buf[debutBuf + POS_CAPTEURS + 1] = adresse2;
            buf[debutBuf + POS_CAPTEURS + 2] = adresse3;
            buf[debutBuf + POS_COURANT] = add4;             // Valeur courant.

            if ( (*portA & 0x04) == 0x04 ) {                // Indication seuil tension.
                        buf[debutBuf + POS_ALIM] = 1;
            }
            else {
                        buf[debutBuf + POS_ALIM] = 0;
            }

}
```

static void CalculAlpha (unsigned char C1, unsigned char C2, unsigned char C3)

```
// Calcul de l'angle et du sens de correction.
// le résultat sera mis dans <angle>
{
   int RX;
   int RY;

   if ( ( (C3 - C1) <= 3 ) && ( (C3 - C2) <= 3 ) && ( (C2 - C1) <= 3 ) )
   {
            RX = 0;
            RY = 0;
   }
   else {

   RX = RACINE_3_2 * (C3 - C1);                 // * 100    Optimisation de calcul du microcontrôleur.
   RY = (10 * C2 - 5 * ((C1 + C3)));            // * 10
   }

   if ( (RX >= 0) && (RY > 0) )                 // 1er cadran : 0 - PI/2
   {
            sens  = 0;
            alpha = Arctan(RX / RY);
            return;
   }

   if ( (RX <= 0) && (RY > 0) )                 // 2eme cadran : PI/2 - PI
   {
            sens  = 1;
            alpha = Arctan((-RX) / RY);
            return;
   }

   if ( (RX <= 0) && (RY < 0) )                 // 3eme cadran : PI - 3*PI/2
   {
            sens  = 1;
            alpha = 180 - Arctan(RX / RY);
            return;
   }

   if ( (RX >= 0) && (RY < 0) )                 // 4eme cadran : 3*PI/2 - 2*PI
   {
            sens  = 0;
            alpha = 180 - Arctan(RX / (-RY));
            return;    }
```

```
   if ( (RX > 0) && (RY == 0) )                    // AXE OX+
   {
            sens  = 0;
            alpha = 90;
            return;
   }

   if ( (RX < 0) && (RY == 0) )                    // AXE OX-
   {
            sens  = 1;
            alpha = 90;
            return;
   }

   if ( (RX == 0) && (RY == 0) )                   // Valeurs nulles
   {
            sens = 0;                              // Aucune rotation
            alpha = 180;
            return;
   }
}
```

static void RapportPWM (void) // Commande de la puissance (ou pas) du moteur si correction nécessaire.
// C'est-à-dire angle de correction >= MARGE_ANGLE.

```
{

  if ( (*portA & 0x04) == 0x04 ){                  // gestion du seuil de tension
            *portA &= 0xFE;                        // arrêt  du moteur si seuil bas
  } else {

  if ( (180 - alpha) <= MARGE_ANGLE )
   {
            *portA &= 0xFE;                        // éteindre la puissance
   }
   else
   {
            *portA |= 0x01;                        // fournir de la puissance
   }
  }
}
```

static void PWM (void) // Commande du sens de rotation du moteur

```
{
   if (sens == 0)                                  // sens rotation
   {
            *portA &= 0xFD;                        // sens à zero
   }
   else
   {
            *portA |= 0X02;                        // sens à un
   }

}
```

static void LanceEmission(void)
// Lance l'emission de la trame pour la FSK

```
{
   bufenv = bufrec;                                // le buffer rempli sera reservé à l'emission
   bufrec = bufApres[bufrec];                      // change de buffer de reception

   Checksum();                                     // calcul le checksum pour la trame

   nbOctetEnv = 1;
   trameEnvoye = 0;

   *scdr = buf[idBuf[bufenv]];                     // envoie du premier caractère

   *sccr2 |= 0x80;                                 // reactivation de l'IT SCI

}
```

void Asserv (void){
// Gestion Asservissement

```
unsigned short compteur = 0;
unsigned short compteur1 = 0;
```

```
  while (1){                                    // boucle infinie du programme proc
            AcqTrame();

            CalculAlpha(adresse1,adresse2,adresse3);  // calcul de l'angle de correction en fction des valeurs quantifiées.

