Pilote automatique embarqué dans un modèle réduit d'avion

Rapport de Projet I4 ESIEE

Lubin Kerhuel
Suivi par Dr.Dariush Faghani
avec l'aide de Frédéric Péneau

2 octobre 2003

FIG. 1 – *modèle d'aile volante utilisé : IXIR II*

0. Une copie de ce rapport se trouve sur le site http://lubink.free.fr

Table des matières

Table des figures

1 Introduction

Ce rapport présente l'étude d'un pilote automatique pour modèle réduit d'avion. Il aborde la conception d'une centrale inertielle miniature ainsi que la programmation du calculateur embarqué. Le tout ayant permis la réalisation d'un système temps réel miniature intégrée sur un modèle réduit d'aile volante.

1.1 Problématique

Les difficultés de la réalisation d'un pilote automatique pour modèle réduit viennent essentiellement des contraintes provenant de la partie modélisme. En effet, le système complet doit mesurer moins d'une dizaine de centimètres afin d'entrer dans notre modèle réduit. Il doit peser quelques dizaines de grammes pour ne pas détériorer ses qualités de vole. Et enfin, son alimentation électrique doit être la même que celle des servos moteurs, soit environs 4.5 Volts. Le budget de tout ceci devant être le plus réduit possible.

1.2 Domaines abordés

Ce projet fait appel aux trois domaines suivants :
- L'électronique analogique
- Le filtrage numérique
- L'automatisme

La partie électronique analogique consiste en la réalisation du montage global incluant le traitement des signaux provenant des différents capteurs.

Le filtrage numérique est une partie importante. Elle consiste à tester plusieurs méthodes de filtrage des données. Ces essais de filtrage se font sous Matlab. Une étude mathématique du filtre de kalman étendu est présentée dans les annexes du rapport. Cette partie comprend aussi l'étude de la gestion temps réel des différents signaux.

Une fois les données filtrées dans le microcontrôleur, on étudie les méthodes de stabilisation du modèle.

2 Etude

2.1 Modèle de l'avion sous Simulink

Le modèle d'avion réalisé sous simulink permet de comprendre la mécanique du vol. Une fois les constantes du modèle fixés, celui-ci permet de calculer un asservissement optimal ainsi que d'effectuer des simulations. Sur ce modèle, on a placé quatre Accéléromètres placés en croix. Cette disposition des capteurs est étudiée dans l'étude des Centrales Inertielles.

2.1.1 Présentation générale

Le modèle réalisé sous simulink comprend 3 parties distinctes que nous développerons ici. Ces trois parties sont reliées entre elles comme le montre le schéma de la figure 2 qui permet de simuler le comportement de l'ensemble. La première partie est le grand bloc *Dynamics* présenté sur la figure 3 correspondant à la modélisation des forces et des moments de l'avion. Le comportement de l'avion est modélisé dans le bloc *EQ. of Motion* de la figure 7. Et en dernier, le comportement des accéléromètres est modélisé dans le bloc *Accelerometers* de la figure 8.
Le bloc appelé *6DoF Animation* permet d'obtenir une visualisation graphique en traçant l'avion en trois dimensions qui évolue dans l'espace. L'avion laisse une traînée derrière lui permettant de visualiser sa trajectoire.

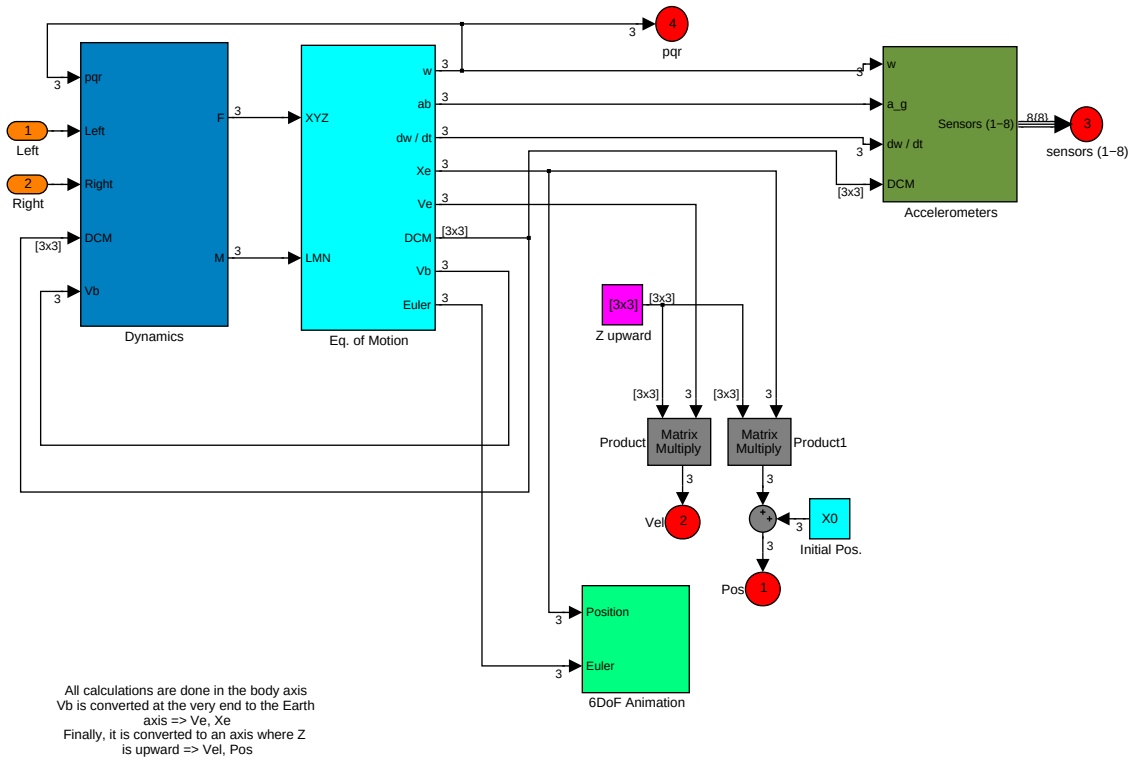

FIG. 2 – *Modele Avion sous niveau 1*

2.1.2 Dynamique de l'aile volante

L'aile est soumis à 2 types de force : d'une part, son poids $P = M * g$ et d'autre part, la résultante aérodynamique créée par son avancement dans l'air. Ces deux forces peuvent se décomposer en une force F appliquée au centre de gravité de l'aile et en un moment M. Le

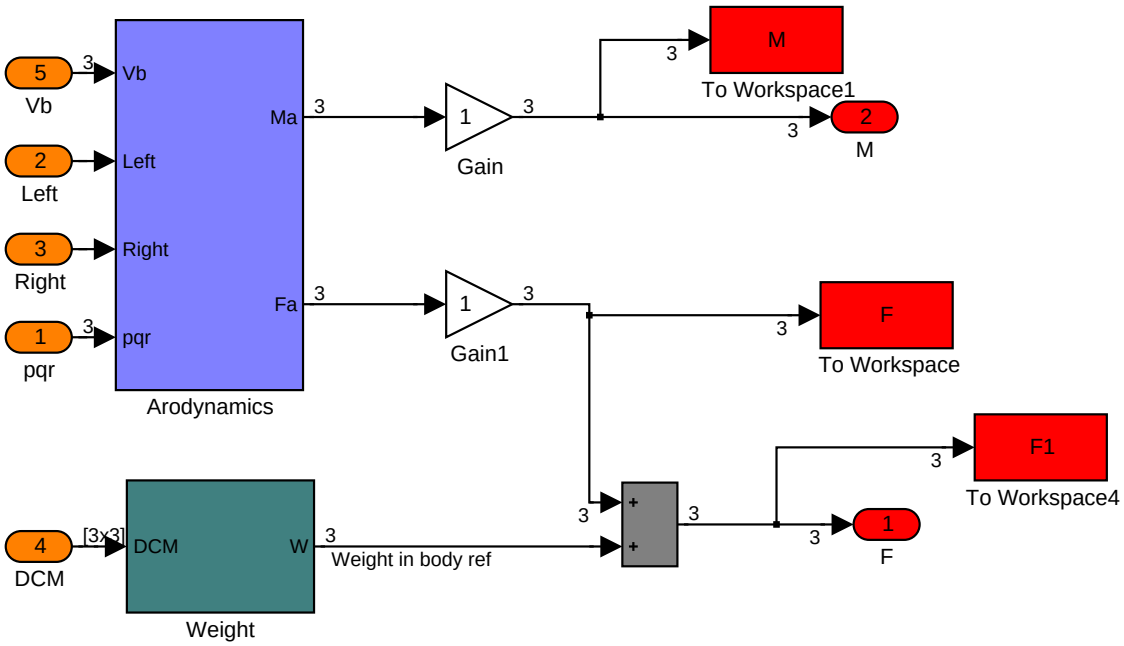

FIG. 3 – *Modele Avion Dynamiques*

poids ne crée aucun moment sur l'aile. Les forces étant exprimées dans le repaire de l'aile, la direction de la force due au poids est calculée en utilisant une matrice de transformation appelée ici DCM pour Direct Cosine Matrix. C'est la matrice de rotation permettant de passer du repère terrestre au repère de l'aile.

La figure 4 détaille le bloc appelé Aerodynamics de la figure 3 et montre la décomposition de l'aile en éléments simples pour simuler séparément chaque partie de l'aile. Ainsi, Les 3 parties composant l'aile sont les ailes, les winglets et enfin, les ailerons. On notera que chacun de ces blocs a comme entrée la vitesse de l'aile et sa vitesse angulaire. Les ailerons ont en plus de ces entrées leurs entrées de commande déterminant leur angle par rapport à leur position de repos. On trouve à la sortie de ces éléments la force engendrée dans le repaire de l'aile ainsi que le moment.

La figure 5 montre la modélisation des ailes droite et gauche. Le comportement de chaque aile est décrit dans un bloc prenant comme unique entrée sa vitesse par rapport à l'air. Cette vitesse est calculée pour le centre aérodynamique de l'aile en utilisant les vitesses de translation et de rotation respectivement Vb_G et pqr.

La modélisation de l'aile droite est détaillée dans la figure 6. La premier bloc appelé Alpha-Beta permet de calculer l'angle d'incidence Alpha et de lacet Beta à partir du vecteur vitesse Vb_M. La portance est inversement proportionnelle à l'angle de lacet de l'avion, décalé de l'angle de la flèche de l'aile. Ceci constitue la partie supérieure du schéma. Ensuite, le coefficient de portance est calculé à partir des fonctions donnant la portance en fonction de l'angle d'incidence α. Ces fonctions C_x et C_z sont tirées de la littérature dédiée et sont normalement déterminées en soufflerie. La portance et la trainée étant exprimées dans le repère constitué par la vitesse de l'aile, il faut les transformer dans le repère de l'aile. Ceci est réalisé dans le bloc appelé Aero -> Body.

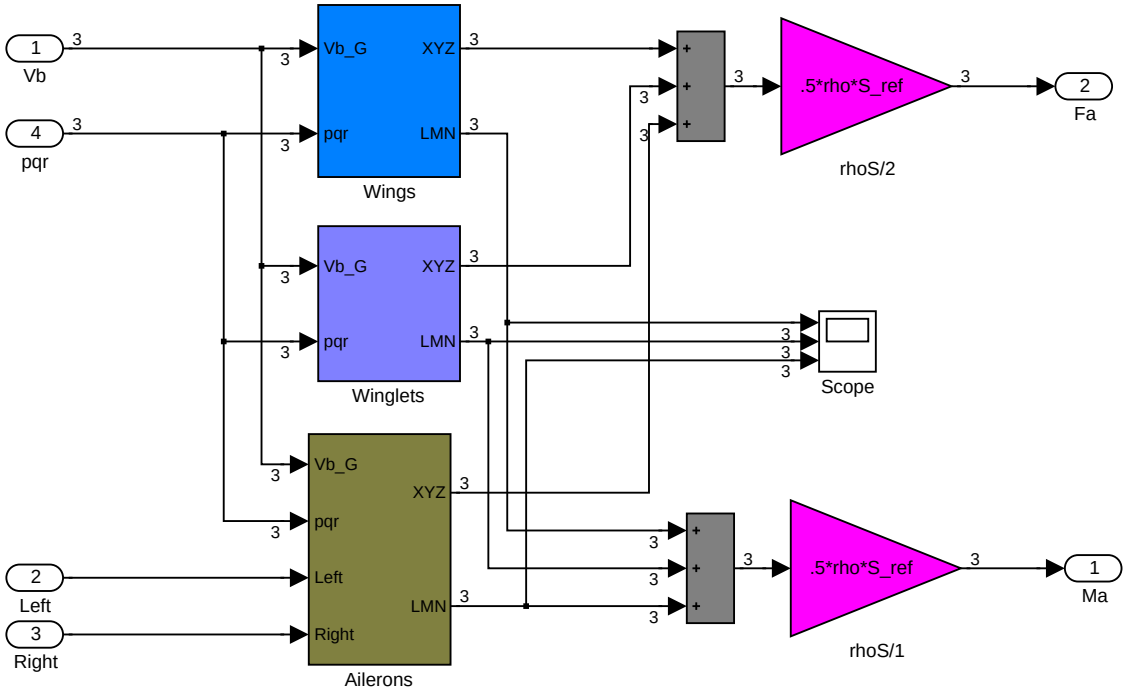

FIG. 4 – *Modele Avion Aerodynamique*

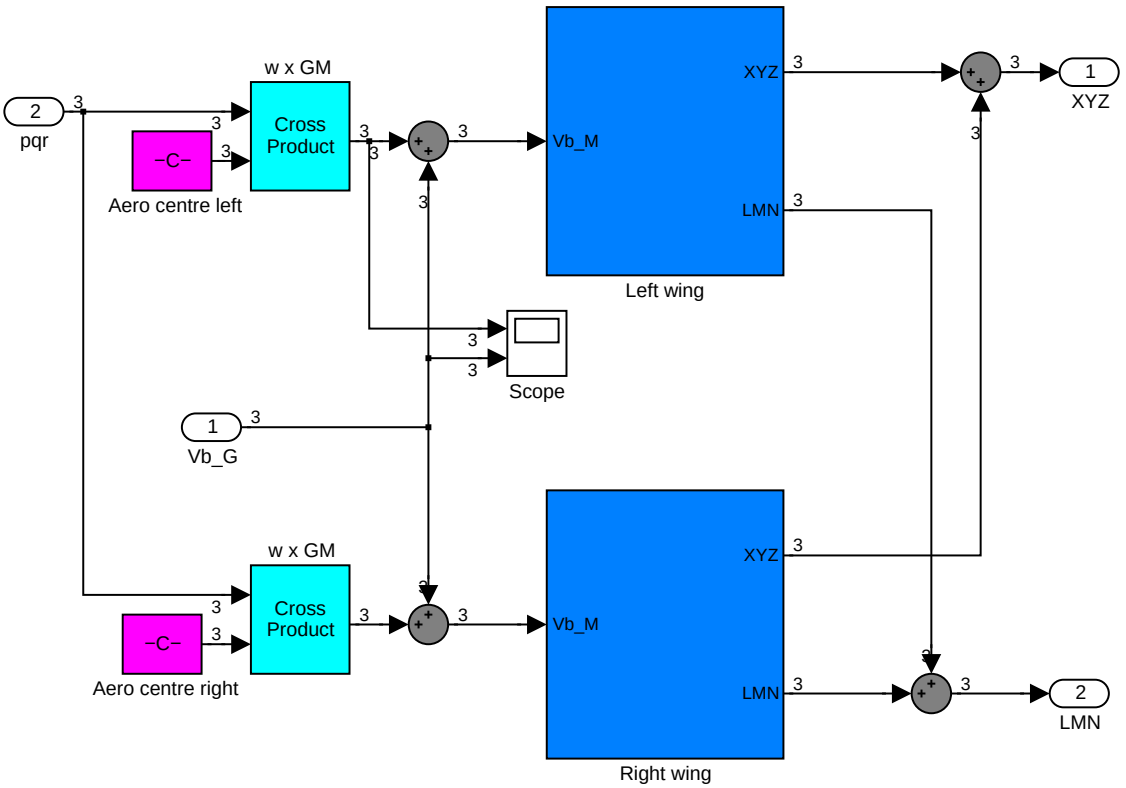

FIG. 5 – *Modele Avion : Aerodynamique des ailes*

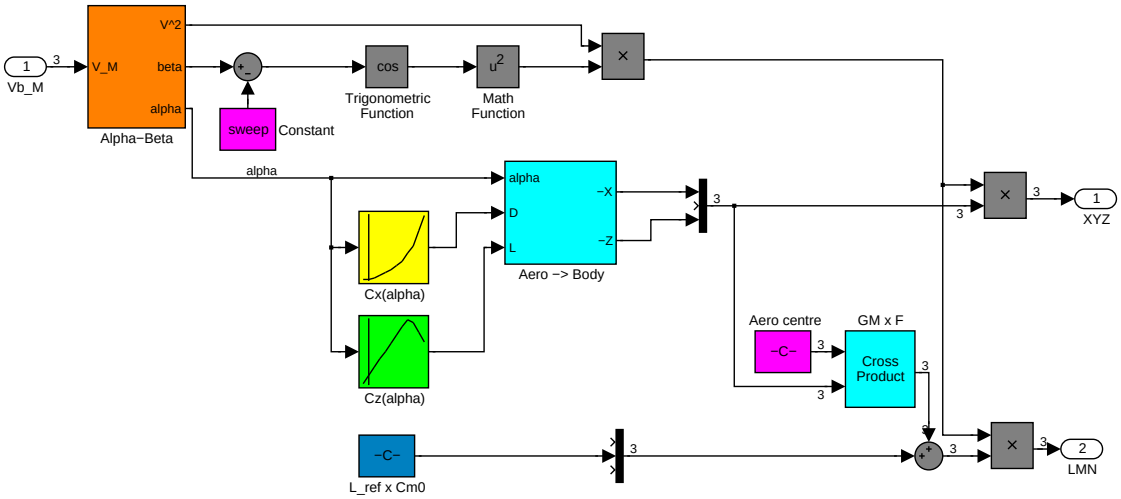

FIG. 6 – *Modele Avion : Aerodynamique aile droite*

2.1.3 Equations du mouvement

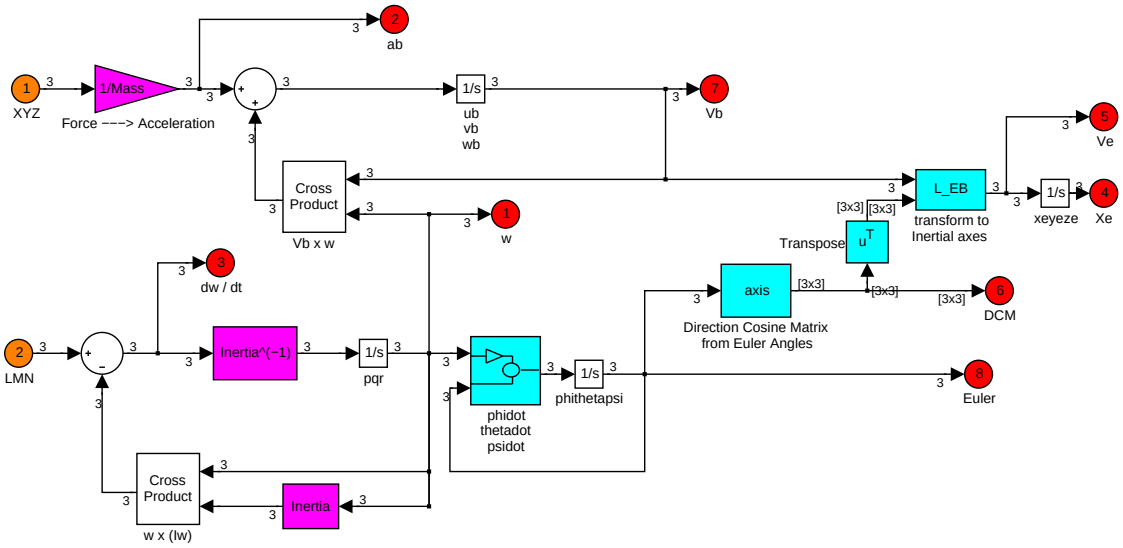

FIG. 7 – *Modele Avion Equations des Mouvements*

La figure 7 modélise le comportement de l'aile à partir des forces et des couples auxquels il est soumis. On utilise ici les équations de la dynamique fondamentale.

2.1.4 Simulation des Acceleromètres

On cherche à simuler la centrale inertiel comprenant 4 accéléromètres décrites dans la section suivante. L'emplacement des quatres accéléromètres sont représentés dans la figure 8p.8.Chaque composant permet de mesurer deux axes. Les accélérations obtenues par ces accéléromètres sont d'une part dues à l'accélération de la gravité et d'autre part dues aux accélérations des composants. Ces accélérations sont représentées dans la figure 9. La distance GM étant fixe,on est parti de l'équation :

$$\frac{dGM}{dt} = w \wedge GM \quad (1)$$

$$\frac{d^2GM}{dt^2} = w \wedge (w \wedge GM) + \frac{\delta w}{\delta t} \wedge GM \quad (2)$$

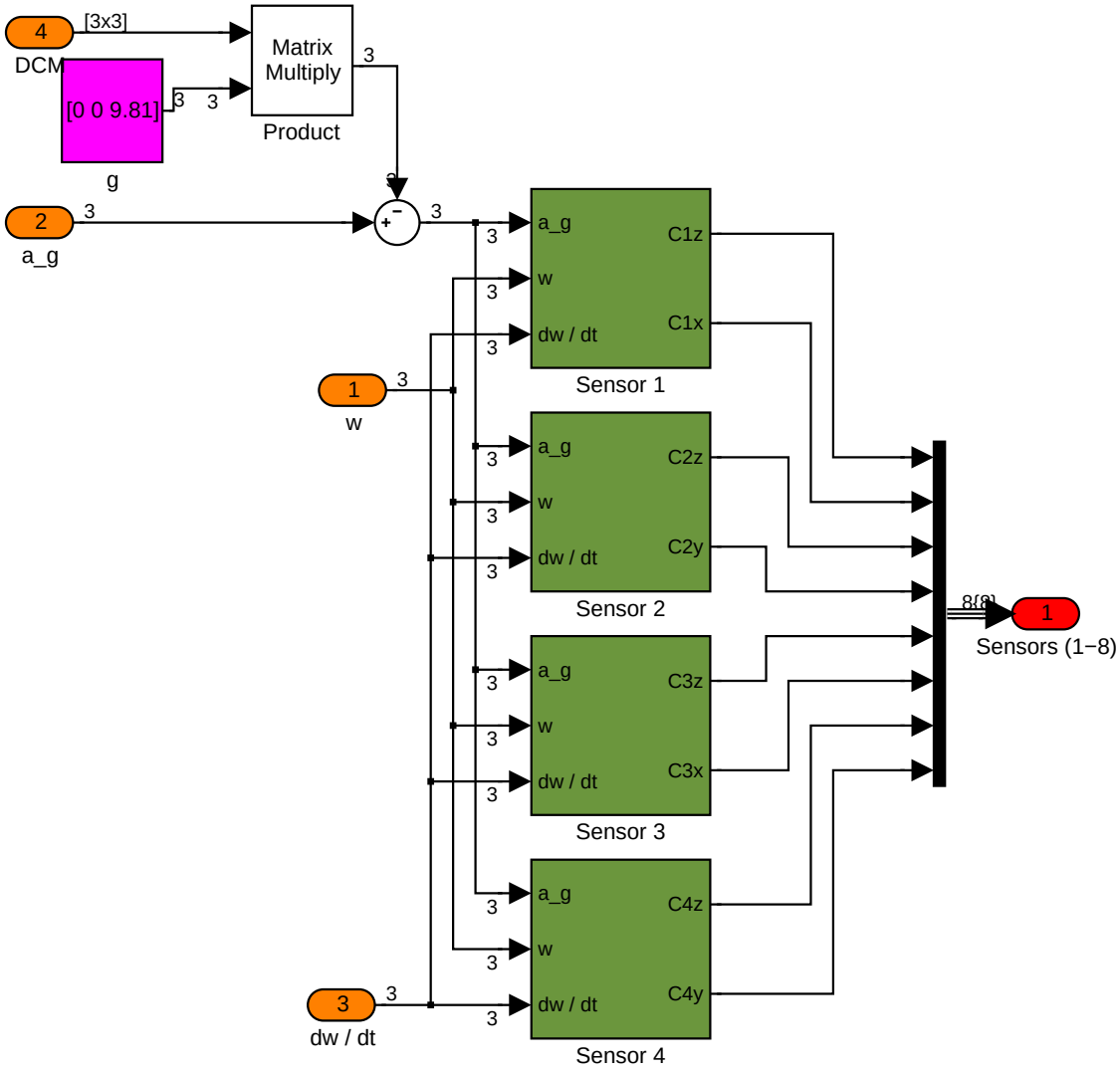

FIG. 8 – *Modele Avion Accelerometres*

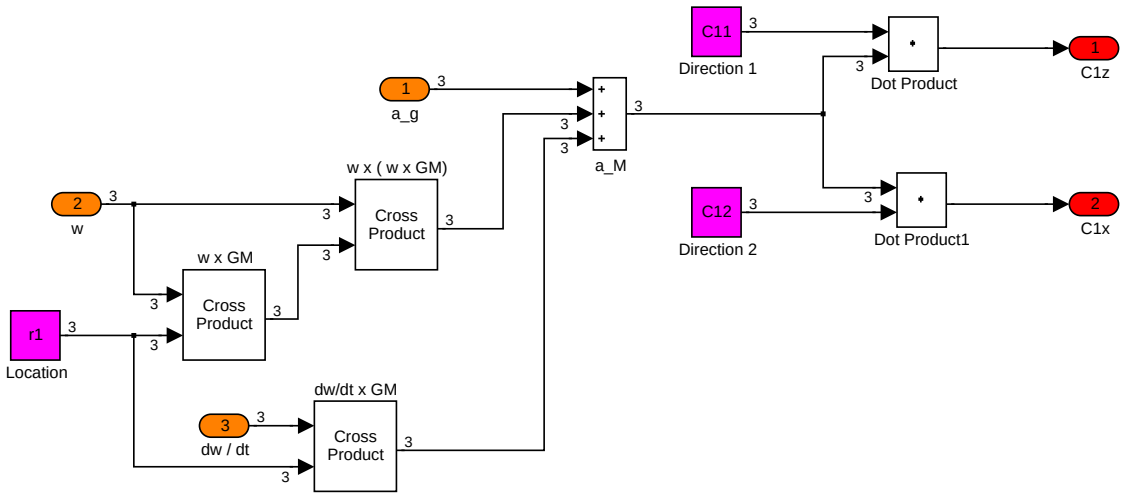

FIG. 9 – *Modele Avion Accelerometre detail*

2.2 Centrale Inertielle Ideale

2.2.1 Différents types de centrales inertielles

Jusqu'à récemment, les centrales inertielles étaient réservées aux domaines très pointus de l'aéronautique et de l'espace. Elles étaient lourdes et chères. Grace aux progrès de la microélectronique, le couplage de certain micro capteurs appelé MEMS permet de réaliser des centrales ayant des performances acceptables tout en restant peu onéreuses. Aujourd'hui, les centrales inertielles apparaissent dans les combinaisons de jeux virtuels ainsi que dans les modèles réduits d'hélicoptères où ils apportent une précieuse aide au pilotage.

Les gyroscopes mécaniques sont les plus anciens et les plus répandus. Ils sont constitués d'un disque en rotation sur un axe, lui-même libre de mouvement de rotation sur ces trois axes. La toupille ainsi constituée, grâce au phénomène de précession, est supposée regarder fixement une étoile quel que soit le mouvement de son socle. Afin de connaître la position angulaire sur les 3 angles de l'espace, deux disques rotatifs sur deux axes non parallèles sont nécessaires.

les gyroscopes optiques utilisent un rayon laser réfléchi par trois miroirs formant ainsi un cercle de lumière. La donnée de rotation est donnée par le défilement des franges d'interférences. Il faut un cercle de laser par axe. Ces gyroscopes très précis sont ceux qui sont utilisés dans les fusées.

Les gyroscopes elctroniques sont ceux que nous allons utilisés. Ils existent en plusieurs variantes. Les capteurs microélectronique appelés " Microelectromechanical Systems " ou MEMS pouvant être utilisés sont les suivants :

- Accéléromètres : pour la mesure des accélérations et de l'accélération de gravité G en statique
- Gyromètres : pour la mesure des vitesses angulaires
- Magnétomètres : pour la mesure du champ magnétique terrestre

Les accéléromètres permettent, lorsque les capteurs sont statiques, de déterminer les angles sur le plan perpendiculaire à l'axe vertical en mesurant l'accélération de la gravité G sur les trois dimensions. Cependant, les mesures sont faussées dès lors que les capteurs subissent une accélération supplémentaire à G. C'est pourquoi lorsque les mesures sont biaisées, il est nécessaire de combiner ces données avec celle de gyromètres afin de connaître la vitesse angulaire de la centrale inertielle et ainsi retrouver par intégration l'attitude de celle-ci.

Comme les accéléromètres, les magnétomètres permettent de déterminer les angles de la centrale avec le plan perpendiculaire au champ magnétique ambiant. Le champ magnétique étant non colinéaire à la gravité G, il est possible en combinant les données des Accéléromètres et des Magnétomètres de connaître complètement la position angulaire de la centrale inertielle lorsque celle-ci est statique. On obtient alors 3 angles.

Les gyromètres permettent d'obtenir après intégration des données une estimation de la position angulaire. L'utilisation seule de gyromètre n'est pas envisageable car l'intégration des données de vitesse angulaire intègre aussi le bruit. Celui-ci n'étant pas centré, on dérive inévitablement de la valeur réelle de l'angle. Il est donc nécessaire d'utiliser des accéléromètres et-ou de magnétomètres afin de corriger le résultat de l'intégration.

Deux accéléromètres placés sur le même axe de part et d'autre d'une tige de longueur connue permettent d'obtenir l'accélération angulaire de celle-ci quel que soit le centre de rotation. Ce dispositif que nous expérimentons ici fonctionne mais est moins précis que des gyromètres. En effet, à partir des accélérations angulaires, il est nécessaire d'intégrer deux fois la donnée pour obtenir la position par rapport à l'angle initial. Cette double intégration engendre des erreurs importantes.

2.2.2 Filtre de Kalman

La centrale inertielle la plus efficace utilisant des composants MEMS utilise chaque type de capteur dans au moins trois axes, afin d'avoir une redondance d'information. La difficulté pour réaliser ce type de centrale est de combiner toutes ces données de manière optimale. De plus, pour une application donnée, il est possible d'inclure dans le calcul l'attitude de la centrale inertielle. Cette dernière astuce permettant de pondérer dynamiquement l'influence de chaque capteur afin de donner à ceux dont les données semble le plus valide un poids prépondérant. Ce calcul est effectué par un filtre adaptatif spécifique appelé filtre de kalman. Le fonctionnement de celui-ci est développé dans les annexes.

2.3 Centrale inertielle avec 4 accéléromètres

2.3.1 Disposition des capteurs

Le composant utilisé : l'ADXL 202 permet de mesurer l'accélération sur deux axes perpendiculaire. Les 4 Accéléromètres mesurant chacun deux axes sont placés au bout d'une croix (fig : 10). Cette disposition permet de connaître l'accélération de la croix dans les 3 dimensions, ainsi que les accélérations angulaires de celle-ci. Bien que cette disposition permette de mesurer ces deux phénomènes, elle ne permet pas de mesurer la force centrifuge lors des rotations.

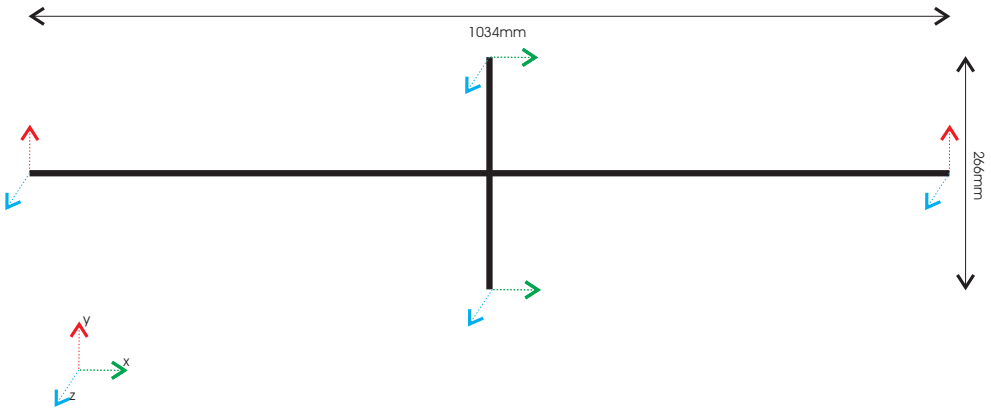

FIG. 10 – *Disposition physique des capteurs*

2.3.2 Traitement des Données

On présente ici les différentes méthodes testées afin d'obtenir l'attitude de la centrale inertielle. Les calculs initialement réalisés dans des fichiers de commande Matlab (.m) ont été ensuite réalisés dans Simulink. On ne présentera ici que les schémas Simulink. En dehors du fait que ceux-ci soient plus intuitifs, ils ont l'avantage d'être convertibles en programme source C compilable avec un compilateur pour microcontrôleur en utilisant le package *Real Time Embedded Coder* de Matlab.

2.3.3 Attitude Statique

On réalise ici une centrale inertielle en supposant que celle-ci soit statique. On néglige les effets d'une accélération de la centrale sur les capteurs. Le schéma d'ensemble (11) permet de voir les différentes étapes du calcul. La sortie 3 est le résultat du calcul. Les sorties 1 et 2 montrent le résultat des calculs intermédiaires.

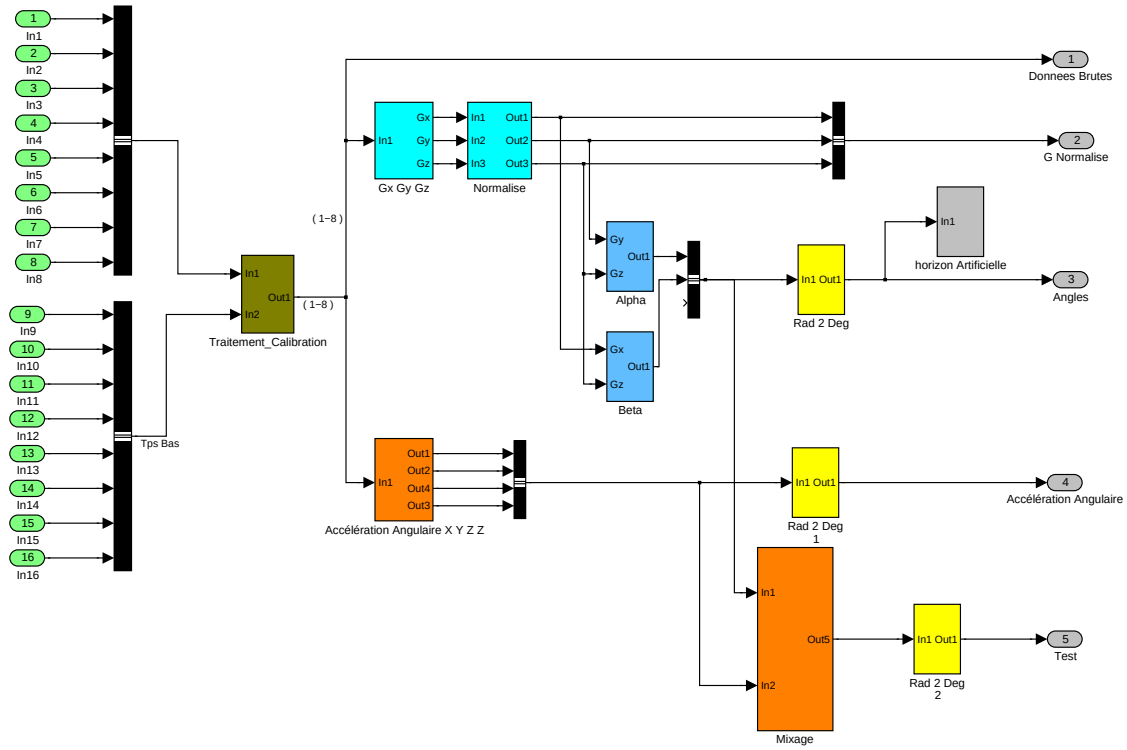

FIG. 11 – *schéma d'enssemble*

Les calculs préliminaires permettent à partir des modulations de largeur d'impulsion provenant des capteurs (fig.16 p.23) de déterminer les accélérations correspondantes. Les 8 entrées du haut sont les durées des impulsions hautes et les 8 entrées du bas sont les durées entre ces impulsions. La somme des deux étant le temps des cycles. En faisant le rapport cyclique et en le mettant à l'échelle, on obtient les accélérations sur chacun des axes (fig.12). Les capteurs ayant été calibrés comme expliqué dans le chapitre traitant des capteurs, on ajoute un offset et un second gain à chaque valeur obtenant ainsi le résultat optimal.

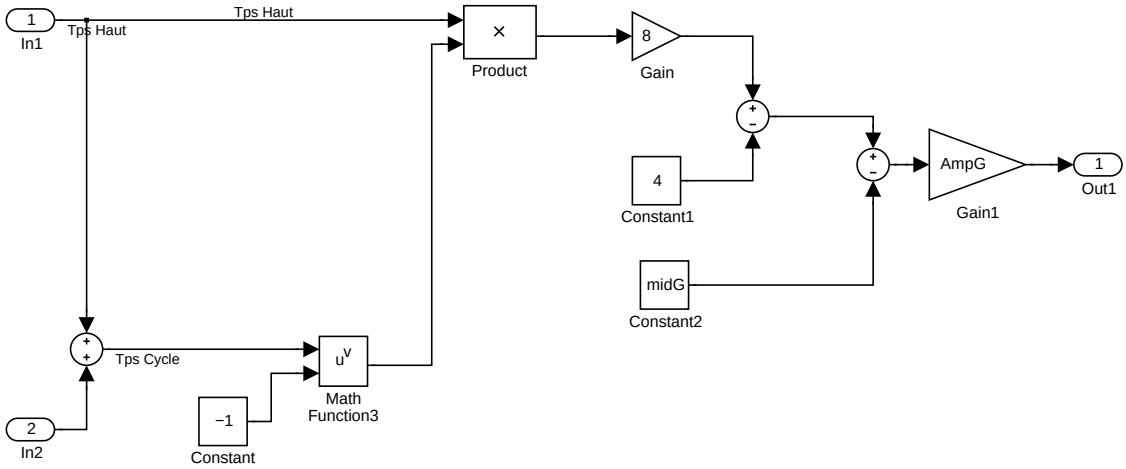

FIG. 12 – *Calibration et Calcul de l'accélération*

On obtient 8 valeurs :

- Gz_1
- Gx_1
- Gz_2
- Gy_2
- Gz_3
- Gy_3
- Gz_4
- Gx_4

La centrale étant statique, les capteurs donnent l'accélération de la pesanteur. Celle-ci étant perpendiculaire au plan horizontal, connaître la direction du vecteur d'accélération G revient à connaître la position du plan horizontal.

On moyenne les 8 données obtenues afin de connaître l'accélération dans les 3 axes (fig.13).

Le fait de réaliser une moyenne permet de diminuer le bruit.

Le bruit des capteurs ayant les mêmes caractéristiques, La variance de la somme des deux données est plus faible que celle de chaque donnée.

Le vecteur obtenu est théoriquement normalisé. Cependant, il existe toujours une petite erreur. Afin d'améliorer les résultats des fonctions trigonométriques utilisées, on le normalise en le multipliant par un facteur K proche de 1 (fig.14)

$$K = \frac{1}{\sqrt{Gx^2 + Gy^2 + Gz^2}} \tag{3}$$

Les angles Alpha et Beta sont calculés de la même manière. Afin d'obtenir la meilleure résolution pour les angles proches de 0, les angles sont calculés à partir de l'arc sinus de l'accélération horizontal des deux axes. Le troisième axe est utilisé afin de lever l'indétermination (fig.15). On peut ainsi obtenir une mesure de l'angle sur tout le cercle.