      if (trameEnvoye == 1)                     // si pas de trame en cours d'émission
            {
             LanceEmission();
            }
            if (compteur == 300)                // réglages de l'asservissement (bricolage prenant en compte le temps de réaction du
                                                // moteur).
            {
             RapportPWM();                      // action de correction
             compteur = 0;
            }
            if (compteur1 == 8)                 // bricolage évitant les oscillations du moteur
            {
             PWM();                             // détermination du sens de rotation de la tourelle.
             compteur1 = 0;
            }

  compteur1 ++;                                 // incrémentation des compteurs « bricolages » permettant la réaction du moteur.
  compteur ++;
  }
}

//------------------ Handlers d'IT -------------------------- // Gestion des Interruptions systèmes (Emission & Réception série)
```

void handlerSCI (void)

```
// Gestion IT RX & TX SCI
{
   // handler TX
   if ( *scsr & 128 )                           // IT sur SCDR empty
    {
            if ( nbOctetEnv >= FIN_BUFFER )     // si la trame a été envoyé
      {                                         // on indique la fin
               trameEnvoye = 1;
               *sccr2 &= ~0x80;                 // on désactive l'IT SCI

            }
      if (nbOctetEnv < FIN_BUFFER)
            {                                   // envoie du caractère suivant
              *scdr = buf[bufenv* FIN_BUFFER + nbOctetEnv];

       nbOctetEnv++;
            }
    }
  }
```

void handlerPWM (void)

```
                                                // Gestion d'IT pour la RTI afin de contrôler le PWN (solution non retenue
                                                // finalement mais je laisse l'handler à titre de documentation…)
{
 *TFLG1 = 0x40;                 // clear OC2F
/*
 if (enCoursPWM) {

            *TOC2 = ( *TOC2 + PERIODE_PWM - Ticks );

            *portA &= 0xFE;     // eteindre la puissance
            enCoursPWM = 0;       // Le signal PWM est a 0
 }
 else {
            Ticks = nbTicksOn;
            *TOC2 = ( *TOC2 + Ticks );

            *portA |= 0x01;       // fournir de la puissance
      enCoursPWM = 1;           // autorise une nouvelle PWM
 }*/
}


// ------------ fonctions auxiliaires ----------------
```

```
unsigned char Arctan(int X)
                                        // Calcul de l'arctan entre 0 et PI/2 avec une table de valeurs en constante.
{
  unsigned char milieu = 0;

   if (X > tan[22])                     // Parcours bête du tableau de valeurs de l'arctan jusqu'à la valeur ad'hoc.
   {        return 90;
   }

   while ( (X > tan[milieu]) && (milieu <= 22) ){
            milieu++;
   }

   return (milieu * 4);                 // On retourne la valeur multipliée par 4 car on travaille réellement entre 0° et 90°.


}

int main()
                                        // le point d'entrée du programme :
                                        // il mets en place les inits (InitProg() ), puis il lance la procédure d'asservissement
                                         Asserv()
{
   InitProg();
   Asserv();
   return 0;
}
```

Déclaration du fichier d'interface lux.h

```
//----------------Fichier d'interface Lux.h----------------------//

#ifndef lux_h
#define lux_h

#define DebugBreak __asm__("swi");

//---Definition des structures de donnees---//

extern volatile unsigned char alpha;
extern volatile unsigned char sens;

extern const unsigned short tan[23];

//------------- Definition des constantes ------------//

#define RACINE_3_2          87   //0.866     // Valeur precalculee (*100)
#define CONSTANTE_K         6    //0.0055    // Constante d'asservissement (Beta/180)(*1000)

#define MARGE_ANGLE         26               // Déviation d'angle à ignorer


//------------ Definition des registres --------------//

/* les regs. d'usage general */

#define init (volatile unsigned char *) 0x1039        // Options config sys
#define portA (volatile unsigned char *)0x1000        // le regs du port A
#define DDRA (volatile unsigned char *)0x1001         // le regs de direction du port A
```