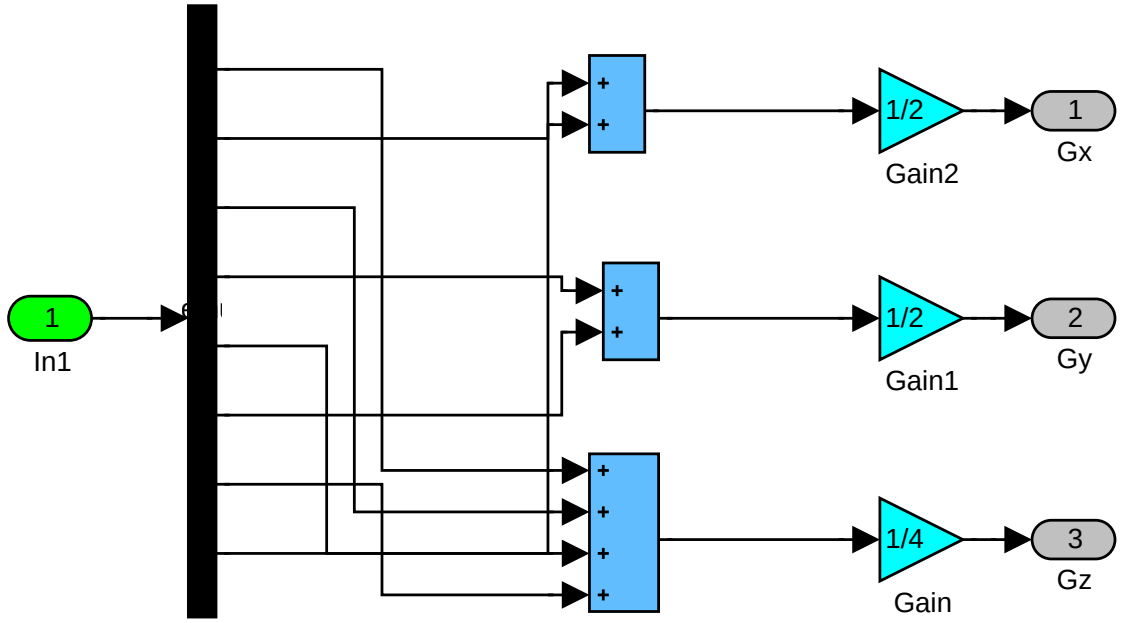

FIG. 13 – *Moyenne de l'accélérationde sur les 3 axes*

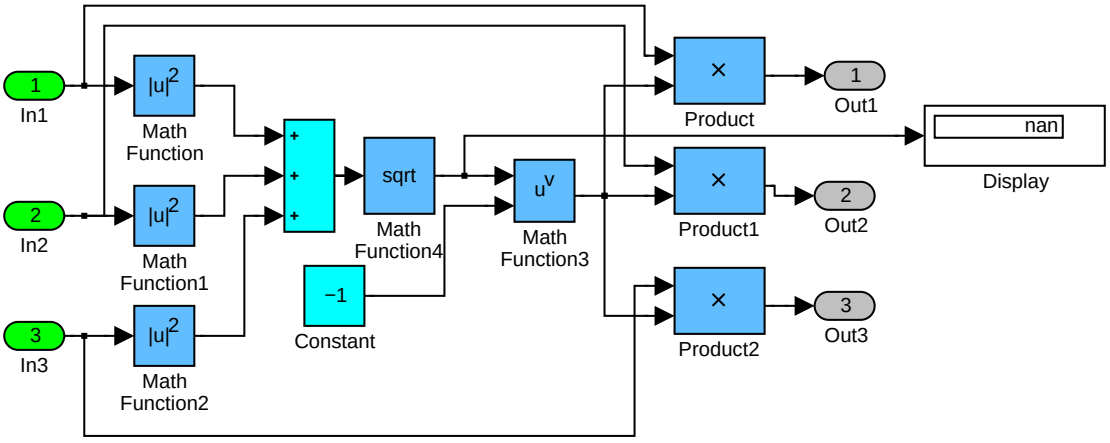

FIG. 14 – *Normalisation de l'accélération de la pesanteur*

2.3.4 Attitude Dynamique

La disposition des capteurs au bout d'une croix permet d'extraire les accélérations angulaire de cette croix. Cette information peut être intégrée au calcul de l'angle et ainsi permettre de corriger de l'erreur lorsque la croix est soumise à une accélération. Les quelques tests effectués n'ont pas été poussés et les résultats ne sont donc pas satisfaisant. Mais il vaudrait la peine de revenir sur cette expérimentation.

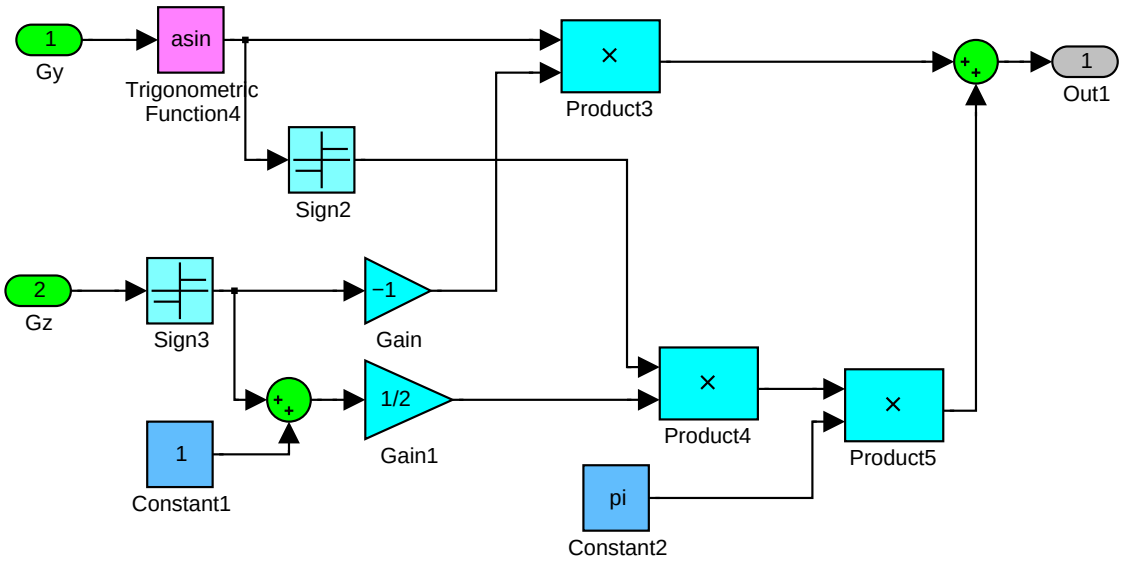

FIG. 15 – *Calcul des Angles Alpha et Beta*

2.4 Centrale inertielle avec 1 accéléromètre

On réalise un montage simplifié comprenant un unique accéléromètre. Celui-ci est placé au centre aérodynamique de l'aile. Les mesures sont effectués dans le plan horizontal, suivant l'axe de rouli et de tanguage de l'aile. Un simple arc sinus des valeurs acquises permet d'obtenir les angles du capteur avec le plan horizontal. Ces calculs sont valides lorsque le capteur est statique. Les résultats obtenus sont satisfaisants pour être utilisés dans le pilote automatique.

2.5 Pilote Automatique

2.5.1 Fonctionnement

Le pilote automatique réalisé permet de stabiliser l'assiette (axe de tangage) et l'inclinaison (axe de roulis) de l'aile sur deux angles. Ces deux angles sont déterminés par les deux voies de la télécommande. Lorsque celle-ci est éteinte, le calculateur verrouille ces deux angles sur leur valeur courante. L'aile vole alors de manière autonome.

Lors du décollage, les capteurs subissent une accélération brutale. Cette accélération fausse l'estimation de l'horizontal. Le lancée main implique de lancer l'aile d'un coup sec, l'accélération subite est alors importante. Lors d'un tractage par un sandow de 8mm et d'une longueur de 25m, l'accélération est moins forte, mais est longue.Or, lorsque l'aile accélère, le calculateur croit que celui-ci cabre. Il essaye alors de rétablir la trajectoire en envoyant une commande pour piquer. Ceci entraîne le crash du modèle. Il est donc nécessaire de désenclencher l'asservissement durant la phase de décollage. Pour ce, il suffit d'éteindre la télécommande puis de la rallumer. Le système n'enclenche pas l'asservissement durant les dix premières secondes. Un buzzer émettant un son d'amplitude croissante permet de déterminer approximativement le temps qu'il reste avant la mise en route du pilote automatique. Il est ainsi possible d'effectuer de beaux décollages.

Un accéléromètre placé au centre aerodynamique de l'aile permet de déterminer les angles de ce dernier avec l'horizontal. On approxime la fonction arc sinus par la première bissectrice du plan. Le calcul est simplifié, en conséquence, l'angle calculé sera moins précis.

Le calculateur est conçu pour une aile volante. Les surfaces de contrôle de celle-ci sont les deux ailerons situés sur les ailes droite et gauche. Un mouvement dans le même sens des deux ailerons permet de faire cabrer ou d'incliner l'aile alors qu'un mouvement différentiel amène l'aile en virage. Chaque servos moteur commandant les ailerons reçoit une instruction qui est un mixage entre la commande en profondeur et en roulis.

Il est possible de piloter un avion comportant un empennage à l'aide des seuls ailerons. Cependant, les caractéristiques de vole de celui-ci sont nettement diminuer lorsque la commande tend à faire piquer ou cabrer. Il est préférable d'adapter le système avec une commande pour la profondeur sur l'empennage et une commande différentielle pour le roulis sur les ailerons.

2.5.2 Difficultés

Les principales difficultés viennent de la gestion des différents signaux à acquérir et à émettre ainsi que de la faible résolution des calculs dans le microcontrôleur.

Le microcontrôleur utilisé comportant peu de place mémoire pour le programme comme pour les variables, on se restreint à l'utilisation d'entier de 16 bits. Ceci implique de faire très attention aux dépassements de capacité et limite la complexité des calculs effectués.

Les différentes valeurs à acquérir et à émettre sont modulés en largeur d'impulsion (PWM pour Pulse Width Modulation). La valeur transmise est obtenue en chronométrant la durée de cette impulsion. Il y a au total 4 signaux à acquérir : les deux premiers sont les valeurs émises par les deux voies de la télécommande. Les deux suivants sont les valeurs des accélérations captées par le capteur. Les signaux du récepteur sont synchronisés dans le sens où

les eux impulsions arrivent l'une après l'autre. Mais les impulsions du capteur ne sont pas synchronisées avec les impulsions du récepteur. Le système doit donc être capable de chronométrer de manière précise plusieurs signaux à la fois. De plus, il doit moduler deux signaux en largeur d'impulsion pour commander les servos. Tout erreur de chronométrage de l'un de ces signaux entraîne une erreur sur la donnée reçue ou émise, ce qui au final fait vibrer les servos moteurs. Ceci est très mauvais car non seulement, cela consomme beaucoup d'énergie et perturbe le vole de l'aile, mais le montage électronique est aussi parasité par les baisses de tension engendré.

3 Experimentations

3.1 Accéléromètres ADXL 202

Cette section explique comment utiliser les accéléromètres et justifie le choix des valeurs des résistances et des capacités dans le montage électronique. La disposition des capteurs pour les tests effectués est ensuite développée. Et Enfin, il décrit comment sont acquises les données avec le microcontrôleur et analyse la précision des échantillons ainsi récupérés.

3.1.1 Montage électronique

Les accéléromètres de la famille ADXL sont simples à utiliser. Deux broches fournissent deux données analogiques et deux autres fournissent les mêmes données modulées en largeur d'impulsion à connecter sur des entrées numériques. On utilise ces 2 signaux modulés pour acquérir les données. La période de la modulation est configurable à l'aide d'une résistance. Deux capacités permettent de filtrer les signaux analogiques utilisés dans les capteurs avant qu'ils ne soient convertis numériquement en largeur d'impulsion. C'est le filtre anti-repliement.

Le quartz utilisé pour le PIC oscille à $Fosc = 20MHz$. La vitesse des instructions est donc de :

$$Fosc/4 = 5Mhz$$

La vitesse maximale des compteurs est égale à la vitesse d'exécution des instructions soit $5Mhz$. Les convertisseurs utilisés dans le capteur ADXL ont une résolution de 14 bits. Si on veut limiter les pertes en précision, le temps T_2 d'un cycle d'impulsion doit être assez long pour permettre au pic de compter jusqu'à 2^{14}.

$$T_2 > \frac{2^{14}}{Fosc/4} \tag{4}$$

$$Rset = 125 * 10^6 * T_2 \tag{5}$$

$$Rset > 409600 \tag{6}$$

La résistance minimum que l'on peut utiliser sans perdre en précision est de 409600Ω. On utilise une résistance de $470K\Omega$ La résolution de l'acquisition est alors de :

$$T_2 = \frac{Rset}{125 * 10^6} \tag{7}$$

$$NbrBitsResolution = log2\left(T_2 * \frac{fosc}{4}\right) \tag{8}$$

$$NbrBitsResolution = 14.1984 \tag{9}$$

On est donc bien au-dessus des 14 bits du convertisseur du capteur. La fréquence d'acquisition est :

$$FreqAdxl = \frac{1}{T_2} \tag{10a}$$

$$FreqAdxl = 266Hz \tag{10b}$$

On choisi une capacité de filtrage de $0.22\mu F$ ce qui correspond à une fréquence de coupure à 3dB de 25Hz. La fréquence d'échantillonnage élevée permet de réduire le bruit numérique généré lors de l'acquisition des données.

On ajoute à cela une capacité de découplage de $65nF$ ainsi qu'une résistance de protection de 100Ω

Les capteurs sont câblés de telles façons que l'on peut remplacer sans souder la résistance déterminant la période T_2 ainsi que les deux capacités de filtrage des deux axes du capteur.

3.1.2 Acquisition des données

On utilise les entrées numériques du PIC 16f877. Il faut acquérir 8 signaux modulés en largeur d'impulsion. La méthode la plus simple serait de lire en permanence 8 entrées du PIC en chronométrant les temps des états haut et bas de chacune de ces entrées. Mais cette méthode utiliserait toutes les ressources du PIC qui ne pourrait rien faire d'autre puisque ces routines seraient en temps critique. L'utilisation des interruptions est plus appropriée.

Deux entrées spécifiques CCP1 et CCP2 permettent d'acquérir des signaux PWM. Chaque changement d'état de l'une de ces entrées entraînant l'arrêt d'un compteur et déclenchant une interruption.

Les six autres signaux sont acquis à l'aide du port B. Quatres entrées de ce port (RB4 → RB7) permettent de déclencher une interruption lors d'un changement d'état de ces mêmes entrées. La différence avec les entrées Ccp1 et Ccp2 sont qu'ici, aucun compteur n'est arrêté. Il faut donc gérer l'arrêt des compteurs à l'intérieur de l'interruption. Quatres signaux pourraient donc être connectés sur les pins RB4 → RB7. Mais il resterait deux signaux PWM à acquérir sans possibilité d'utiliser une interruption. Cela nous obligerait à faire du pooling sur deux entrées. Ce n'est pas acceptable pour les raisons citées précédemment. La solution trouvée est la suivante : deux signaux PWM sont connectés respectivement sur RB6 et RB7 et sont ainsi acquis.
A partir des quatre signaux restant à acquérir, on en génère 2 en connectant ces 4 signaux 2 à 2 sur un ou exclusif. Ces deux signaux ainsi créés sont connectés sur les entrées déclenchant une interruption : RB5 et RB4. Les 4 signaux sont aussi connectés sur les pin RB0 → RB4. Les pins RB0/RB1 ou RB2/RB3 sont alors lues et analysés à chaque fois qu'une interruption est déclenchée par RB4 ou RB5 respectivement .

3.1.3 Analyse des données acquises

La première figure (16) est un nuage de points correspondant aux données envoyées par le microcontrôleur à Matlab. Ce sont des données brutes représentant les temps d'état haut et bas (respectivement Tu et Td sur la doccumentation des accéléromètres) chronométrés des 8 signaux pwm générés par les 4 capteurs ADXL202.

En additionnant les temps Tu et Td de chaque voie, on obtient 8 différents temps de cycles identiques 2 à deux comme on le voit sur la figure (17). Ces 4 différents temps correspondent aux 4 capteurs utilisés. Les temps de cycle des deux voies d'un même capteur sont identiques. Cela peut être dû à une faible variation de résistivité des quatre résistances utilisées par chacun des capteurs pour fixer le temps cyclique. Ou cela peut être intrinsèque aux capteurs.

Sans calibration des capteurs, le calcul de G théorique donne les résultats présentés sur la figure (18). Les pics ou points excentrés observés sur ces figures sont dus aux erreurs de

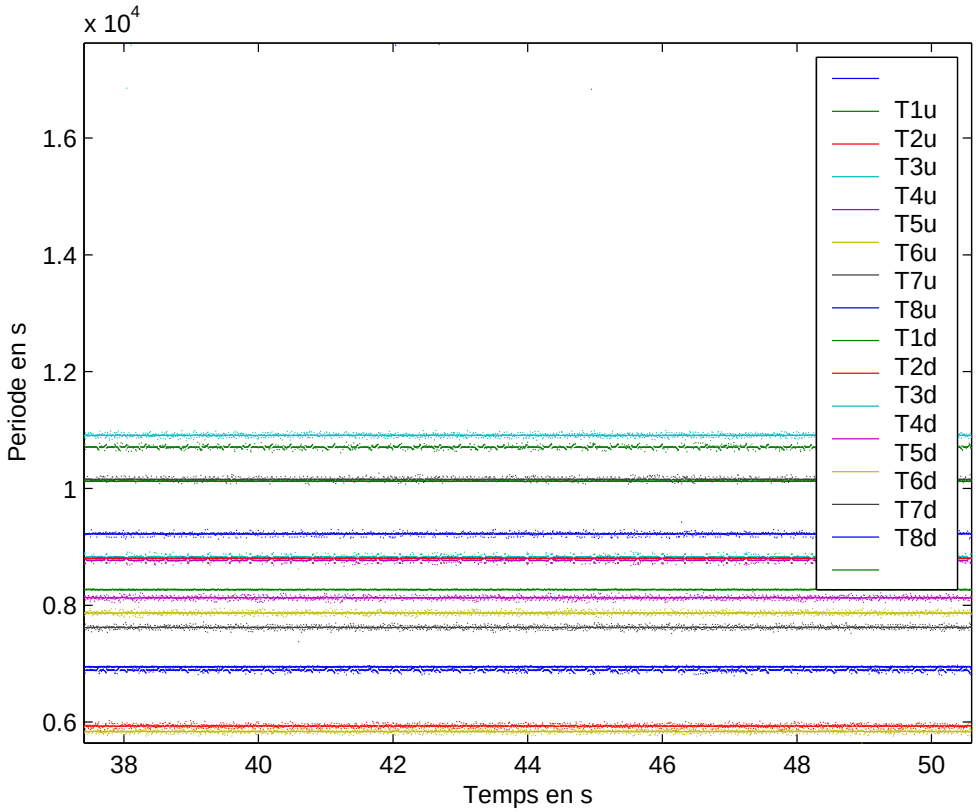

FIG. 16 – *Données acquises*

chronométrages des signaux comme cela est expliqué par la suite. Le nombre de ces erreurs est pour les différentes voies :

$$0 \quad 0 \quad 2 \quad 3 \quad 3 \quad 0 \quad 2 \quad 3 \tag{11}$$

La variance de G

$$s =< \sqrt{\sum (G_i - < G_i >)^2} >$$

donne pour chacune des 8 voies :

$$0.0018 \quad 0.0016 \quad 0.0398 \quad 0.0422 \quad 0.0127 \quad 0.0090 \quad 0.0401 \quad 0.0576 \tag{12}$$

En supprimant ces erreurs de chronométrages, on obtient une variance s :

$$0.0018 \quad 0.0016 \quad 0.0091 \quad 0.0089 \quad 0.0091 \quad 0.0090 \quad 0.0084 \quad 0.0088 \tag{13}$$

Les deux premières voies ne comprennent aucune erreur de chronométrage et leur variance est 4,5 fois plus faible que les voies 3 à 8 Ces deux voies sont meilleures que les autres car elles utilisent le module matériel du microcontrôleur conçu pour chronométrer des signaux PWM. Il n'y a pas d'erreur de lecture sur ces voies car le microcontrôleur ne peut pas laisser passer une transition du signal PWM. Les autres voies utilisent simplement les interruptions sur changement d'état sur quatre entrées du PIC. Les voies 5 et 6 sont directement connectées sur 2 de ces interruptions. C'est pourquoi les erreurs de lecture sont plus faibles. La voie 5 comprend 3 erreurs probablement dues à un faux contact. Les voies 3-4 et 7-8 comprennent plus d'erreur car elles utilisent un xor. Si les voies 3 et 4 ou 7 et 8 ont une transition au même

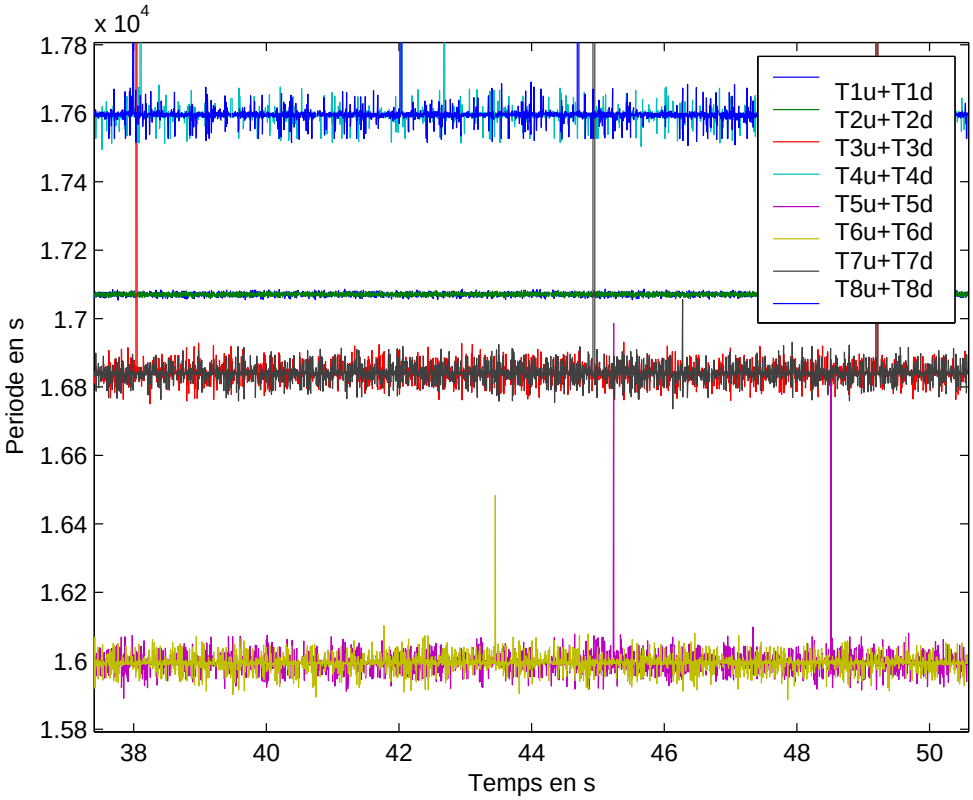

FIG. 17 – *Temps des Cycles PWM des 4 capteurs ADXL*

instant, le résultat du xor ne change pas. Le microcontrôleur manque alors la transition. L'erreur de chronométrage génère alors un pic. La variance des voies 3 à 8 est plus importante car le chronomètre n'étant pas arrêté par le déclenchement de l'interruption, on l'arrête lorsque l'on entre dans la routine d'interruption. Or si l'interruption se déclenche alors qu'une autre interruption est en train d'être traitée, son traitement est décalé dans le temps. D'où une petite erreur de chronométrage. Dans la section sur l'expérimentation du pilote automatique, on décrit une méthode d'acquisition afin que ces deux voies aient une variance aussi faible que les deux premières voies.

On connecte sur un xor les deux signaux d'un même capteur. Les Temps de l'état haut des deux voies étant centrés sur leur centre respectif, on obtient un pic lorsque les deux voies envoient une même donnée. Un essai de connexion sur un xor de deux voies provenant de différents composants accéléromètres montre que l'on obtient plus d'erreur du fait de la différence des périodes des cycles des capteurs (cf fig **??**).

On calibre les 8 axes en mesurant l'accélération due à la gravité dans les deux sens et pour chaque axe. La figure 19 montre un exemple de calibration. La précision obtenue ici est très bonne.

3.1.4 Resultats avec 4 accelerometres

Les résultats sont visualisés dans simulink grâce à un horizon artificiel (fig.20). On ne dispose pas de système permettant de mesurer l'angle réel de la croix réalisé avec l'horizontal. Cependant, sur l'axe de roulis, on constate que l'axe transversal de la croix est parfaitement

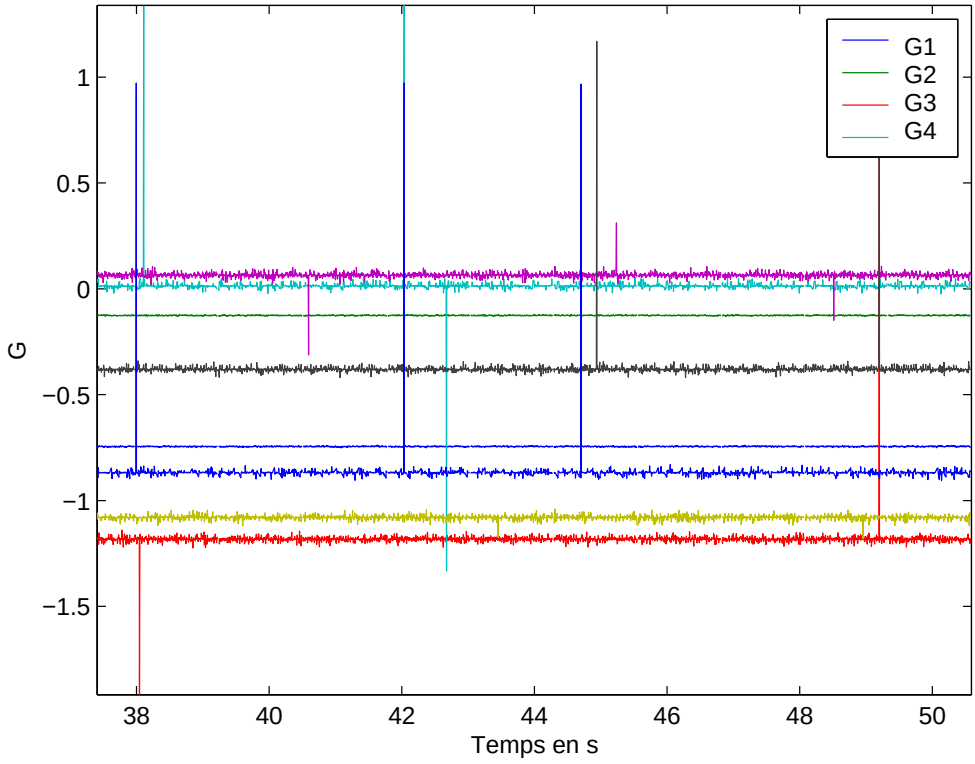

FIG. 18 – *G calculé Theorique*

parallèle avec l'image de l'horizon artificiel

On constate en traçant les courbes de l'évolution des angles que pour des changements assez rapides de position de la croix, les erreurs dues aux accélérations dynamiques mesurées sont très petites. On peut donc utiliser ce dispositif tel quel dans un avion radioguidé dont les accélérations angulaires sont petites. Cependant, les capteurs étant placé sur des barres en bois rigides, l'expérimentation en prend pas en compte les modifications géométriques de l'aile en vole. En effet, lors d'un vol, les ailes sous l'action de la portance se soulèvent. Les axes des différents capteurs sont alors modifiés et les données obtenues faussées.

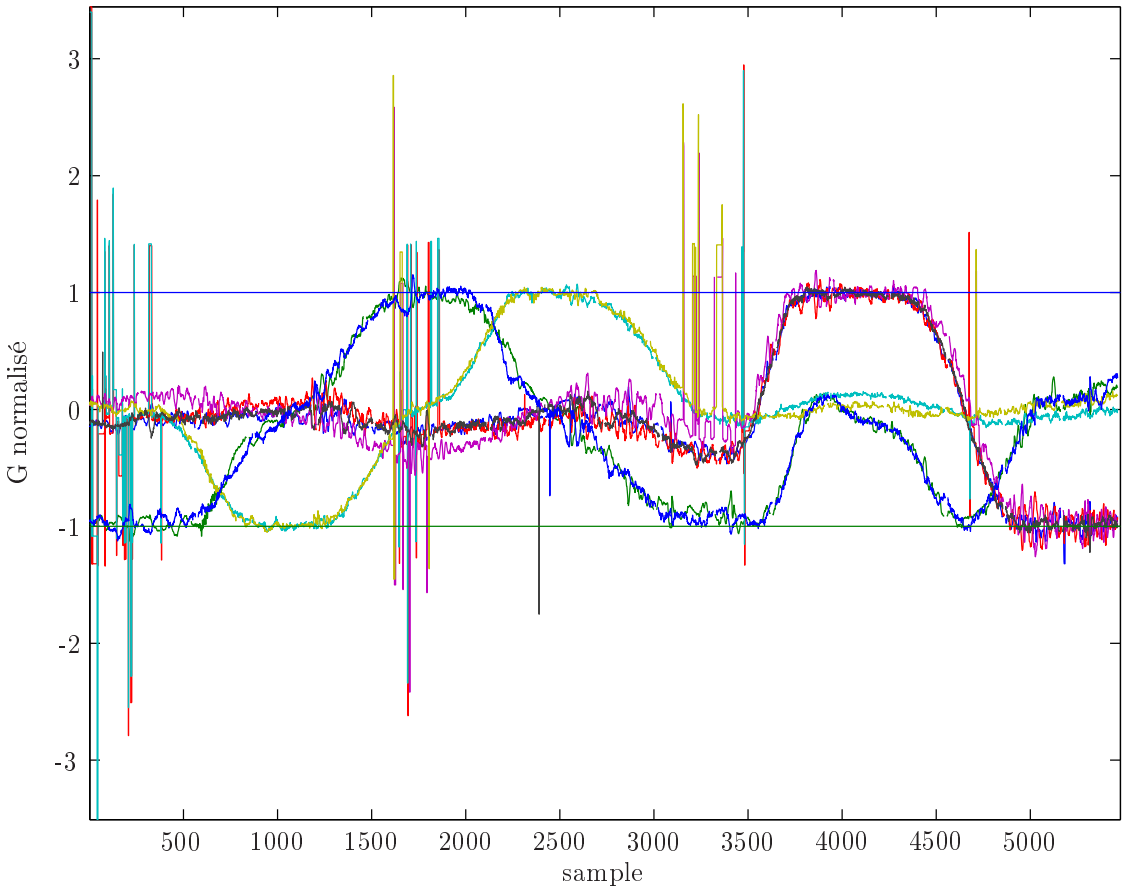

FIG. 19 – *Exemple de calibartion des 8 axes*

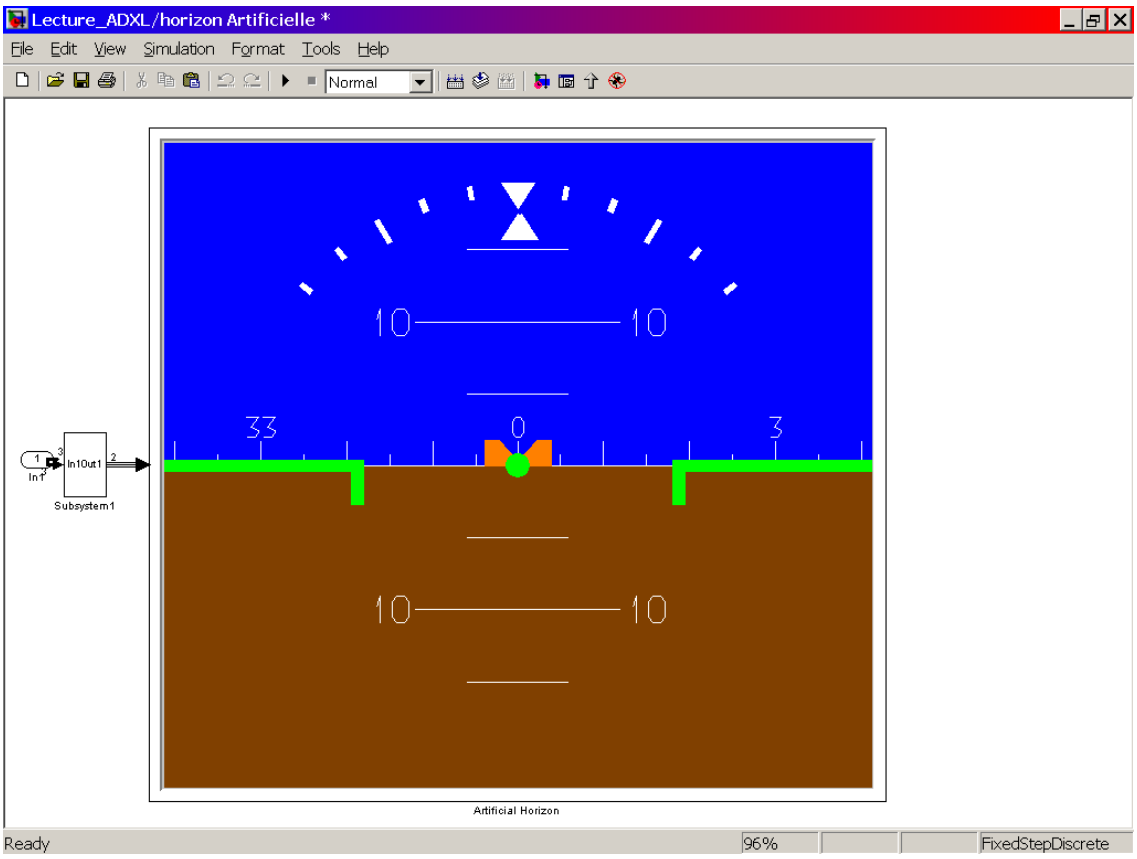

FIG. 20 – *Horizon Artificielle*

3.2 Pilote Automatique

3.2.1 Calculs implantés et mode d'emploie

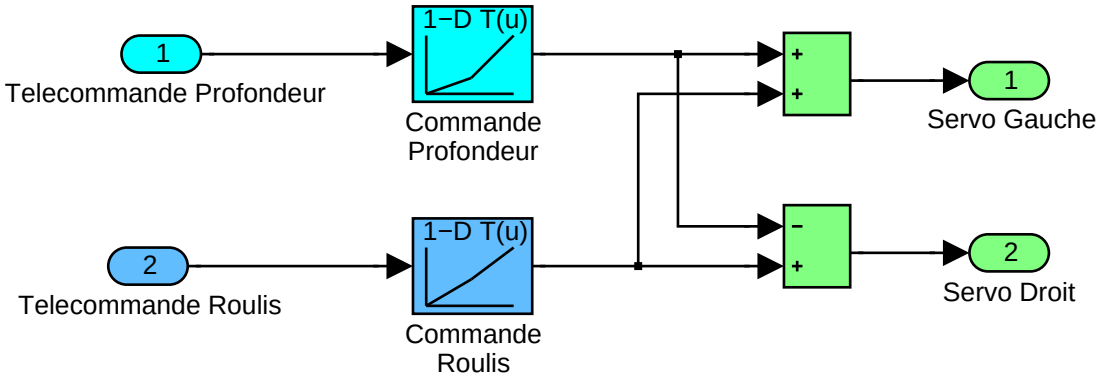

FIG. 21 – *mixeur simple*

La première fonction à réaliser est celle de mixage. Un simple mixeur avec les commandes exponentielles est présenté sur la fig **??**. La commande de chaque axe est modifiée une fonction. On ne définit pas cette fonction à l'aide d'un exponentiel car ceci utiliserait trop de place dans le pic et engendrerait des erreurs d'arrondis de calculs. La fonction est donc définit à l'aide de quatres segment de droite. Deux dans la partie positive et deux dans la partie négative (seul la partie positive est représenté sur le schéma 21). Celles-ci sont configurables par le pilote sur le terrain. L'intérêt est double : d'une part améliorer le confort de pilotage en définissant une courbes proche d'une exponentiel. Ainsi, les commandes sont moins sensibles près du neutre et plus sensible aux extrémités permettant un pilotage fin pouvant aussi être un peu plus sportif. Les débattements positif et négatif des ailerons peuvent être différents lors d'une mise en virage ... Le second intérêt est de corriger les éventuelles dissymétries de la tringlerie de l'aile.

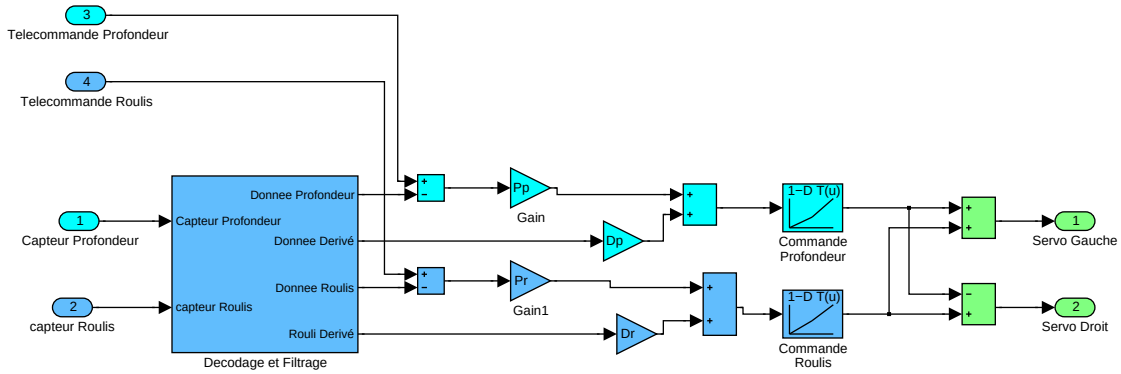

FIG. 22 – *CorrectionbProportionnel Derivative*

Pour stabiliser l'aile, la commande en amont du mixeur est modifié. Le schéma 22 montre le calcul effectué. C'est un simple correcteur Propotionnel dérivatif. L'angle de référence étant fixé par la télécommande. Le terme dérivatif est calculé uniquement à partir des données

des capteurs. Cela afin d'éviter qu'une commande petite mais brusque sur la télécommande entraîne une grande correction.

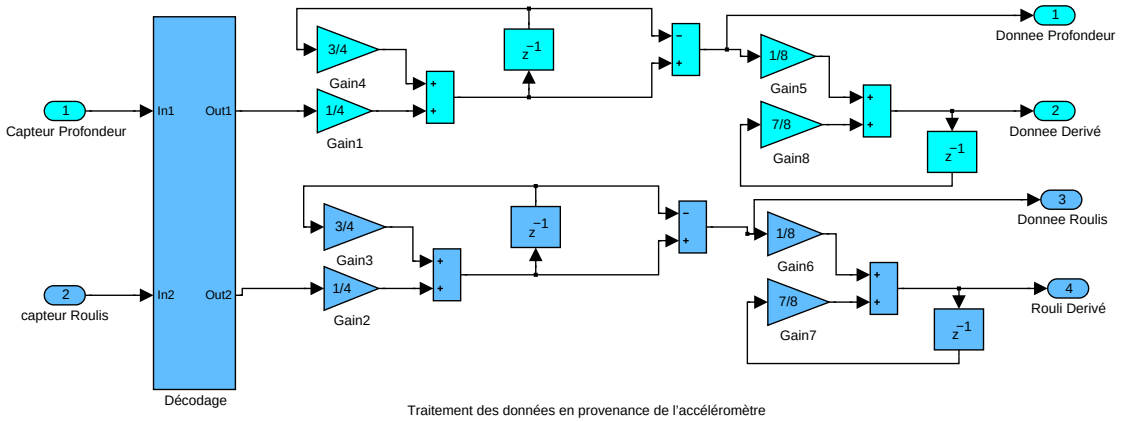

FIG. 23 – *Decodage et Filtrage*

Les données sont préalablement filtrées évitant ainsi que le bruit des capteurs se répercute sur les servos. Ceux-ci se mettraient alors à vibrer, consommant alors beaucoup d'énergie et parasitant le montage électronique. Le schéma des filtres utilisé est montré dans la figure 23. On utilise que des IIR. Ceux-ci ont l'avantage d'être implanté très facilement et de nécessiter très peu de variable, économisant ainsi la mémoire RAM. Des simulations de filtrage sur les données recueillies dans matlab ont permis de déterminer les coefficients optimaux. L'inconvénient majeur de ces filtres reste le temps de latence qu'ils engendrent.

Les données de la télécommande ne sont pas filtrées car leur bruit est très faible. Cependant, deux variables conservent une copie des données filtrées de la télécommande. Lorsque le signal de la télécommande est parasité, ces variables ne sont plus mises à jour et remplacent la donnée de la télécommande. Comme la détection du parasitage du signal n'est pas instantanée, cela permet de ne pas bloquer ces valeurs uniquement sur la dernière donnée reçue qui peut être faussée. La détection du parasitage des données de la télécommande se fait en comparant celle-ci à un gabarit.

3.2.2 Emission - Reception des signaux

Les deux données provenant du récepteur et les deux données provenant des capteurs sont modulés en largeur d'impulsion. La commande des servos moteurs se fait également en modulations d'impulsion. Il faut donc pouvoir chronométrés tous les signaux y compris lorsque ceux-ci se chevauchent. Il n'est donc pas possible d'utiliser des boucles pour chronométrer les différents signaux. La précision nécessaire pour que les servos ne tremblent pas est de l'ordre du temps d'exécution d'une instruction.

Pour résoudre ce problème, on utilise les interruptions du PIC 16f628. Trois interruptions permettent de chronométrer les quatres signaux en entrée.

La première interruption est "CCP Interrupt". Celle-ci est sert à acquérir un signal modulé en largeur d'impulsion. Étant conçu pour cela, la précision obtenue est très bonne. Malheu-

reusement, le microcontrôleur ne dispose que d'une seule entrée comme celle-ci. Il faut donc utiliser d'autres astuces afin de chronométrer les trois signaux restants.

L'interruption "PORTB change interrupt" est déclenchée lorsque l'une des broches du port B change d'état. En relevant les temps auxquels les différentes broches changent d'état, il est possible de chronométrer un signal PWM. On configure le Timer 1 afin que ce dernier soit incrémenté à chaque cycle d'instruction. Le relevé de temps se fait alors par rapport au Timer 1. Ce système fonctionne, mais on observe un peu de bruit sur les signaux captés. Le problème actuel est que le PIC ne peut traiter qu'une interruption à la fois. En conséquence, si une interruption x se déclenche alors qu'une autre interruption y est en train d'être traité, le traitement l'interruption x est différé et s'exécute après la fin du traitement de l'interruption y. Lorsque l'interruption "PORTB change interrupt" est déclenchée alors qu'une autre interruption est en cours de traitement, le relevé du Timer 1 s'effectue un peu plus tard. Ceci engendre une erreur sur le chronométrage de l'impulsion en cours. Cette erreur est aussi présente sur l'impulsion suivante du fait que chaque temps relevé sépare deux impulsions. Afin de résoudre ce problème, chaque interruption vérifie à la fin de son traitement si une autre interruption a été déclenchée. Si c'est le cas, un flag est positionné permettant à l'interruption déclenchée en second de savoir que son traitement à été différé. Lorsqu'une interruption de lecture de signal PWM est différée, cette dernière ne met pas à jour les valeurs de chronométrage capturé. Deux signaux sont capturés avec cette interruption.