```
/* registres du CAN */

#define adctl (volatile unsigned char *)0x1030      // ADCTL reg de control du CAN
#define option (volatile unsigned char *)0x1039     // OPTION reg de control du CAN
#define adr1 (volatile unsigned char *)0x1031       // le 1 reg des données du CAN
#define adr2 (volatile unsigned char *)0x1032       // le 2 reg des données du CAN
#define adr3 (volatile unsigned char *)0x1033       // le 3 reg des données du CAN
#define adr4 (volatile unsigned char *)0x1034       // le 4 reg des données du CAN


/* registres de la SCI */

#define scdr (volatile unsigned char *)0x102F       // SCDR reg données de la SCI
#define sccr1 (volatile unsigned char *)0x102C      // SCCR1 reg de ctrl de la SCI
#define sccr2 (volatile unsigned char *)0x102D      // SCCR2 reg de ctrl de la SCI
#define baud (volatile unsigned char *)0x102B       // BAUD reg de ctrl de la SCI
#define scsr (volatile unsigned char *)0x102E       // SCSR reg d'etat de la SCI
#define sciIT (volatile unsigned short *)0xFFD6     // @ vector d'IT de la SCI

/* registres TIMER */

volatile unsigned short* TCNT = (short*)0x100F;    /* compteur TIMER */
volatile unsigned short* TOC2 = (short*)0x1019;    /* compteur Compare2 */

#define TMSK1 (volatile unsigned char *)0x1022     /* activation ITComp */
#define TMSK2 (volatile unsigned char *)0x1024     /* activation RTI */
#define PACTL (volatile unsigned char *)0x1026     /* controle RTI */
#define TFLG1 (volatile unsigned char *)0x1023     /* activation ITCom */
#define TFLG2 (volatile unsigned char *)0x1025     /* activation RTI */


// ------------ Variables internes et alias -------------- //

#define POS_CAPTEURS    (unsigned char)1      // Debut 1er octet capteurs
#define POS_COURANT     (unsigned char)4      // Debut 1er octet Courant
#define POS_ALIM        (unsigned char)5      // Octet indication seuil Alim
#define POS_CHECKSUM    (unsigned char)6      // Octet detection erreur
#define FIN_BUFFER      (unsigned char)7      // Fanion de fin de trame

//-------- Fonctions accessibles depuis d'autres modules ----------//

extern void InitProg();
extern void Asserv();

void handlerSCI (void) __attribute__((interrupt));     // handler d'IT pour la SCI (TX & RX)

void handlerPWM (void) __attribute__((interrupt));  // handler d'IT pour le timer du PWM

#endif
```

//-----Fin de definition de lux.h-----//

-------Script DB11 pour l'envoi du programme principal-------

```
# This script loads lux.s19

include expandF1.db11          // Gestion des fonctionnalités du 68hc11F1.
#include debug.db11            // Active l'option de debugage de DB11.
#include debugee.db11

#include swi_ram_install_cerbere.db11

s *0x105d=4                    // Ecriture dans les registres système
s *0x1003=0x1F                 // CAN - SCI - Clock E.
s *0x1039=0xC0
#s *0x1030=0x20
                               // Autorise l'écriture dans l'EEPROM.
#machine yann                  // (en commentaire) On fixe les vecteurs
#s *0xffd6 = 0x200             // d'interruptions. (SCI = 0x200, TIMER
#s *0xffd8 = 0x208             // = 0x204).
#s *0xffda = 0x208
#s *0xffdc = 0x208
#s *0xffde = 0x208
#s *0xffe0 = 0x208
#s *0xffe2 = 0x208
#s *0xffe4 = 0x208
#s *0xffe6 = 0x208
#s *0xffe8 = 0x208
#s *0xffea = 0x208
#s *0xffec = 0x208
#s *0xffee = 0x208
#s *0xfff0 = 0x204
#s *0xfff2 = 0x208
#s *0xfff4 = 0x208
#s *0xfff6 = 0x208
#s *0xfff8 = 0x208
#s *0xfffa = 0x208
#s *0xfffc = 0x208
#s *0xfffe = 0x1100            // RESET ACTIF au démarrage.

# Chargement du programme dans la RAM interne
file ../lux/lux.s19

s pc=0x1100                    // On fixe le pointeur d'instructions au
                               // début du programme (= valeur du RESET).
```

-------Script DB11 pour l'envoi du bootstrap-------

```
# This script loads talkFE80.s19 and bootstrap.s19 in EEPROM
# then it's up to you to quit or verify the installation.

include debug.db11

machine Yann                   // Autorise l'écriture dans l'EEPROM.