Le quatrième signal est capturé à l'aide de l'interruption "External Interrupt RB0/INT". Cette interruption fonctionne de manière similaire à l'interruption du port B.

Pour l'émission sur les servos, le problème est différent. Admettons que l'on veuille émettre une impulsion d'une durée T, ou T est exprimé en temps de cycle d'instruction. Le timer 2 du PIC permet de déclencher une interruption lorsque ce timer atteint la valeur de la variable T2CON. Le timer 2 est un timer 8 bits. Les impulsions les plus courtes à émettre ont une durée de 1000 cycles instruction. Donc le timer 2 est trop petit. On peut utiliser un pre scaler afin de ralentir l'incrémental du timer, ce qui permettrait de convertir le temps t sur moins de 8 bits. Cependant, la perte de résolution est non négligeable. Il existe aussi un post-scaler qui permet de ne déclencher l'interruption lorsque le timer 2 a atteint n fois la valeur de T2CON, ou n est la valeur du post-scaler. n est compris entre 0 et 15. Lorsque l'interruption se déclenche, on positionne la broche de sortie à 1 ou à 0 suivant que l'on veut générer une impulsion haute ou basse. Si T est plus grand que $255 * 16$, on soustrait à T cette valeur, et on place T2CON à 255 et n a 15. À la prochaine occurrence de cette interruption, on recommence le procédé jusqu'à ce que T soit inférieur à 255. n pouvant prendre des valeurs inférieures à 15. Alors, au déclenchement suivant de l'interruption, on arrête l'impulsion en cours d'émission et on commence la suivante.

Comme pour le cas de la lecture des signaux, une erreur sur le signal émit apparaît lorsqu'une interruption se déclenche juste avant que l'interruption signalant la fin de l'émission du signal soit déclenchée. Le traitement de cette seconde interruption étant alors différé, le signal modulé est faussé. Contrairement au cas de la lecture, il n'est pas possible d'annuler une impulsion émise. La solution consiste donc à interdire toutes les interruptions lorsque T a une valeur inférieure à un seuil fixé. Il faut aussi veiller ne pas chargé dans T2CON de valeurs trop petites car lors du déclenchement de l'interruption, le compteur est remis à zéro, et se met à compter pendant l'exécution même de l'interruption. Si on place dans T2CON une valeur inférieure au timer, le résultat sera biaisé. Dans le programme, T2CON ne reçoit jamais de

valeur inférieure à 128 et les interruptions sont toutes inhibés sauf celle du timer 2 lorsque le temps t restant à émettre est inférieur à 512. Il faut prévenir avec des flags les interruptions dont le traitement aura été différé à cause de cette coupure. On utilise pour cela le même flag utilise pour indiquer que deux interruptions ont été déclenchées au même moment. (voir code source B.1 p.38)

3.2.3 Montage Electronique

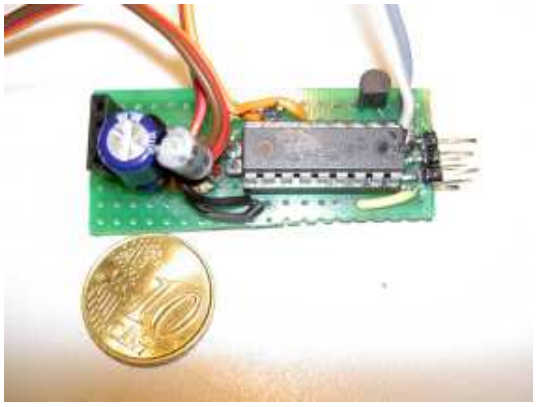

FIG. 24 – *Montage Electronique : Calculateur*

FIG. 25 – *Montage Electronique : Accelerometre ADXL202*

Le montage électronique est très simple. Le PIC16f628 n'a besoin d'aucun composant externe pour fonctionner (fig.24). Il utilise un circuit interne pour générer son horloge. La fréquence de celle-ci est de 4Mhz. Il est alimenté en 0-5v par les broches d'alimentation vdd et vss. Les sorties de commande des servos sont connectées directement aux servos via les broches Ra6 et Ra7. Le récepteur envoie ces signaux sur les broches Rb2 et Rb3 et les deux signaux des deux accéléromètres sont capturés via les broches Rb4 et Rb5. Un condensateur chimique de $100\mu F$ est placé sur l'alimentation du PIC. Le capteur ADXL202 n'a pas la même alimentation que le pic et les servos. En effet, les servos engendrent des parasites sur les lignes

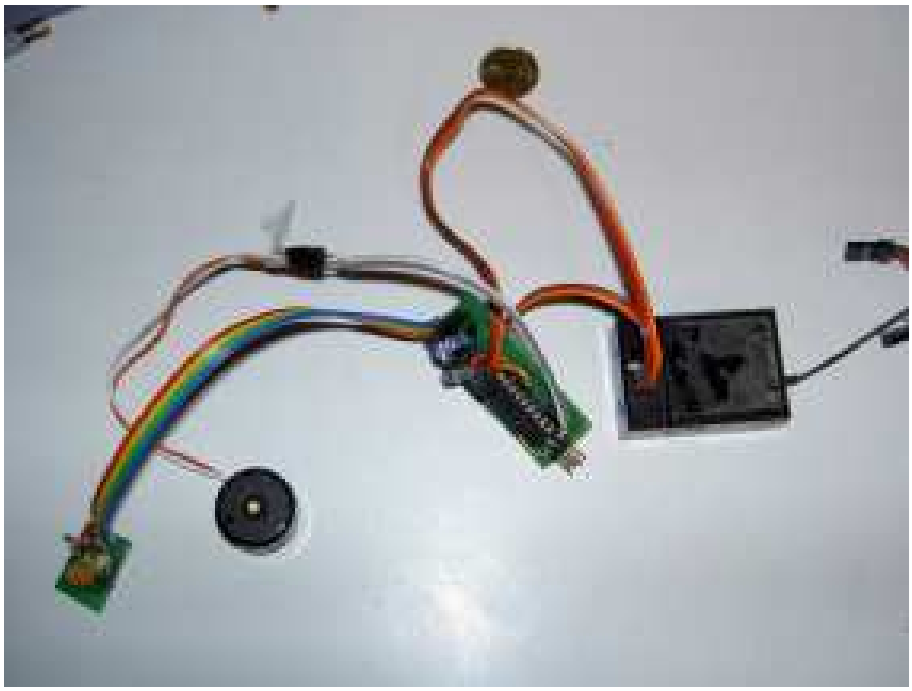

FIG. 26 – *Montage Electronique : Système complet*

d'alimentation qui perturbent de manière conséquente le fonctionnement de l'accéléromètre. Celui-ci est donc alimenté par la même alimentation, mais filtré à l'aide d'une résistance de $4{,}7K\Omega$ en série et d'une capacité chimique de $220\mu F$ en parallèle. Le filtre anti-repliement du capteur est constitué de deux capacités céramiques de $.1\mu F$. La résistance définissant la durée des impulsions PWM est de $1.5M\Omega$ (fig.25) (cf équations 4 p.21). Enfin, le beeper étant connecté sur une sortie ayant une impedance importante, on utilise un transistor afin d'adapter l'entrée du beeper à cette sortie analogique (fig.26).

3.2.4 Partie modélisme

Nous avons choisi L'avion IXIR dont les plans sont sur Internet à l'adresse http://membres.lycos.fr/afstorm/ C'est une aile volante conçue en dépron d'une envergure de 1,25m pesant approximativement 400g (fig.1 p1). Il a été choisi pour ses caractéristiques suivantes :

- construction rapide
- grande envergure (plus de 1m) permettant une certaine souplesse dans la création d'un système embarqué point de vue poids et volume.
- bonne résistance aux atterrissages brusques

3.2.5 Premières expérimentations

Les premières expérimentations du pilote automatique ont été effectuées alors que celui-ci ne disposait pas encore du dispositif permettant de couper momentanément l'asservissement. Il a donc été difficile de lancer l'aile fortement. Seul un correcteur proportionnel était alors implanté. Cependant, les quelques vols effectués ont montré que le concept fonctionnait. Le coefficient du correcteur proportionnel étant trop important sur l'axe de tangage, l'aile a effectué des ondulations autour de sa position d'équilibre. La correction du roulis était elle lente, mais fonctionnait bien.

4 Conclusion

Ce projet montre qu'il est possible de réaliser un système temps réel embarqué avec des composants relativement simples et répandus. Les résultats obtenus lors des premiers vols prouvent que le concept est bon : Il est possible de stabiliser un modèle réduit de planneur à l'aide d'accéléromètres. Pour améliorer les résultats, il faudrait embarquer plus de capteurs dans l'avion. Que ce soit des accéléromètres judicieusement placés, des gyromètres ou encore des magnétomètres. Le microcontrôleur (PIC 16f628) étant saturé par le programme actuel, il est nécessaire d'en utiliser un plus gros comme le PIC16f877 permettant d'effectuer des calculs plus poussés avec les capteurs ajoutés.

Un rapport de projet de 3^e année traitant de la stabilisation d'un modèle réduit à l'aide de gyromètres se trouvant à l'adresse http://lubink.free.fr à coté de ce présent rapport qu'il pourra compléter.

A Filtre de Kalman

On étudie ici le fonctionnement d'un filtre de Kalman. Celui-ci semble bien adapté au problème. On dispose du modèle mathématique de l'avion. Il est donc possible de prévoir son évolution future. A l'aide des capteurs, on peut aussi retrouver l'état de l'avion. Le filtre de Kalmlan permet de pondérer ces deux calculs de manière optimale comme expliqué ci après.

A.1 Fonctionnement d'un filtre de Kalman

Un filtre de Kalman nécessite de connaître l'équation de l'évolution du système. A partir de cette équation, il tente de différencier les données provenant des capteurs du bruit permettant ainsi d'estimer de manière optimale l'état du système.

A.1.1 Sur un système linéaire

On a le système d'équation suivant :

$$x(m) = A(m,m-1)x(m-1) + v_1(m) \quad (14)$$
$$y(m) = H(m)x(m) + v_2(m) \quad (15)$$

où

$x(m)$ est l'état du système de dimension P

$A(m,m-1)$ est la matrice P*P d'évolution du système

$y(m)$ est la mesure provenant des capteurs de dimension M

$H(m)$ est la matrice M*P reliant l'état à la mesure

$v_1(m)$ est le vecteur de bruit de l'évolution du système. Il représente l'incertitude du système. Si celui-ci est nul, les mesures des capteurs n'influenceront pas l'estimation de l'état car le filtre fera entièrement confiance à l'équation d'évolution

$v_2(m)$ est le vecteur de bruit sur les mesures. Si celui-ci est nul, les mesures seront considérées comme exactes et le filtre estimera l'état $x(m)$ à partir de la seule estimation $y(m)$.

Le filtre de Kalman consiste à estimer $\hat{x}(m)$ en utilisant l'état estimé précédent $\hat{x}(m-1)$ ainsi que l'ensemble des mesures $[y(0); y(1)...y(m)]$.

Partant de l'équation (15), on obtient pour l'espérance mathématique de $\hat{y}(m)$:

$$< \hat{y}(m) > = < H(m)\hat{x}(m) + v_2(m) > \quad (16)$$

le bruit v_2 étant blanc et décorélé avec l'état, on a :

$$< \hat{y}(m) > = < H(m)\hat{x}(m) > + < v_2(m) > \quad (17)$$
$$< \hat{y}(m) > = < H(m)\hat{x}(m) > \quad (18)$$

$\hat{x}(m)$ et $\hat{y}(m)$ sont reliés par la matrice H.Estimer $\hat{x}(m)$ de mainère optimale revient à estimer $\hat{y}(m)$ de manière optimale.

Le filtre de Kalman part de la supposition que la mesure estimée $\hat{y}(m)$ est optimale. l'erreur de prédiction $\tilde{y}(m)$ est alors défini comme

$$\tilde{y}(m) = y(m) - \hat{y}(m|m-1) \tag{19}$$

si $\hat{y}(m)$ est optimal, alors l'erreur $\tilde{y}(m)$ contient la partie non prédictible de y. $\hat{y}(m|m-1)$ étant estimée par une combinaison linéaire des observations précédentes $[y(0); y(1)...y(m-1)]$, $\hat{y}(m|m-1)$ est le projeté orthogonal de $y(m)$ dans cet espace. $\tilde{y}(m)$ est donc perpendiculaire à $\hat{y}(m|m-1)$:

$$< \tilde{y}(m)\hat{y}^T(m|m-k) > = 0, \qquad k > 0 \tag{20}$$

La partie non prédictible $\tilde{y}(m)$ de $y(m)$ contient le bruit ainsi que l'innovation apportée par la mesure. On l'appelle vecteur d'innovation.

L'espace vectoriel $[y(0); y(1)...y(m)]$ augmenté par le vecteur $\tilde{y}(m)$ et l'hypothèse $\tilde{y}(m)$ perpendiculaire à $\hat{y}(m|m-1)$ nous donne par récursion:

$$< \tilde{y}(m)\tilde{y}^T(m-k) > = 0, \qquad k > 0 \tag{21}$$

Le filtre de Kalman estime $\hat{x}(m+1|m)$ à partir des deux équations (14) et (15) en pondérant leur importance respective suivant l'équation suivante:

$$\hat{x}(m+1|m) \quad = \quad A(m+1,m)\hat{x}(m|m-1) + K(m)\tilde{y}(m) \tag{22}$$

où la matrice K(m) de taille P*M est appelée gain de Kalman. La première partie du second membre fait apparaître l'équation d'évolution du système $A(m+1,m)\hat{x}(m|m-1)$. La seconde partie introduit les données des capteurs dans l'équation d'estimation. $\tilde{y}(m)$ étant relié à la mesure $y(m)$ par l'équation (19). Pour déterminer la matrice de pondération K, on va utiliser les hypothèses précédentes.

On multiplie l'équation par $\tilde{y}^T(m)$ et on calcule l'espérance mathématique :

$$< \hat{x}(m+1|m)\tilde{y}^T(m) >=< A(m+1,m)\hat{x}(m|m-1)\tilde{y}^T(m) > + < K(m)\tilde{y}(m)\tilde{y}^T(m) > \tag{23}$$

En utilisant (15) puis (20), la première partie du second membre devient :

$$\begin{aligned} < A(m+1,m)\hat{x}(m|m-1)\tilde{y}^T(m) > \quad &= \quad < A(m+1,m)H^{-1}\hat{y}(m|m-1)\tilde{y}^T(m) > & \text{(24a)} \\ &= \quad 0 & \text{(24b)} \end{aligned}$$

d'où

$$\begin{aligned} K(m) \quad &= \quad < \hat{x}(m+1|m)\tilde{y}^T(m) >< \tilde{y}(m)\tilde{y}^T(m) >^{-1} & \text{(25a)} \\ K(m) \quad &= \quad < \hat{x}(m+1|m)\tilde{y}^T(m) > \Sigma_{\tilde{y}\tilde{y}}^{-1}(m) & \text{(25b)} \end{aligned}$$

avec $\Sigma_{\tilde{y}\tilde{y}}^{-1}(m)$ la matrice M*M de covariance de $\tilde{y}(m)$.

Le calcul de $\hat{x}(m+1|m)$ nécessite la connaissance de $K(m)$. Il faut trouver une autre relation partant de l'équation (25).
On définit $\tilde{x}(m)$:

$$\begin{aligned} \tilde{x}(m) &= x(m) - \hat{x}(m|m-1) && (26)\\ \hat{x}(m+1|m) &= x(m+1) - \tilde{x}(m+1) && (27) \end{aligned}$$

(25) et (27) donnent

$$K(m) = < \big(x(m+1) - \tilde{x}(m+1)\big)\tilde{y}^T(m) > \Sigma^{-1}_{\tilde{y}\tilde{y}}(m) \quad (28)$$

avec (21), (28) s'écrit :

$$K(m) = < x(m+1)\tilde{y}^T(m) > \Sigma^{-1}_{\tilde{y}\tilde{y}}(m) \quad (29)$$

soit avec (14) puis (15)

$$\begin{aligned} K(m) &= < \big(A(m+1,m)x(m) + v_1(m)\Big)\tilde{y}^T(m) > \Sigma^{-1}_{\tilde{y}\tilde{y}}(m) && (30)\\ K(m) &= < \Big(A(m+1,m)H^{-1}(m)y(m)\Big)\tilde{y}^T(m) > \Sigma^{-1}_{\tilde{y}\tilde{y}}(m) && (31) \end{aligned}$$

Avec (19), $K(m)$ s'écrit :

$$K(m) = < \Big(A(m+1,m)H^{-1}(m)\tilde{y}(m) - \hat{y}(m)\Big)\tilde{y}^T(m) > \Sigma^{-1}_{\tilde{y}\tilde{y}}(m) \quad (32)$$

avec (20) on obtient finalement :

$$K(m) = < A(m+1,m)H^{-1}(m)\tilde{y}(m)\tilde{y}^T(m) > \Sigma^{-1}_{\tilde{y}\tilde{y}}(m) \quad (33)$$

$\tilde{y}(m)$ et $\tilde{x}(m)$ sont reliés par la la matrice H. A partir de (15) et (19) on a :

$$\begin{aligned} \tilde{y}(m) &= y(m) - \hat{y}(m) && (34a)\\ &= H(m)x(m+1) + v_2(m) - H(m)\hat{x}(m) && (34b) \end{aligned}$$

$$\begin{aligned} < \tilde{y}(m) > &= < H(m)x(m) > + < v_2(m) > - < H(m)\hat{x}(m) > && (35a)\\ &= < H(m)\Big(x(m) - \hat{x}(m)\Big) > + < v_2(m) > && (35b)\\ &= H(m) < \tilde{x}(m) > + < v_2(m) > && (35c) \end{aligned}$$

$$< \tilde{x}(m) > = H^{-1}(m) < \tilde{y}(m) - v_2(m) > \quad (36a)$$

On a alors en remplaçant dans l'expression de $K(m)$ (33) $\tilde{y}(m)$ par $H(m)\tilde{x}(m)$

$$\begin{aligned} K(m) &= < A(m+1,m)H^{-1}(m)H(m)\tilde{x}(m)\Big(H(m)\tilde{x}(m)\Big)^T > \Sigma^{-1}_{\tilde{y}\tilde{y}}(m) && (37)\\ K(m) &= < A(m+1,m)\tilde{x}(m)\tilde{x}^T(m)H^T(m) > \Sigma^{-1}_{\tilde{y}\tilde{y}}(m) && (38) \end{aligned}$$

On exprime $\Sigma^{-1}_{\tilde{y}\tilde{y}}(m)$ en fonction de $\tilde{x}$ en partant de (35). $v_2(m)$ étant décorrélé avec $\tilde{x}(m)$:

$$\begin{aligned} \Sigma^{-1}_{\tilde{y}\tilde{y}}(m) &= \tilde{y}(m)\tilde{y}^T(m) && (39a)\\ &= < H(m)\tilde{x}(m)\tilde{x}^T(m)H^T(m) > + < v_2(m)v_2^T(m) > && (39b)\\ &= H(m)\Sigma_{\tilde{x}\tilde{x}}(m)H^T(m) + \Sigma_{v_2v_2}(m) && (39c) \end{aligned}$$

$K(m)$ s'exprime en fonction de la variance de l'erreur de prédiction $\Sigma_{\tilde{x}\tilde{x}}(m)$. Il reste à déterminer une expression permettant de la calculer. On exprime $\tilde{x}(m|m-1)$ en fonction de $\tilde{x}(m-1|m-2)$:

$$\begin{aligned}\tilde{x}(m|m-1) &= x(m) - \hat{x}(m|m-1) \\ &= A(m,m-1)x(m-1) + v_1(m) \\ &\quad -\Big(A(m,m-1)\hat{x}(m-1|m-2) + K(m-1)\tilde{y}(m-1|m-2)\Big) \\ &= A(m,m-1)\Big(x(m-1) - \hat{x}(m-1|m-2)\Big) \\ &\quad +v_1(m) - K(m-1)\tilde{y}(m-1|m-2)\end{aligned} \tag{40}$$

$$K(m-1)\tilde{y}(m-1|m-2) = K(m-1)\Big(H(m-1)\tilde{x}(m-1|m-2) + v_2(m-1)\Big) \tag{41}$$

soit :

$$\begin{aligned}\tilde{x}(m|m-1) &= \Big(A(m,m-1) - K(m-1)H(m-1)\Big)\tilde{x}(m-1) \\ &\quad +v_1(m) - K(m-1)v_2(m-1)\end{aligned} \tag{42}$$

On en déduit la matrice $\Sigma_{\tilde{x}\tilde{x}}(m)$:

$$\begin{aligned}\Sigma_{\tilde{x}\tilde{x}}(m) &= \tilde{x}(m|m-1)\ \tilde{x}^T(m|m-1) \\ &= L(m)\ \Sigma_{\tilde{x}\tilde{x}}(m-1)\ L(m)^T + \Sigma_{v_1v_1}(m) \\ &\quad +K(m-1)\ \Sigma_{v_2v_2}(m-1)\ K^T(m-1)\end{aligned} \tag{43}$$

avec

$$L(m) = A(m,m-1) - K(m-1)H(m-1) \tag{44}$$

A.1.2 Sur un système non linéaire : Filtre de Kalman Etendu

Le filtre de Kalman vue précédemment s'applique lorsque le système est décrit par les équations (14) et (15). Ces équations impliquent modèle ainsi que les observations soient de type linéaire. On peut avoir besoin de modéliser un système non linéaire. Cela est le cas dans notre centrale inertiel puisque les calculs font apparaître des fonctions trigonométriques.

Nous nous intéressons maintenant à un modèle non linéaire d'observation et d'évolution du système. Les équations régissant un tel système s'écrivent ainsi :

$$x(n+1) = F\big(n,x(n)\big) + v_1(n) \tag{45}$$

$$y(n) = H\big(n,x(n)\big) + v_2(n) \tag{46}$$

On linéarise F et H au point la dernière approximation :

$$F(n+1,n) = \left.\frac{\delta F\big(n,x(n)\big)}{\delta x}\right|_{x=\hat{x}(n|n-1)} \tag{47}$$

$$H(n+1,n) = \left.\frac{\delta H\big(n,x(n)\big)}{\delta x}\right|_{x=\hat{x}(n|n-1)} \tag{48}$$

En développant la formule de taylor au premier ordre, on obtient :

$$F\big(n,x(n)\big) \simeq F\big(n,\hat{x}(n|n-1)\big) + F(n+1,n)\Big(x(n) - \hat{x}(n|n-1)\Big) \tag{49}$$

$$H\big(n,x(n)\big) \simeq H\big(n,\hat{x}(n|n-1)\big) + H(n+1,n)\Big(x(n) - \hat{x}(n|n-1)\Big) \tag{50}$$

Les équations (45) et (46) peuvent alors s'approximer :

$$x(n+1) = F\big(n,\hat{x}(n|n-1)\big) + F(n+1,n)\Big(x(n) - \hat{x}(n|n-1)\Big) + v_1(n) \quad (51)$$

$$y(n) = H\big(n,\hat{x}(n|n-1)\big) + H(n+1,n)\Big(x(n) - \hat{x}(n|n-1)\Big) + v_2(n) \quad (52)$$

En posant :

$$d(n) = F\big(n,\hat{x}(n|n-1)\big) - F(n+1,n)\hat{x}(n|n-1) \quad (53)$$

$$\bar{y}(n) = y(n) - H\big(n,\hat{x}(n|n-1)\big) + H(n+1,n)\hat{x}(n|n-1) \quad (54)$$

$d(n)$ et $\bar{y}(n)$ sont connus à l'instant n. En réécrivant les équations (51) et (52) ainsi :

$$x(n+1) = F(n+1,n)x(n) + v_1(n) + d(n) \quad (55)$$

$$\bar{y}(n) = H(n+1,n)x(n) + v_2(n) \quad (56)$$

On obtient les propriétés que les équations du filtre de Kalman linéaire (14 & 15). On peut utiliser les relations démontrés précédemment.

B Pilote automatique

B.1 Programme C implanté

Listing 1: *Mixer628.c*

identifierstyle

```
#define pic16f628
#include <pic.h>
#include <RS232e.c>

//-------------------------------------------------------------------
//-DEFINE-----------------------------------------------------------
//-------------------------------------------------------------------


//--Routine de modification de bits--
#define testbit_on(data,bitno) ((data>>bitno)&0x01)
#define bit_set(var,bitno) ((var) |= 1 << (bitno))
#define bit_clr(var,bitno) ((var) &= ~(1 << (bitno)))


//--Donnee EEPROM : image de la structure mem ds l'EEPROM
//--16 bits : low-high
__EEPROM_DATA(0,0,0,0,0,0,128,0);
__EEPROM_DATA(64,128,0,64,128,0,64,128);
__EEPROM_DATA(0,64,128,0,64,128,0,64);
__EEPROM_DATA(128,0,64,128,0,64,0xFF,0xFF);

//--Configuration mode de fct du PIC
   CONFIG(INTIO & MCLRDIS & LVPDIS & BORDIS & WDTDIS & PWRTEN);

#define DeconteurInterImpulsion 20000;
//--Tps entre les trains d'impulsions emises sur les servos
//--On ne respecte pas la normes actuel
//--20ms entre fin emission sur servos2 et début émission servos droit
//--quelque soit la longueur de l'impulsion émise.
//--LE CYCLE N'EST DONC PAS DE 20ms

//-------------------------------------------------------------------
//--Variables globales-----------------------------------------------
//-------------------------------------------------------------------



//--Donnees Globales
unsigned char static  INDF @ 0x00;  // pointeur "hardware" du PIC
            // on tape dedans sans prévenir le compilateur
            // attention : erreur possible

unsigned char modeConfig; // Chronometre 3s sans signal puis 3s en butte
        // Global car utilise ds main() et ds interruption

unsigned int TemoinTeleEteinteCourte; // Si telecommande éteinte, Neutralise asservissement
        // permet décolage ( acceleration importante -> capteur faux )