# set the config register
set *0x103f=0xff

                               // Envoi le fichier bootstrap dans l'EEPROM.
file ../lux/bootstrap/bootstrap.s19
```

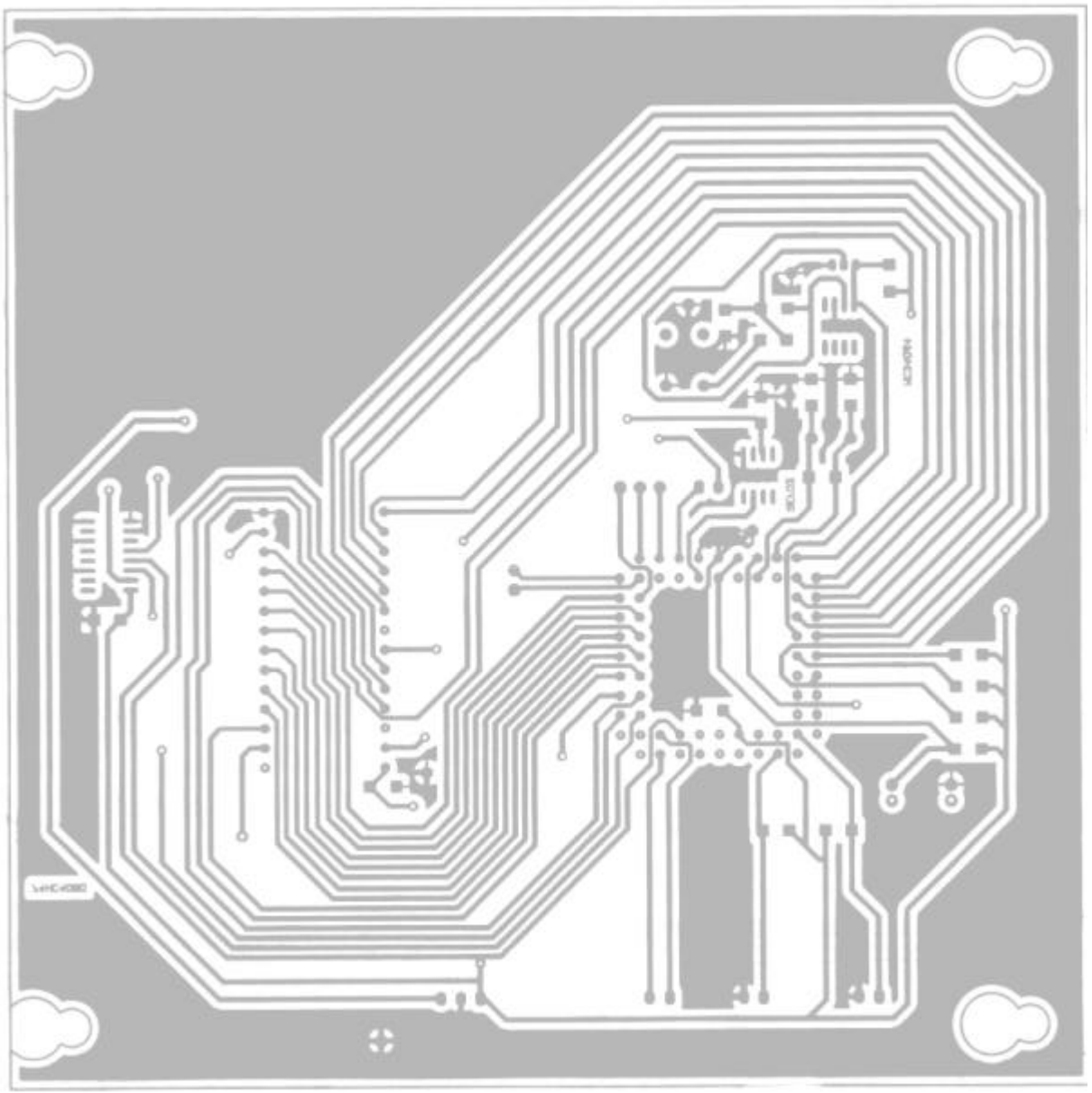
Fichier Excel résultats « Premier lâché » (20 premières mesures).

n° trame	capteur 1	capteur 2	capteur 3	mesure courant	mesure courant (mA)	angle soleil/panneau (°)	bilan énergétique
1	183	2	206	167	327	6	
2	189	2	198	167	327	2	*Bilan moyen par trame avec asservissement*
3	189	2	198	167	327	2	-189 mA
4	179	2	198	167	327	5	
5	177	0	198	113	222	6	