```

```
50
51  // lors de la configuration des débattements des servos, la routine mixage doit
52  // réagir différement
53  bit TemoinConfigTrim;
54
55  int AsservissementProfondeur; // Valeurs remplaçant les commandes ds routine de mixage
56  int AsservissementRouli; // l'asservissement se fait à ce niveau
57
58  // Les signaux sont capturé à l'aide d'interruption -> Variables Globales
59  struct PWMx { // Structure utilisés à la capteur des signaux : Telecommande & Capteur
60    unsigned int up;
61    unsigned int down;
62  };
63  typedef struct PWMx PWM;
64
65  bank1 PWM PWM_1; // Telecommande
66  bank1 PWM PWM_2; // ADXL
67  bank1 PWM PWM_3; // ADXL
68  bank1 PWM PWM_4; // Telecommande
69
70  bit semaphorePWM1; // Temoins MAJ des valeurs
71  bit semaphorePWM2;
72  bit semaphorePWM3;
73  bit semaphorePWM4;
74
75  //Donnees lu par l'interruption, MAJ ds le prog principal
76  unsigned int ImpulsionG; //Impulsion à emettre sur servo G
77  unsigned int ImpulsionD; // idem sur servo D
78
79  // Emission en cours. Utilise seulement ds l'interruption, mais bits tjs global
80  bit EmittingPWMG;
81  bit EmittingPWMD;
82
83  // donnees concernant chaque voie de la telecommande
84  struct telecommandex {
85    int value; // Valeur utilisés ds calculs ( valeur réel ou copie de filtre )
86    unsigned int filtre ; // si donnee valide : filtrage
87    unsigned char HorsGabarit; // donnee valide ? ( seuil ...)
88  };
89  typedef struct telecommandex telecommande;
90
91  telecommande Rouli; // 2 instances pr 2 voies
92  telecommande Profondeur;
93
94  struct Accelerometrex {
95    int value;
96    int Derivative;
97    int zero;
98  };
99  typedef struct Accelerometrex Accelerometre;
100
101   bank1 Accelerometre ADXLTanguage;
102   bank1 Accelerometre ADXLRouli;
103
104 //--Valeurs du débatement maximal et intermédiaire pour 1 voie et sur "un sens"
105 //--4 instances sont nécessaire pour définir les 2 voies dans les 2 sens
106 struct Polyx {
107   int    max;   // stock les 2 position, divise par 4
108   signed char mid;    // de même : facteur 4
109    };
110 typedef struct Polyx Poly;
111 // point de vue capacité : la valeur max ne pose pas de problème
112 // [-32768 +32767] *4 -> [-131072 +131068]...
113 // La plage utilisé sera de l'ordre [-700 +700] (on dépasse les débattements standard)
114 // La valeur mid : intermédiaire peut définir un débatement standard complet d'un servo
115 // [-128 +127] * 4 -> [-512 +508] un débattmeent standard étant [-500 +500]
116
117
118
119 // Structure en RAM recopie INTEGRALEMENT dans l'EEPROM
120 // Chargé en RAM au démarrage et copié en ROM lors de modifications
121 // Contient la configuration des débattements ( commande "exponentiel")
122 // + la correction de la position zeros des servos
123 // + variables
124 // L'ordre des variables est importante :
125 // La fonction de configuration config_trim() utilise un pointeur pr parcourir
126 // les différentes valeurs (afin de réduire la taille du programme)
127 struct memx {
128
129 int trimG;
130 signed char reglage1;   // pr avoir la même structure que les polynomes
131 int trimD;
132 signed char reglage2;
133
134 Poly polyProfPosG;
135 Poly polyProfPosD;
136 Poly polyProfNegG;
137 Poly polyProfNegD;
138
139 Poly polyRouliPosG;
140 Poly polyRouliNegD;
141 Poly polyRouliPosD;
142 Poly polyRouliNegG;
143
144 } bank1 mem;
```

```
145
146
147
148 //-------------------------------------------------------------------------
149 //--fonctions------------------------------------------------------------
150 //-------------------------------------------------------------------------
151
152 //-------------------------------------------------------------------------
153 //--Emission Tension Variable sur Beeper---------------------------
154 //-------------------------------------------------------------------------
155 void EmetBeeper(unsigned char value) {
156   // value compris entre 0 et 17 inclus !
157
158   if (value > 16 || value == 0) {
159     VRCON = 0; // Etteint convertisseur N/A
160     TRISA2 = 0; // definit en sortie
161     if (value > 16) RA2 = 1;
162     else RA2 = 0;
163     return;
164   } else {
165   TRISA2 = 1;   // definit en entrée ( sinon, CC)
166   VRCON = (0b11000000 | (value+1));
167   }
168 }
169
170
171 //-------------------------------------------------------------------------------------
172 //-Calcul Commande exponentiel-------------------------------------------
173 //-------------------------------------------------------------------------------------
174 int CommandeExpo(unsigned int Commande,Poly* poly) {
175
176   int tmp1;
177   int tmp2;
178
179 //  int16 Tmp_c_bNum;
180
181 //-------------------------------------------------------------------------------------
182 //-Fonction Linéaire-----------------------------------------------------------
183 //-------------------------------------------------------------------------------------
184
185   tmp1 = poly->mid;
186   if (Commande <= (512/2)) {
187     tmp1 = tmp1 * Commande;
188     tmp1 = tmp1 / (512/8);  // *4
189   } else {
190     tmp2 = poly->max;
191     tmp2 = tmp2 - tmp1;
192     Commande = Commande - (512/2);
193     tmp2 = (tmp2/2) * (Commande /4) ; // on degrade precision commande ici
194     tmp2 = tmp2 / (512/64);  // *4*8*2
195     tmp1 = tmp2 + tmp1*4;  // *4
196   }
197 //-------------------------------------------------------------------------------------
198 //-------------------------------------------------------------------------------------
199
200
201 /*
202 //-------------------------------------------------------------------------------------
203 //-Fonction Polynomiale--------------------------------------------------------
204 //-------------------------------------------------------------------------------------
205 // Fonction Polynomiale . C'est plus jolie, mais le polynome explose parfois, de plus,
206 // il est difficile en corrigeant les dissimétries de la trinlerie d'obtenir 2 polynomes
207 // dont l'un n'explose pas avant l'autre.
208   // Calcul de B:
209   tmp1 = poly->mid;
210   tmp1 = tmp1 * 4.0;
211   tmp2 = poly->max;
212   Tmp_c_bNum = tmp1 - tmp2; //bNum
213
214   tmp2 = Commande*(long)Tmp_c_bNum;
215   tmp2 = (long)tmp2 / (long)bDen;  // plus touche tmp2
216
217
218   // Calcul de C :
219
220   Tmp_c_bNum = poly->max;
221   Tmp_c_bNum = Tmp_c_bNum * 2.0;
222   Tmp_c_bNum = Tmp_c_bNum - tmp1;
223   Tmp_c_bNum = (long)const0ms5_9_2/(long)Tmp_c_bNum; // c_
224
225   tmp1 = Commande * Commande;
226   tmp1 = (long)tmp1 / (long)Tmp_c_bNum;  //
227
228   tmp1 += tmp2;
229 //-------------------------------------------------------------------------------------
230 //-------------------------------------------------------------------------------------
231 */
232   return  tmp1;
233   }
234
235
236
237
238 //-------------------------------------------------------------------------------------
239 //-Saturation------------------------------------------------------------------
```

```
240 //--------------------------------------------------------------------------------
241
242 // On accepte un dépassement de 300
243 unsigned int SatureCommandeServos(unsigned int Commande){
244   if (Commande > 2300)
245     return 2300;
246   else if (Commande < 700)
247     return 700;
248   return Commande;
249 }
250
251
252
253
254
255 //--------------------------------------------------------------------------------
256 //--mixage POLYNOMIAL-----------------------------------------------------------
257 //--------------------------------------------------------------------------------
258 void mixagePolynomial(){ // Mixage puis Emission
259     int CommandeSg;
260     int CommandeSd;
261     unsigned int tmp;
262
263 // on remplace Rouli.value par AsservissementRouli
264 // et Profondeur.value par AsservissementProfondeur
265
266     if (AsservissementRouli >= 0) {
267       CommandeSg = CommandeExpo(AsservissementRouli, &mem.polyRouliPosG);
268       if (!TemoinConfigTrim)
269       CommandeSd = CommandeExpo(AsservissementRouli, &mem.polyRouliPosD);
270       else     // cas si trimage Rouli en cours
271       CommandeSd = CommandeSd = -CommandeExpo(AsservissementRouli, &mem.polyRouliNegD);
272     } else { tmp = - AsservissementRouli;
273       CommandeSg = -CommandeExpo(tmp, &mem.polyRouliNegG);
274       if (!TemoinConfigTrim)
275       CommandeSd = -CommandeExpo(tmp, &mem.polyRouliNegD);
276       else  // cas si trimage Rouli en cours
277       CommandeSd = CommandeExpo(tmp, &mem.polyRouliPosD);
278     }
279
280     if (AsservissementProfondeur >= 0) {
281       CommandeSg += CommandeExpo(AsservissementProfondeur, &mem.polyProfPosG);
282       CommandeSd -= CommandeExpo(AsservissementProfondeur, &mem.polyProfPosD);
283     } else { tmp = - AsservissementProfondeur;
284       CommandeSg -= CommandeExpo(tmp, &mem.polyProfNegG);
285       CommandeSd += CommandeExpo(tmp, &mem.polyProfNegD);
286     }
287
288   CommandeSd = SatureCommandeServos(CommandeSd + 1500 + mem.trimD);
289   CommandeSg = SatureCommandeServos(CommandeSg + 1500 + mem.trimG);
290
291 // critique : si interruption
292 // ImpulsionD & G est lu par l'interruption qui emmet les crénaux PWM
293 // Si cette interruption se déclenche alors que la donnée est en train d'être MAJ
294 // Erreur. Par exemple, si le poid faible est écrit et le poid fort pas encore..
295 // Statistiquement, cette erreur est négligeable, donc on ne bloque pas les interruptions
296   ImpulsionD = CommandeSd;
297   ImpulsionG = CommandeSg;
298 }
299
300
301
302 //--------------------------------------------------------------------------------
303 //--Filtrage--------------------------------------------------------------
304 //--------------------------------------------------------------------------------
305
306  filtre (telecommande *donnee,PWM *impulsionTele) {
307   unsigned int up;
308   unsigned int down;
309   unsigned int donneeFiltre;
310
311   up = impulsionTele->up -700; // on limite les accés indirecte : place mémoire
312   down = impulsionTele->down;
313   donneeFiltre = donnee->filtre;
314
315   // On effectue les calculs sur des unsigned pour économiser place mémoire
316
317   // Filtre IIR : b = 1/32 et a = 31/32
318   // 1600 * 32 = 51200 -> 51200 < 65535
319   if (up < (2300-700 )   //  (up >= (700-700)) // >0 : sert à rien
320     && down > 14000 && down < 24000 ) {
321     if (donnee->HorsGabarit == 0) {
322       donnee->filtre = (donneeFiltre * 31 + up) /32 ;
323       donneeFiltre =  up;
324     } else {
325       donnee->HorsGabarit = donnee->HorsGabarit - 1 ;
326     }
327    } else {
328     donnee->HorsGabarit = 8;
329   }
330
331 // conversion en valeur signé comprises entre +-700 (La norme étant +- 500)
332   donnee->value = donneeFiltre-(1500-700);
333 }
334
```

```
335
336 //-------------------------------------------------------------------
337 //--Petites fonctions-----------------------------------------------
338 //-------------------------------------------------------------------
339 unsigned char abs_Hi(int donnee) {
340   donnee = donnee >> 6;
341   if (donnee < 0) donnee = -donnee;
342   return donnee;
343 }
344
345 signed char Hi(int donnee) {
346   signed char a;
347   a = donnee >> 7;
348   if (a < 0) a = a+1;
349   return a;
350 }
351
352 //-------------------------------------------------------------------
353 //--traitement signal telecommande----------------------------------
354 //-------------------------------------------------------------------
355 // Renvoie 1 si un signal est arrivé
356 // et filtre le signal arrivé.
357 // la valeur Rouli.value ou Profondeur.value de la structure Telecommande est à jour
358 bit raffraichit Telecommande() {
359   unsigned char tmp;
360   tmp = 0;
361   if (semaphorePWM1) {
362     semaphorePWM1 = 0;
363      filtre (&Rouli,&PWM 1);
364     tmp = 1;
365   }
366   if (semaphorePWM4) {
367     semaphorePWM4 = 0;
368      filtre (&Profondeur,&PWM_4);
369     tmp = 1;
370   }
371
372   return tmp;
373 }
374
375 //-------------------------------------------------------------------
376 //--Sous routine de config_trim()-----------------------------------
377 //-------------------------------------------------------------------
378 // fait évoluer les valeurs en fct de la Telecommande (Profondeur.value)
379 int place(int donneeTrim){
380   signed char a;
381
382   a = Hi(Profondeur.value);
383   donneeTrim = donneeTrim + a;
384   if (donneeTrim > 256) donneeTrim = 256;
385   if (donneeTrim < -256 ) donneeTrim = -256;
386   return donneeTrim;
387 }
388
389 // Boucle un peu compliqué permettant de configurer les valeurs
390 // max et mid de chaque commande. L'intérêt de la boucle est
391 // d'économiser de la mémoire ROM
392 void config_trim() {
393   unsigned char i;  // pr boucle
394   unsigned char j;  // pr boucle
395   int tmp ;   // variable intermediaire
396   static bit temoin1; // temoin Telecommande en bute : pas de maj
397       // maj valeurs ssi temoin1 = 1 -> pas en bute
398
399   union typptr {  // On veut un pointeur pointant tanto un char tanto un int
400     char* c;  // warning à la compilation !
401     int* i;
402     } ptr;
403
404 // Tableau de la position des commandes
405 // Chaque valeur *125 correspond à la position des commandes
406 // chaque valeur sert pour régler gauche puis droite
407 const char array[] = {0,0,4,2,-4,-2,4,2,-4,-2};
408 /*    zeroG - zeroD
409     rien
410     Profondeur Positive max G&D
411     Profondeur Positive Milieu G&D
412     Profondeur Negative max G&D
413     Profondeur Negative Milieu G&D
414   ici, lorsqu'on règle le positif sur une voie on règle
415   ensuite le negatif sur l'autre voie ( bit TemoinConfigTrim )
416     Rouli Positive max G&D
417     Rouli Positive Milieu G&D
418     Rouli Negative max G&D
419     Rouli Negative Milieu G&D
420 Le pointeur Ptr parcours la structure mais en avançant puis revenant en arrière ...
421 */
422
423 //--Initialisation Vatiables----------------------------------------
424
425   TemoinConfigTrim = 1;  // Variable global
426     // change comportement de mixagePolynomial()
427     // Permet d'inverser le rouli sur 1 voie pour réglage
428     // Permet d'aligner les 2 ailerons
429
```

```
430   temoin1 = 0;
431   ptr.c = &mem; // pointe sur la structure contenant les valeurs des réglages
432
433   i = 0;
434   j = 0;
435
436   while (temoin1 == 0) {   // boucle tant que la télécommande est en buté
437   if (raffraichit_Telecommande()) {
438     if (abs_Hi(Rouli.value) < 3) { // (rouli)
439       temoin1 = 1;}
440   }
441   } //fin while
442
443   while (i < 10) {  // 20 valeurs à réglé au total
444
445     while (j <2) {  // Gauche puis droite
446
447     // attend telecommande. Synchro sur 2 signal, sinon, bug !
448   if (semaphorePWM1 && semaphorePWM4 && raffraichit_Telecommande()) {
449
450   if (temoin1 == 1) { // MAJ DES VALEURS
451     // en fct de la parite de i, ptr pointe sur un int ou un char
452     // On utilise les donnees de la telecommande reçu ici !
453     if (i & 0b000000001) *(ptr.c) = place(*(ptr.c));
454       else *(ptr.i) = place(*(ptr.i));
455   } // fin if temoin1 == 1
456
457   // gère le passage à la commande suivante
458     if ((abs  Hi(Rouli.value) > 5 ) && (temoin1 == 1)) {
459       temoin1 = 0 ;
460     }
461     if (abs  Hi(Rouli.value) < 3 && temoin1 == 0) {
462       temoin1 = 1;
463       if (j != 1)
464         ptr.c = ptr.c + 3;  // int suivant
465       else
466         ptr.c = ptr.c -1; // char precedant
467       j++;
468     }
469
470   // Ici, on remplace les donnees de la telecommande pour visualiser
471   // l'emplacement des ailerons
472   if (temoin1 == 1) {
473     tmp = (signed char) array[i];
474     tmp = tmp * 125;
475     Profondeur.value = 0;
476     Rouli.value = 0;
477     if (i<=5) Profondeur.value = tmp;
478     else Rouli.value = tmp;
479   }
480
481   AsservissementProfondeur = Profondeur.value;
482   AsservissementRouli = Rouli.value;
483
484   // On utilise la même routine que pendant le vole. On est ainsi certain du résultat
485   mixagePolynomial();
486
487   } // fin  raffraichit  Telecommande
488
489   }// fin while j
490   if (i & 0b000000001)
491     ptr.c = ptr.c+2;  // passe à la structures de polynome suivant
492   j = 0;
493   i++;
494   }// fin while i
495
496   TemoinConfigTrim = 0;  // change comportement de mixagePolynomial()
497 }
498
499 /* ne marche pas : probablement, fct EEPROM  READ doivent être appelé de la racine
500 //-- Lecture donnees ds EEPROM----------------------
501 void LectureEcritureEEPROM(unsigned char lecture) {
502   unsigned char* PtrChar;
503
504   PtrChar = &mem;
505   for (tmp3 = 0 ;tmp3 < (3*10) ;tmp3 ++ ) {
506     if (lecture != 0) *PtrChar = EEPROM_READ(tmp3);
507     else EEPROM_WRITE(tmp3,*PtrChar);
508     PtrChar++;
509   }
510
511
512   FSR = &mem;
513   for (tmp3 = 0 ;tmp3 < (3*10) ;tmp3 ++ ) {
514     if (lecture != 0) INDF = EEPROM_READ(tmp3);
515     else EEPROM_WRITE(tmp3,INDF);
516     FSR++;
517   }
518
519 }
520 */
521
522
523 void filtreADXL(Accelerometre* Accel,PWM* donnee){
524   int tmp;
```

```
525   int AccelValue = Accel->value;  // limtie acces indirecte
526   int donneediff = donnee->up - donnee->down;
527
528   tmp =( (AccelValue * 3) + (signed int) (donneediff) ) / 4;
529   Accel->Derivative = ( (Accel->Derivative * 7) + tmp - AccelValue ) / 8;
530
531   Accel->value = tmp;
532
533 }
534
535 //-------------------------------------------------------------------------------
536 //-------------------------------------------------------------------------------
537 //--MAIN-----------------------------------------------------------------
538 //-------------------------------------------------------------------------------
539 //-------------------------------------------------------------------------------
540
541
542
543 void main( void ) {
544
545 unsigned char tmp3;     // Variable temporaire pr boucle
546 unsigned char TemoinPresenceAccelerometre; // Compte nombre d'impulsion du capteurs
547
548 //---INITIALISAITON-------------------------------
549
550 T1CON = 0b00001001;  // Enable Timer 1 prescaller = 1:1 = Fosc/4
551
552 TRISB = 0b00111111; // RB5 RB4 RB3 en entrée ; par default, tt en entree
553
554 TRISA = 0b00111111; // RA7 RA6 en sortie
555
556 //PORTB = 0b00111001; // pull up
557 //OPTION = 0b01111111; // PullUP/ INTEDG ...
558 //RBPU = 0;  // pull up
559
560 CCP1CON = 0b00000101; // capture every rising edge
561       // 0100 : falling edge
562
563 //PR2 = 255;   // Period Register Default value at startup
564 T2CON = 0b00000100; // Timer2 6-3 postscaller - 2:On - 0-1 Prescaler
565
566 initRS232e(12); // 19200
567
568 //  Initialise  Variables
569
570   ImpulsionG = 1500 ;
571   ImpulsionD = 1500;
572
573
574 //---FIN INITIALISAITON---------------------------------
575 //-------------------------------------------------------------
576 //-- Lecture donnees ds EEPROM------------------------
577
578   FSR = &mem;
579   for (tmp3 = 0 ;tmp3 < (3*10) ;tmp3 ++ ) {
580     INDF = EEPROM_READ(tmp3);
581     FSR++;
582   }
583
584
585 //-----------------------------------------------------------
586 // config interruptions
587
588 INTEDG = 1;  // rising edge sur RB0
589
590 PIE1 = 0b00000110; // EEIE CMIE RCIE TXIE - CCP1IE TMR2IE TMR1IE
591 //PIE1 = 0b00000100; // EEIE CMIE RCIE TXIE - CCP1IE TMR2IE TMR1IE
592
593 RBIE = 1;  // interrupt on change enable for PORTB configured as input
594 INTE = 1;
595
596 GIE = 1;
597 PEIE = 1;
598
599 TemoinConfigTrim = 0;  // change comportement de mixagePolynomial()
600
601
602 //-----------------------------------------------------------
603 //-----------------------------------------------------------
604 for (;;) {     // main boucle
605
606   if (raffraichit_Telecommande() || semaphorePWM2 || semaphorePWM3) {
607
608       if (semaphorePWM2 || semaphorePWM3) {
609         if (semaphorePWM2) {
610           filtreADXL(&ADXLTanguage,&PWM_2);
611           semaphorePWM2 = 0;
612 //          while(send232Buff(ADXLTanguage.value,1+4*0));
613 //          while(send232Buff(ADXLTanguage.Derivative,1+4*2));
614
615         }
616
617         if (semaphorePWM3) {
618           filtreADXL(&ADXLRouli,&PWM_3);
619           semaphorePWM3 = 0;
```

```
620 //           while(send232Buff(ADXLRouli.value,1+4*1));
621 //           while(send232Buff(ADXLRouli.Derivative,1+4*3));
622         }
623
624         if (TemoinPresenceAccelerometre < 250)
625           TemoinPresenceAccelerometre = TemoinPresenceAccelerometre + 2;
626       } else { // c'est la telecommande qui a bougé
627         if (TemoinPresenceAccelerometre != 0) TemoinPresenceAccelerometre --;
628       } // fin        if (semaphorePWM2 || semaphorePWM3) {
629
630
631
632     // sans asservissement :
633       AsservissementProfondeur = Profondeur.value;
634       AsservissementRouli = Rouli.value;
635
636     // asservissement ssi télécommande allumé depuis plus de n secondes
637     // & capteur ADXL présent
638     if (TemoinTeleEteinteCourte == 0) {
639     if (TemoinPresenceAccelerometre > 120) {  // Asservissement si capteur présent
640         if (TemoinPresenceAccelerometre < 128) { // capture zero des accelerometres
641           ADXLTanguage.zero = ADXLTanguage.value;
642           ADXLRouli.zero = ADXLRouli.value;
643         }
644         AsservissementProfondeur = Profondeur.value +
645           ( (ADXLTanguage.value - ADXLTanguage.zero) + ADXLTanguage.Derivative * 20 )
646           /(2*4);
647
648         AsservissementRouli = Rouli.value -
649           ( (ADXLRouli.value - ADXLRouli.zero) + ADXLRouli.Derivative * 20)
650           /(3*4);
651     }// fin  if (TemoinPresenceAccelerometre > 120
652     }// fin   if (TemoinTeleEteinteCourte != 0)
653
654
655 //    send232Buff(Rouli.HorsGabarit,0+4*0);
656 //    send232Buff(PWM_2.up,1+4*1);
657
658
659     if (!Rouli.HorsGabarit && !Profondeur.HorsGabarit) { // telecommande allume !
660       if (modeConfig >= 120) {
661         TemoinTeleEteinteCourte = 0;
662         if ((abs_Hi(Rouli.value) >= 3) && (abs_Hi(Profondeur.value) >= 3) )
663           modeConfig++; else modeConfig--;
664       }
665       else {
666       if (modeConfig > 20 && modeConfig < 50 ) TemoinTeleEteinteCourte = 500;  // Initialise compteur
667                 // celui est decompté ds interruption d'Emission sur les Servos
668       modeConfig = modeConfig/2;
669       }
670     }
671
672   // Gestion BEEPER
673     if (TemoinTeleEteinteCourte != 0)
674       EmetBeeper(16+1-TemoinTeleEteinteCourte/30); //17 28
675     else
676       EmetBeeper(0);
677     // Alume Beeper si Telecommande précédement éteinte et maintenant allumé !
678   // fin  Gestion BEEPER
679
680     if (modeConfig == 255) {
681       modeConfig = 0;
682       config_trim();
683     FSR = &mem;
684     for (tmp3 = 0 ;tmp3 < (3*10) ;tmp3 ++ ) {
685       EEPROM_WRITE(tmp3,INDF);
686       FSR++;
687       } // fin  for
688     } // fin  mode config == 255
689
690
691     mixagePolynomial();
692
693 } // fin  if (semaphorePWM1 || semaphorePWM2) {
694
695
696
697
698 //  if (semaphorePWM4){
699 //    while(send232Buff(PWM_4.down,1+4*0));
700 //    while(send232Buff(PWM_4.up,1+4*2));
701 //    semaphorePWM4 = 0;
702 //  }
703 //  if (semaphorePWM3){
704 //    while(send232Buff(PWM_3.down,1+4*2));
705 //    while(send232Buff(PWM_3.up,1+4*3));
706 //    semaphorePWM3 = 0;
707 //  }
708
709
710
711 }// fin  for (;;)
712
713
714 /*
```

```
715   if (semaphorePWM4){
716     while(send232Buff(PWM_4.down,1+4*0));
717     while(send232Buff(PWM_4.up,1+4*1));
718     semaphorePWM4 = 0;
719
720 */
721
722
723 } // fin void main
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754 //-------------------------------------------------------
755 //-------------------------------------------------------
756 //-- INTERRUPTIONS ---------------------------------------
757 //-------------------------------------------------------
758 //-------------------------------------------------------
759
760
761
762
763
764
765 #pragma interrupt_level 0
766 void interrupt IntVector(void)
767 {
768   static unsigned int CurrentPWM; // decompteur PWM actuellement émis
769
770
771   static unsigned int CCP1TMR;
772   static unsigned char Tmp; // Sauve de l'Etat du port B pour analyse changement bit
773   unsigned char Tmp2;
774   static unsigned char RB;
775
776   static unsigned int RBTMR; // Arret Compteur pour port B
777   static unsigned int RB2TMR;
778   static unsigned int RB3TMR;
779   static unsigned int RB4TMR;
780
781   static bit semaphoreSyncroTelecommande;
782
783   static bit StateTXIE;
784 //  static bit ErrCCP1IF; // pas la peine : horloge bloqué par interrupt
785   static bit ErrRBIF;
786   static bit ErrINTF;
787
788   static bit ErrRBIF4;
789   static bit ErrRBIF5;
790   static bit ErrINTF0;
791
792 //---------- sert pour 2 interruption du port B
793   RBTMR = (unsigned char) (TMR1L + 4) ; // 7 instruction separe lecture poids faible/fort
794   RBTMR = (TMR1H << 8) + RBTMR;
795 //-------------------------------------------------------
796
797
798 //-------------------------------------
799 // Interruption Generation des impulsions de commande PWM utilisant le timer2
800 //-------------------------------------------------------
801 // On jongle avec le post-scaler du timer 2 afin de générer un minimum d'interruption
802   if (TMR2IF && TMR2IE) {
803     TMR2IF = 0;
804     T2CON = ( T2CON & 0b00000111);
805
806     if (!EmittingPWMG && !EmittingPWMD && CurrentPWM > 14000 && semaphoreSyncroTelecommande){
807       CurrentPWM == 14000;
808       semaphoreSyncroTelecommande = 0;
809     }
```

```
810
811     if (CurrentPWM == 0) {
812       PR2 = 255;
813       TMR2IF = 0;
814
815       if (EmittingPWMG){
816         RA6 = 0;
817         EmittingPWMG = 0;
818         RA7 = 1;
819         EmittingPWMD = 1;
820         CurrentPWM = ImpulsionD;
821       } else if (EmittingPWMD){
822         RA7 = 0;
823         EmittingPWMD = 0;
824         CurrentPWM = DeconteurInterImpulsion;
825       } else {
826         if (modeConfig < 128) modeConfig++; // pour entre en mode config
827         if (TemoinTeleEteinteCourte != 0) TemoinTeleEteinteCourte--; // decompteur
828         RA6 = 1;
829         EmittingPWMG = 1;
830         CurrentPWM = ImpulsionG;
831       }
832
833         // Current PWM > 512 sinon, bug !
834
835       T2CON = ( ((CurrentPWM >> 8)-2) << 3) | T2CON;
836       CurrentPWM = CurrentPWM - (( T2CON >> 3) +1)*256;
837
838
839       CCP1IE = 1;
840       TXIE = StateTXIE;
841       RBIE = 1;
842       INTE = 1;
843
844
845     } else {
846
847       if ( CurrentPWM <= 512) {
848
849         if (CurrentPWM <= 256) {
850           CCP1IE = 0;
851           StateTXIE = TXIE;
852           TXIE = 0;
853           RBIE = 0;
854           INTE = 0;
855           PR2 = (unsigned char) CurrentPWM - 1 ;
856           }
857         else PR2 = (unsigned int) (CurrentPWM / 2)-1 ;
858
859         TMR2IF = 0;
860         CurrentPWM = (unsigned int) CurrentPWM - (unsigned char) PR2 -1;
861
862       } else  {   //  < 512
863
864 //        if (CurrentPWM >= 512)
865           T2CON = ( ((CurrentPWM >> 8)-2) << 3) | T2CON;
866
867         CurrentPWM = CurrentPWM - (( T2CON >> 3) +1)*256;
868       }
869
870     }
871
872   } // FIN if (TMR2IF && TMR2IE)
873
874  else {    // !  if (TMR2IF && TMR2IE) {
875
876  //---------------------------------------
877  // Interruption capture PWM sur port B
878    if (RBIE && RBIF) {
879      Tmp2 = PORTB;
880      RBIF = 0;  // rearme interrupt
881      Tmp = (RB ^ Tmp2);
882      RB = Tmp2;
883
884  //--- Traite pin RB4 comme entree PWM
885      if (Tmp & 0b00010000) {
886        if (RB & 0b00010000){   // rising edge
887          if (!ErrRBIF && !ErrRBIF4) PWM_2.down = RBTMR - RB2TMR ;
888          if (!ErrRBIF) RB2TMR = RBTMR;
889        } else {          // falling edge
890          if (!ErrRBIF && !ErrRBIF4) {
891            PWM_2.up = RBTMR - RB2TMR ;
892            semaphorePWM2 = 1;
893            semaphoreSyncroTelecommande = 1;
894            }
895          if (!ErrRBIF) RB2TMR = RBTMR;
896         }
897      ErrRBIF4 = ErrRBIF;
898      }
899
900
901  //--- Traite pin RB5 comme entree PWM
902      if (Tmp & 0b00100000) {
903        if (RB & 0b00100000){   // rising edge
904          if (!ErrRBIF && !ErrRBIF5) PWM_3.down = RBTMR - RB3TMR ;
```

```
905         if (!ErrRBIF) RB3TMR = RBTMR;
906       } else {          // falling edge
907         if (!ErrRBIF && !ErrRBIF5) {
908           PWM_3.up = RBTMR - RB3TMR ;
909           semaphorePWM3 = 1;
910           }
911         if (!ErrRBIF) RB3TMR = RBTMR;
912         }
913     ErrRBIF5 = ErrRBIF;
914
915     }
916
917   ErrRBIF = 0;
918
919   } //Fin interrupt on change RB4-7
920
921 else { // ! if (RBIE && RBIF) {
922
923 //-----------------------------------
924 // interruption externe : RB0
925 //--- Traite pin RB0 comme entree PWM
926
927   if (INTE && INTF) {
928     if (INTEDG){    // rising edge
929       if (!ErrINTF && !ErrINTF0) PWM_4.down = RBTMR - RB4TMR ;
930       if (!ErrINTF) RB4TMR = RBTMR;
931       INTEDG = 0;
932     } else {          // falling edge
933       if (!ErrINTF && !ErrINTF0) {
934         PWM_4.up = RBTMR - RB4TMR ;
935         semaphorePWM4 = 1;
936         }
937       if (!ErrINTF) RB4TMR = RBTMR;
938       INTEDG = 1;
939       }
940
941     INTF = 0;
942     ErrINTF0 = ErrINTF;
943     ErrINTF = 0;
944   }
945
946 else {  // !   if (INTE && INTF) {
947
948 //-----------------------------------
949 // Interruption capture PWM. L'horloge est sauvegardé au moment du déclenchement de l'interruption
950 // 1 entrées PWM capturés ainsi
951   if (CCP1IE && CCP1IF) {
952     CCP1IF = 0;
953     if (CCP1M0) {   // rising edge
954       PWM_1.down = ((CCPR1H << 8)+CCPR1L) - CCP1TMR ;
955       CCP1TMR = (CCPR1H << 8)+CCPR1L;
956       CCP1M0 = 0;} // detect next falling edge
957     else {          // falling edge
958       PWM_1.up = ((CCPR1H << 8)+CCPR1L) - CCP1TMR ;
959       CCP1TMR = (CCPR1H << 8)+CCPR1L;
960       CCP1M0 = 1;
961       semaphorePWM1 = 1;
962       }
963   }
964
965 // PROTOCOLE RS 232 ------------------------------------
966   else if (TXIE && TXIF) {
967     if   ( RS232PtrHaut != RS232PtrBas ) {
968       TXREG = RS232Tab[RS232PtrBas];
969       RS232PtrBas = (RS232PtrBas + 1) & 0b00001111;
970       if (RS232PtrHaut == RS232PtrBas)
971         TXIE = 0;    // enleve interrupt
972     } else TXIE = 0;     // enleve interrupt
973   }
974
975
976 } //    else {  // !   if (INTE && INTF) {
977 } // fin   else { // !  if (RBIE && RBIF) {
978 } // fin   else {    // !   if (TMR2IF && TMR2IE) {
979
980       ErrRBIF = RBIF;
981       ErrINTF = INTF;
982
983 } // FIN INTERRUPTION
```

Listing 1: *Mixer628.c*

C Protocole PIC -> Matlab

Dans cette annexe, les outils créés puis utilisés pendant le déroulement du projet sont étudiés.

C.1 Recuperation de données provenant du micro-contrôleur sous Matlab

La réalisation décrite ci-après permet d'acquérir dans matlab des données provenant du microcontrôleur et d'effectuer ensuite du traitement sur ces données à la volée [1] Ce n'est pas de la simulation temps réel, car les données sont traitées par paquet de N valeurs. Ce système ne permet donc pas de faire des simulations temps réel comme le font les modules xPC Target et Windows Target. Une autre réalisation décrite dans le rapport de projet de fin de 3e année permet de connecter un microcontrôleur PIC sur xPC Target et Windows Target et de réaliser des simulations en Temps réel. C'est-à-dire où les résultats des calculs peuvent être utilisés pour modifier le comportement du système.. La liaison série s'effectue via un port COM du PC. Les données proviennent des capteurs, eux-mêmes reliés au microcontrôleur via les entrées analogiques ou numériques de celui-ci. Ce système nous permet de tester différentes méthodes de filtrage dans matlab avant de les implementer dans le microcontrôleur. Il nous permet aussi de vérifier les calculs intermédiaires effectués dans le microcontrôleur permettant ainsi de debugger les programmes embarqués.

Le système est composé de deux parties distinctes : une interface graphique sous matlab et un programme embarqué dans le microcontrôleur utilisant le périphérique UART de celui-ci ainsi que les interruptions dédiées.

C.1.1 Protocole

Les données sont transmises par le protocol RS232 de manière asynchrone. Le protocole est en sens unique : du microcontrôleur vers matlab. Des octets de contrôle sont envoyés avec des octets de données. Les octets de contrôle permettent de spécifier le type de la donnée qui le suit : 8 ou 16 bits ainsi que le canal de celle-ci. À la réception, Matlab place les données dans une matrice dont chaque colonne correspond à un canal de donnée. Le numéro de ligne où sera enregistrée la prochaine donnée est incrémenté à chaque réception de donnée. Chaque ligne de la matrice contient donc une seule donnée. Les autres colonnes de la ligne sont remplies par Nan.
On obtient une matrice de ce type :

$$\begin{pmatrix} 1 & NaN & NaN \\ NaN & 43 & NaN \\ NaN & NaN & 145 \\ -5 & NaN & NaN \\ \vdots & \vdots & \vdots \end{pmatrix}$$

Les octets de contrôle ont la structure suivante :

- bits 1-0 signature : toujours 01
- bits 3-2 Type de la donnée : 00 => 8 bits ; 01 => 16 bits
- bits 7-4 Canal : 16 canal possible

1.