6	145	3	194	165	324	14	
7	7	1	194	165	324	58	
8	13	1	192	165	324	57	
9	21	1	242	165	324	56	
10	11	1	198	103	202	57	
11	201	2	158	131	257	-12	
12	223	1	142	183	359	-21	
13	11	2	118	163	320	56	
14	3	1	51	163	320	58	
15	7	1	246	27	53	59	
16	125	2	118	131	257	-3	
17	229	1	118	183	359	-29	
18	23	0	118	31	61	49	
19	129	3	118	135	265	-5	
20	237	2	118	135	265	-30	

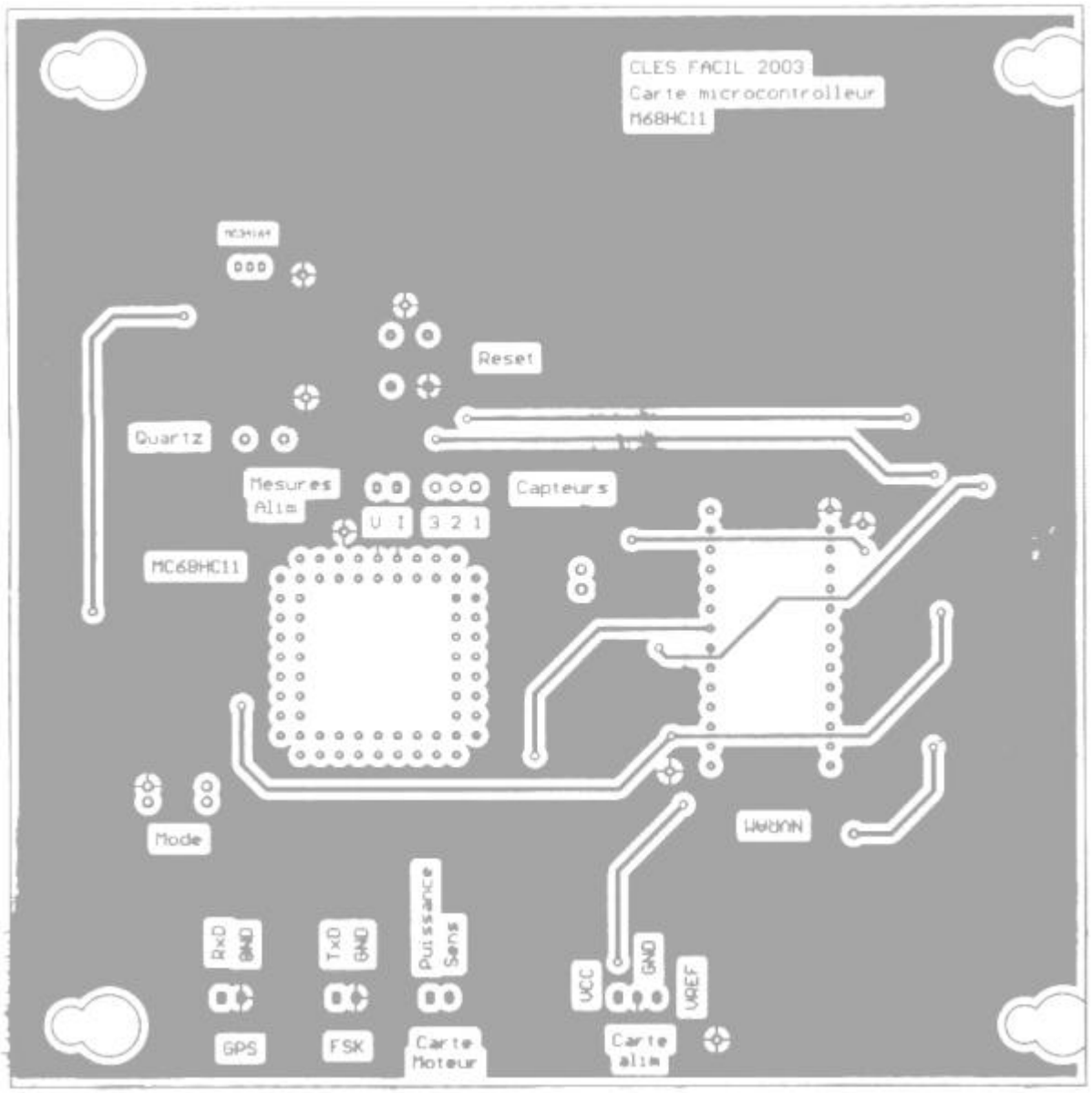
Fichier Excel résultats «Deuxième lâché» (12 premières mesures).

Angle 0-360	Mesure courant	Vie	Cheksum	Checksum ok	consommation (mA)	courant généré (mA)	bilan énergétique (mA)		
								Bilan énergétique par trame sans asservissement	**-235 mA**
244,1	18	1	76	VRAI	280	35	-245		
244,1	20	1	78	VRAI	280	39	-241	*Bilan moyen par trame avec asservissement*	**-189 mA**
244,1	20	1	78	VRAI	280	39	-241	*Gain dû à l'asservissement*	**46 mA**
244,1	19	1	77	VRAI	280	37	-243	*soit*	**19,497 %**
244,1	20	1	78	VRAI	280	39	-241		
244,1	20	1	78	VRAI	280	39	-241		
244,1	20	1	78	VRAI	280	39	-241	*Bilan moyen par trame avec asservissement raté*	**-549 mA**
244,1	20	1	78	VRAI	280	39	-241	*Gain du a l'asservissement*	**-314 mA**
244,1	20	1	78	VRAI	280	39	-241		**133,84 mA**
244,1	19	1	77	VRAI	280	37	-243		
244,1	21	1	79	VRAI	280	41	-239		
244,1	20	1	78	VRAI	280	39	-241		

Annexe plan microcontrôleur

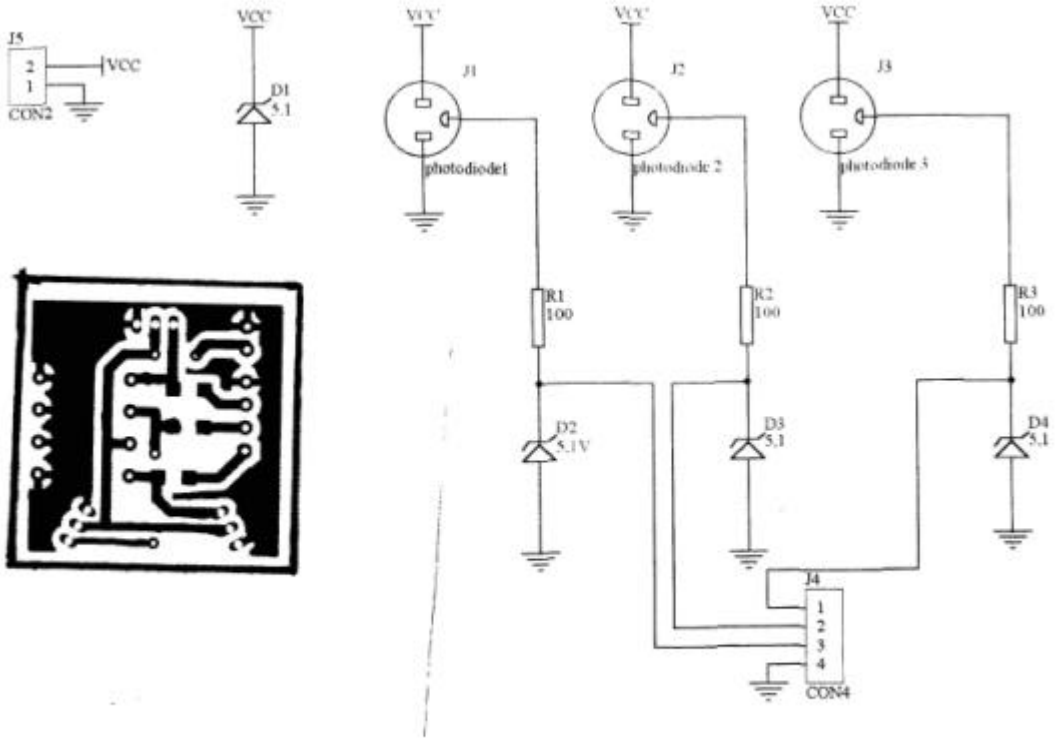


- Typon de la carte microcontrôleur, (verso – côté pistes).



- Typon de la carte microcontrôleur, (recto – côté composants).

Annexe carte capteur



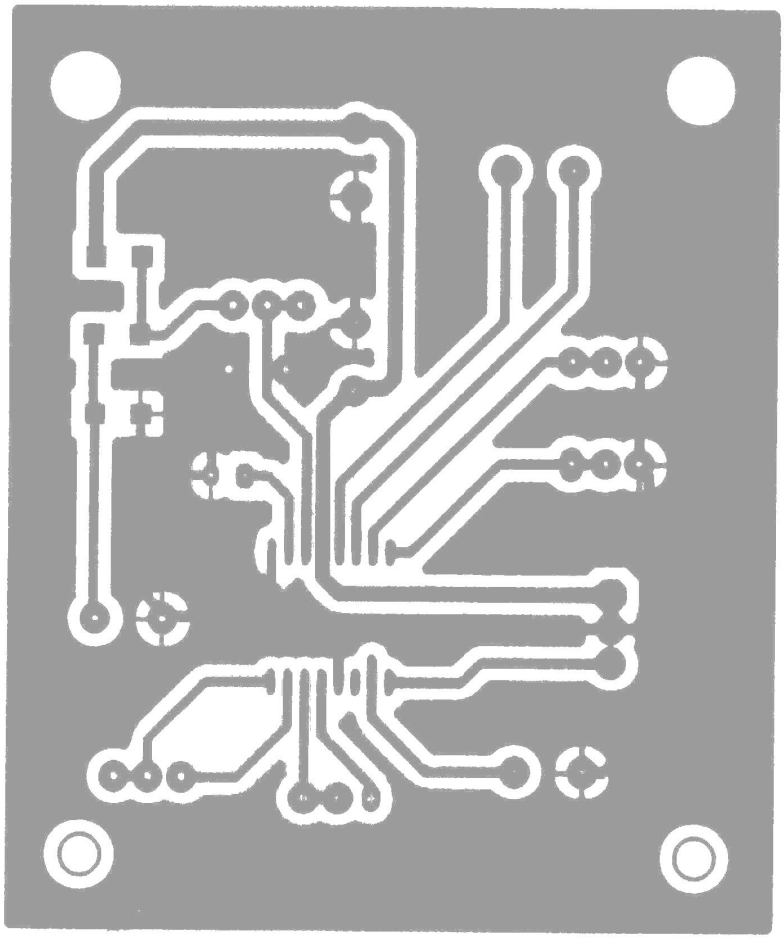
- Schéma et typon de la carte capteur, (verso – côté pistes).

Carte collecteur tournant

- Typon des pistes du collecteur tournant, des lamelles de cuivre fixées sur la tourelle frottaient sur ces pistes pour assurer les connexions électriques.

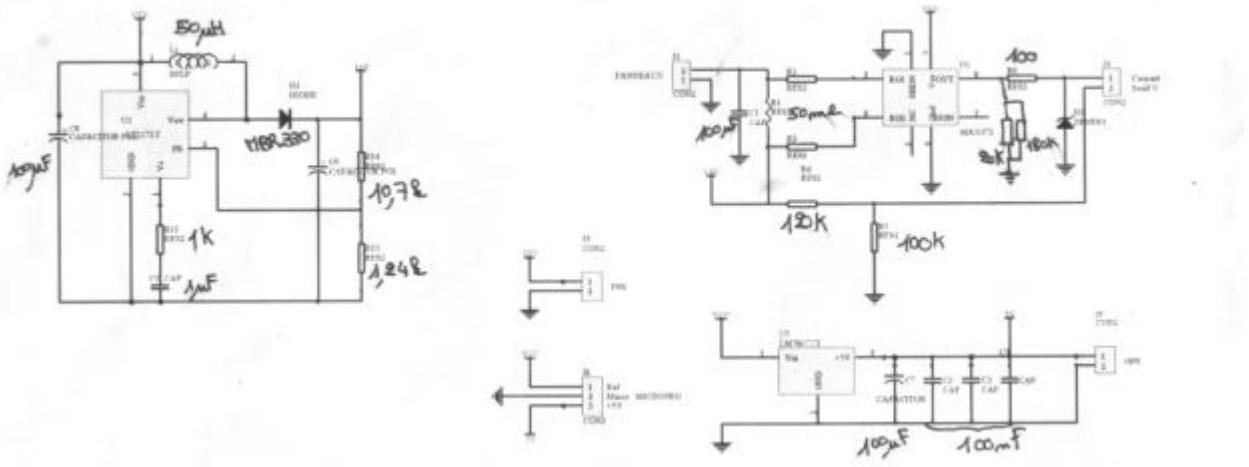
Carte FSK

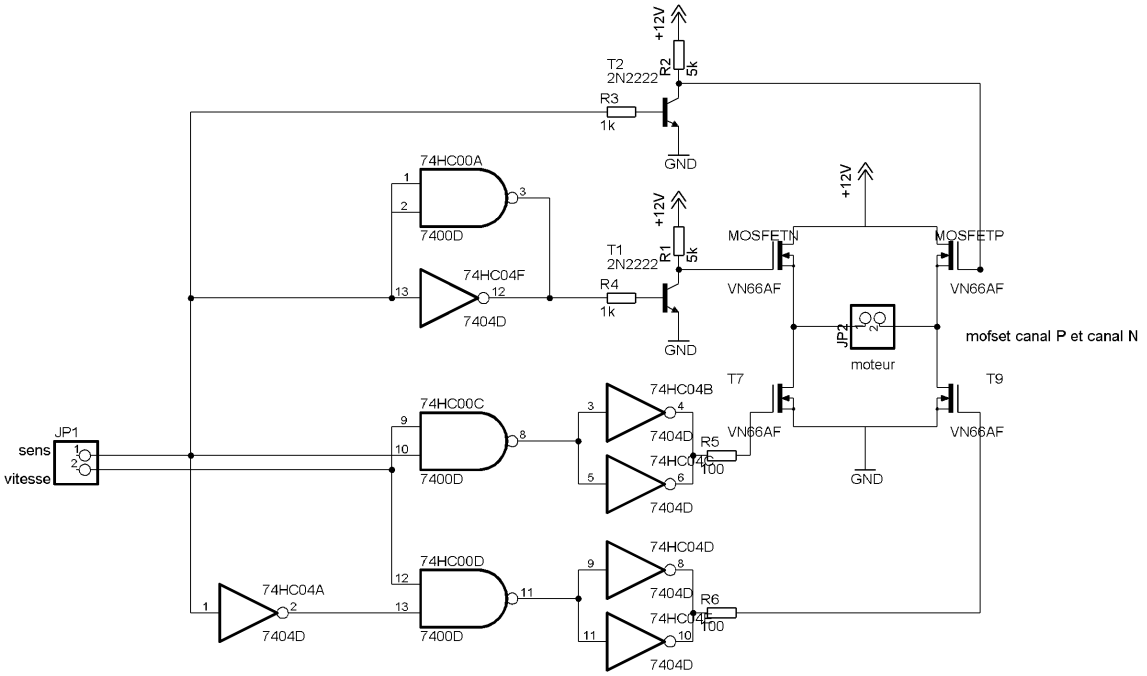
- Typon de la carte modulation FSK, (verso – côté pistes).



Carte alimentation et commande moteur

- Schéma de la carte alimentation (+12V, +5V).





- Schéma de la carte contrôle moteur et implantation des composants de la carte regroupant les alimentations et le contrôle moteur, (recto – côté composants).