La réception se fait par trame de donnée. Matlab récupère les données durant un intervalle de temps Δt configurable : par défaut, $\Delta t = 1s$. La séquence ainsi récupérée est dans un premier temps traité afin d'en extraire les données en minimisant l'effet des erreurs de transmissions. Les octets de contrôle sont signés de par leurs deux derniers bits 01 et chaque octet de contrôle contenant la taille de la donnée envoyée, on peut donc déterminé l'emplacement du prochain octet de contrôle.

L'algorithme d'extraction de donnée crée donc un graphe orienté reliant tous les octets contenant la signature d'un octet de contrôle à l'octet de contrôle le suivant. On extrait ensuite le plus long chemin de ce graphe obtenant ainsi les octets de contrôle et permettant d'extraire les données. L'algorithme est capable de connecter les graphes des séquences se suivant optimisant ainsi la solution entre chaque trame. (listing n°3 p.53)

C.1.2 Interface Matlab

Une interface graphique dans Matlab permet de sélectionner :

- La vitesse de transmission
- Le port COM à utiliser
- Le nom et la taille maximum des variables du workspace dans lesquels sont placées les données
- le temps Δt d'une séquence
- Les calculs et affichage à effectuer en temps réel sur les données
- le schéma Simulink à exécuter après réception des données

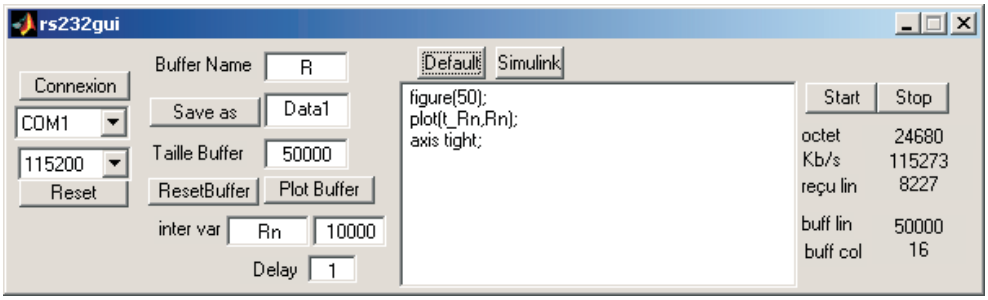

FIG. 27 – *Interface rs232gui*

Le listing du code de l'interface est le numéro (6) p.55.

C.1.3 Routine du Micro-contrôleur

Le microcontrôleur utilisé : le PIC 16f877 dispose d'un périphérique USART dédié aux liaisons séries. En particulier, le module d'émission s'utilise en plaçant l'octet à émettre dans le registre spécifique TXREG. Ce registre est copié automatiquement par le microcontrôleur dans un second registre TSR d'où l'émission débutera par rotation de l'octet dans ce registre. Cette copie a lieu dès que l'émission de l'octet précédent est terminée. L'USART s'arrête si l'octet du registre TXREG n'a pas été mis à jour depuis la dernière copie. Il y a donc en permanence un registre contenant l'octet en train d'être émis et un autre registre avec le prochain octet à envoyer. À chaque fin d'émission, le microcontrôleur déclenche une interruption.

Ce système a un buffer de 1 octet. Ce dernier est trop petit pour notre application car pour envoyer une valeur 16 bits, il nous faut envoyer 2 octets + 1 octet de contrôle soit 3 octets au total. Le système doit donc attendre que les 2 premiers octets soient envoyés avant de pouvoir effectuer d'autre tâches. L'exécution du programme est alors fortement modifiée par l'émission de données. Ceci peut est gênant si l'on veut utiliser ce système de transmission pour debugger un programme qui est en temps critique.

Une routine a donc été développée afin de pouvoir envoyer des données en modifiant peu le déroulement temporel du programme. On utilise un système de pile avec un buffer circulaire. La première partie de la routine se situe dans l'interruption déclenchée par le périphérique USART lorsqu'une donnée a été envoyée. Cette partie prend alors le prochain octet dans la pile qui doit être envoyée et le place dans le registre du périphérique de l'USART. Si la pile est vide, l'USART est arrêté et n'émet alors plus rien. La seconde partie de la routine est la fonction **send232Buff(int valeur, unsigned char cannal)** appelée par programme pour envoyer une donnée. Cette fonction renvoie 0 si le buffer dispose de suffisamment de place pour y placer les données à envoyer et 1 dans le cas contraire. Lorsque la fonction renvoie 1, la donnée n'est pas enregistrée dans le buffer et est donc perdu.

Le système de buffer circulaire pour gérer la pile est optimisé pour être très rapide à l'exécution et utilise peu de ressource mémoire. La pile à une taille de 16 octets. Le code C correspondant est le listing (2) p.51.

C.1.4 Montage Electronique

Le protocole série du PC utilise des niveaux logiques situés à +15V et -15V. Il n'est donc pas possible de connecter directement un PIC dont les niveaux logiques sont situés à 0V et +5V. On utilise donc un composant dédié permettant d'adapter ces niveaux. On a choisi un MAX232. Ce composant nécessite une alimentation de 0-5V et crée par un système de pompe de charge, les +15V et -15V nécessaires à la transmission série avec le PC.

C.2 fonctions PIC

Listing 2: *RS232e.c*

identifierstyle

```
1  /*
2  Compilateur : PICC
3  Timer d'emission basé sur la fréquence du quartz
4
5  Reste à poser GIE et PEIE ( interrupt périphériques )!
6
7  fonctions :
8    bit send232Buff(int valeur, unsigned char cannal)
9    canal(0:1) = nbr de bits a enboye +1
10   canal(6:2) = n° du canal
11 PIC16F877
12 */
13
14 //-------------------------------------------------------------------------
15 //-Definition Entrees Sorties--------------------------------------------
16 //-------------------------------------------------------------------------
17
18 // RC6 en sortie
19
20 //-------------------------------------------------------------------------
21 //-Variables-------------------------------------------------------------
22 //-------------------------------------------------------------------------
23 //bankx unsigned ...
24
25 unsigned char  RS232Tab[16];
26 unsigned char  RS232PtrHaut = 0;
27 unsigned char  RS232PtrBas = 0;
28
29 //-------------------------------------------------------------------------
30 //-config----------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------

// CONFIG UART :

void initRS232e() {
SPBRG = 10;//12 115200@24MHz 10 115200@20MHz   //155;// 2400@24MHz // X = freq /(16*BaudRate) -1;
//SPBRG = 7; // 115200@14043.75


TXSTA = 0b00100100; // bit 7 : Clock Source : 1-master 0-Slave async : don't care
      // bit 6 : parity bit 9 Transmit enable(1)
      // bit 5 : Transmition enable (1)
      // bit 4 : 1-Synchrone 0-Asynchrone
      // bit 3 : unimplemented = 0
      // bit 2 : high baud rate select BRGH (1-rapide)
      // bit 1 : transmit bit status ( 1 - TSR empty) (read only)
      // bit 0 : TX9D : parity bit to send

RCSTA = 0b10010000;
      // bit 7 : SPEN Serial port enable ( configure RC7/Rx-RC6/Tx
      // bit 6 : RX9 9bit Receive enable (1)
      // bit 5 : SREN Signle Receive Enable bit
      // bit 4 : CREN continuous Receive Enable bit(1)
      // bit 3 : ADDEN Address Detect bit
      // bit 2-0 : read only

TRISC6 = 0; // Tx en sortie
}

//-------------------------------------------------------------------------------
//-fonction----------------------------------------------------------------
//-------------------------------------------------------------------------------
//
// valeur : valeur à envoyé
// cannal : XX 0000 00
//       bit 0-1 : nbr bits à enboyé (+1)
//       bit 2-5 : cannal choisi
//       bit 6-7 : non utilisés ( signature octet controle )
// renvoie 0 si donnes mise dans le buffet et 1 si donnée perdu (buffer plein)
bit send232Buff(int valeur, unsigned char cannal) {
  unsigned char Tmp;
  unsigned char Tmp2;
  Tmp = ~(RS232PtrHaut - RS232PtrBas); // Tmp = 7 - (Haut - Bas) modulo 16
  Tmp = Tmp & 0b00001111;

  Tmp2 = (cannal & 0b00000011)+2;  // Nbr bits à envoye
  if (Tmp >= Tmp2) {     // Si on a la place ds le buffer d'envoye
      RS232Tab[RS232PtrHaut] = ((cannal << 2) & 0b11111100) | 1;
            // envoie 6 derniers bits du canal et bit 1=0 ; bit 0=1
            // signature octet controle en poids faible
      RS232PtrHaut = (RS232PtrHaut + 1) & 0b00001111;

    for(Tmp=1; Tmp < Tmp2 ; Tmp++) {
      RS232Tab[RS232PtrHaut] = valeur & 0b11111111;
      valeur = valeur >> 8;
      RS232PtrHaut = (RS232PtrHaut + 1) & 0b00001111;
    }
    Tmp = 0;  // bufferisation réussit
    TXIE = 1; // enable interrupt

  } else Tmp = 1;

  return Tmp;
}

//-------------------------------------------------------------------------------
//-Interruption----------------------------------------------------------------
//-------------------------------------------------------------------------------

/*
#pragma interrupt_level 0
void interrupt IntVector(void)
{
  if (TXIE && TXIF) {
    if  ( RS232PtrHaut != RS232PtrBas ) {
      TXREG = RS232Tab[RS232PtrBas];
      RS232PtrBas = (RS232PtrBas + 1) & 0b00001111;
      if (RS232PtrHaut == RS232PtrBas)
        TXIE = 0;    // enleve interrupt
    } else TXIE = 0;     // enleve interrupt
  }
}
*/
```

Listing 2: *RS232e.c*

C.3 fonctions Matlab

C.3.1 Extraction de données des séquences

Listing 3: *rs232_ analyzBloc.m*

identifierstyle

```
function [R,t_R,sr,ValDernierNoeud,tpsDernierNoeud] = RS232_analyzBloc(s,ValNoeud1,varargin);
% function [R,sr] = RS232_analyzBloc(s,ValNoeud1)
% Extraction optimal des données contenus ds s
% s : tableau des caractères à analysee
% ValNoeud1 : longueur de la derniere chaine effectué : pour analyze de
% bloc consécutifs
% varargin : 2 parametres : tps du 1er octer de la séquence suivi du tps
% correspondant de l'octet suivant le dernier octet de la séquence. le tps
% entre chaque octet de la chaine est supposé linéaire, on donne un tps à
% chaque variable en se basant sur ces 2 valeurs
%
% R : contient en colonnes les valeurs de chaque channel
% t_R : contient les tps correspondant à chaque caractere de controle des
% données si varargin contient les 2 parametres de temps.
% sr : reste de la chaine non analyze car donnes incomplete a concatene a
% la prochaine chaine
% tpsDernierNoeud : le 1er tps correspondant au 1er caractere de la chaine restante


warning off MATLAB:conversionToLogical;
n = length(s);
t_R = [];
if (n < 27)
    R = [];
    sr = s;
    ValDernierNoeud = ValNoeud1;
else

%-------------------------------------------------------------------------
% Constantes
%-------------------------------------------------------------------------

decidePCanal = 1-.1;            % decide existence d'un cannal si contient au moins 1%
% en nbr de valleurs du nombre de valeur ds
% le canal max

if nargin < 3
    nbrColTps = 0;
elseif nargin == 4
    nbrColTps = 1;
    Temps = linspace(varargin{1},varargin{2},n+1);
else error('nombre d"arguments incorrectes');
end

%-------------------------------------------------------------------------
% Pre-Traitement
%-------------------------------------------------------------------------

s_control = bitand(s,3);          % garde 2 derniers bits : signature caract controle
s_control_pos = (find(s_control == 1)) ;

s_control_nbrBits = (bitand(s(s_control_pos),15-3))/4 +1; % calcule nbr bits jusqu'au prochain
% caract de
% controle max
% = 5 !

s_control_nbrBitsdirect(s_control_pos) = (bitand(s(s_control_pos),15-3))/4 +1; % max = 5

s_control_canal(s_control_pos) = (bitand(s(s_control_pos),255-15))/16; % calcule le canal du bit

%-------------------------------------------------------------------------
% Probleme de graphe orienté : on cherche le chemin le plus long
% on part d'un noeud et on parcours le graphe en ajoutant 1 à chaque noeud
% puis on reconstruit en partant du dernier noeud ayant une grande valeur
%-------------------------------------------------------------------------
s_analyz = zeros(n+5,2);
s_analyz(s_control_pos,1) = 1;

if (size(s_control_pos)< 1)
    error('Aucun caractere de controle trouve dans les octets reçu ! ');
end

s_analyz(s_control_pos(1),1) = min(ValNoeud1+1,n); % chainage avec traitement précédent

for i=1:length(s_control_pos);
    a = s_control_pos(i)+s_control_nbrBits(i)+1;
    if (s_analyz(a,1)) > s_analyz(s_control_pos(i),1)
        continue;
    else s_analyz(a,1) = s_analyz(s_control_pos(i),1) + 1;
        s_analyz(a,2) = s_control_pos(i);          % Pour reconstruire la solution
        %         s_analyz(a,3) = s(s_control_pos(i)+1);
    end
end
s_analyz = s_analyz(1:n,:);


%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
% Construction de la solution: on parcours en partant de la fin
%-------------------------------------------------------------------------

%-Initialisation--------------------------------------------------------
```

```
Ptr = n-5-1;                        % addresse max accede actuellement ( 2 caractere peuvent suivre)
                                      % mais on en prend 5 par sécurité ( nbr
                                      % max réel
Rcount = zeros(1,16);
R = zeros(n,16)*NaN;              % mieux vaut prévoir très grand et réduire ensuite
t_R = zeros(n,1)*NaN;
z = 1;


%-------------------------------------------------------------------------
% On recupere la chaine non traite a renvoye pour analyz plusieurs blocs
% on traite fictivement le dernier caractere
blocstart = Ptr - 20 ;
if blocstart > 0
    PtrFinAnalyz = find( s_analyz(blocstart:Ptr+1,1) == max(s_analyz(blocstart:Ptr+1,1)) );
    PtrFinAnalyz = PtrFinAnalyz + blocstart -1;
    ValDernierNoeud = (s_analyz(PtrFinAnalyz,1));


          %rajoute le nombre d'octets qui suivent la dernier caractere de
          %controle traité + le tps correspondant
    PtrFinAnalyz = PtrFinAnalyz + s_control_nbrBitsdirect(PtrFinAnalyz);
    varargin(2)=[];

    if (PtrFinAnalyz == length(s) )
        sr = [];
        if (nargin > 3) tpsDernierNoeud = varargin{2}; end;
    else
        sr = s(PtrFinAnalyz+1:length(s))';
        if (nargin > 3) tpsDernierNoeud = Temps(PtrFinAnalyz+1); end;
    end

else % if blocstart < 0
    sr = s';                    % si moins de 20+1+2 caracteres à traites
    ValDernierNoeud = 0;
    R = [];
    tpsDernierNoeud = varargin{2};
end


% fin traitement  fictif  du dernier caractere

while Ptr-20 > 0
    blocstart = Ptr - 20 ;
    Ptr = find( s_analyz(blocstart:Ptr+1,1) == max(s_analyz(blocstart:Ptr+1,1)) );
    Ptr = Ptr + blocstart -1;

    while ( (Ptr > 0) && (s_analyz(Ptr,1) ~= 0) )
        R(z,s_control_canal(Ptr)+1) = s(Ptr+1);
        if (s_control_nbrBitsdirect(Ptr) == 2)
            R(z,s_control_canal(Ptr)+1) = s(Ptr+1) + s(Ptr+2)*256;
        end
        if (nbrColTps ==1)
            t_R(z)=Temps(Ptr);              % temps de variable = tps du caract de control
        end

        Rcount(1,s_control_canal(Ptr)+1) = Rcount(1,s_control_canal(Ptr)+1)+1;
        Ptr = s_analyz(Ptr,2);
        z=z+1;
    end
end

decimecolone = logical(floor(Rcount*decidePCanal));

R = R(1:z-1,decimecolone);    % Redimensionne R
R = fliplr(R')';     % Reordre R
t  R = t  R(1:z-1);
t_R = fliplr(t_R')';
end % if n < 27
```

Listing 3: *rs232_ analyzBloc.m*

C.3.2 post Traitement matrice de données

Listing 4: *pad.m*

```
identifierstyle
function RtFull = pad(R)
%function Rfull = pad(R)
%remplace les NaN d'une matrice par la première valeur de leurs ligne
%précédente

RtFull = R;
[n,m]=size(R);
tmp = zeros(m,1)*NaN;     % les premières valeurs de la matrices seront NaN
for i=1:n
    for j=1:m
        if (isnan(RtFull(i,j)))
            RtFull(i,j)=tmp(j);
        else
```

```
            tmp(j)=RtFull(i,j);
        end
    end
end

K = find((isnan(RtFull(1:n,:)))==1);   % rempli les premières valeur NaN
for i=fliplr(K')
if (i < n*m)    RtFull(i)=RtFull(i+1); end
end
```

Listing 4: *pad.m*

C.3.3 post Traitement matrice de données

Listing 5: *padR.m*

identifierstyle

```
function [RtFullR,Tout] = padr(R,coltrigger,varargin)
%function [RtFullR,Tout] = padr(R,coltrigger,varargin)
%supprime les NaN d'une matrice en concaténant les lignes alentours
% la matrice RtFullR de retour à moins de lignes
% au passage de nombre à NaN de la colone coltrigger, la ligne est
% sauvegardé
% decime aussi le vecteur temps passé en parametre optionnel

Tuot = [];

if (nargin == 3)
    T = varargin{1};
end

[m,n] = size(R);
if ( coltrigger > n) error('la colonne servant de trigger n"existe pas dans la matrice');
end

RtFull = pad(R);
d= isnan(R(:,coltrigger));
d = diff(d);
d = find(d == 1);
RtFullR = RtFull(d,:);
Tout = T(d,:);
```

Listing 5: *padR.m*

C.3.4 Interface Graphique

Listing 6: *rs232gui.m*

identifierstyle

```
function varargout = rs232gui(varargin)
% RS232GUI M-file for rs232gui.fig
% fonction de transmition PIC -> Matalb via Port Serie
%      RS232GUI, by itself, creates a new RS232GUI or raises the existing
%      singleton*.
%
%      H = RS232GUI returns the handle to a new RS232GUI or the handle to
%      the existing singleton*.
%
%      RS232GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in RS232GUI.M with the given input arguments.
%
%      RS232GUI('Property','Value',...) creates a new RS232GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before rs232gui_OpeningFunction gets called. An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to rs232gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help rs232gui

% Last Modified by GUIDE v2.5 28-Sep-2003 11:08:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @rs232gui_OpeningFcn, ...
    'gui_OutputFcn', @rs232gui_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []) ;
```

```
36  if nargin & isstr(varargin{1})
37      gui_State.gui_Callback = str2func(varargin{1});
38  end
39
40  if nargout
41      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
42  else
43      gui_mainfcn(gui_State, varargin{:});
44  end
45  % End initialization code - DO NOT EDIT
46
47
48  % --- Executes just before rs232gui is made visible.
49  function rs232gui_OpeningFcn(hObject, eventdata, handles, varargin)
50  % This function has no output args, see OutputFcn.
51  % hObject    handle to figure
52  % eventdata  reserved - to be defined in a future version of MATLAB
53  % handles    structure with handles and user data (see GUIDATA)
54  % varargin   command line arguments to rs232gui (see VARARGIN)
55
56  % Choose default command line output for rs232gui
57  handles.output = hObject;
58  % Update handles structure
59  guidata(hObject, handles);
60
61  handles.SerialCOM = hObject;       % Cree une variable global pour la classe JAVA serial
62  guidata(hObject, handles);
63
64  handles.FlagRun = hObject;     % Cree un flag pour debut/stop running
65  guidata(hObject, handles);
66
67  handles.FlagResetBuffer = hObject;      % Cree un flag pour debut/stop running
68  guidata(hObject, handles);
69
70
71  % UIWAIT makes rs232gui wait for user response (see UIRESUME)
72  % uiwait(handles.figure1);
73
74
75  % --- Outputs from this function are returned to the command line.
76  function varargout = rs232gui_OutputFcn(hObject, eventdata, handles)
77  % varargout  cell array for returning output args (see VARARGOUT);
78  % hObject    handle to figure
79  % eventdata  reserved - to be defined in a future version of MATLAB
80  % handles    structure with handles and user data (see GUIDATA)
81
82  % Get default command line output from handles structure
83  varargout{1} = handles.output;
84
85
86  % --- Executes on button press in pushConnexion.
87  function pushConnexion_Callback(hObject, eventdata, handles)
88  % hObject    handle to pushConnexion (see GCBO)
89  % eventdata  reserved - to be defined in a future version of MATLAB
90  % handles    structure with handles and user data (see GUIDATA)
91
92  numPortCOM = get(handles.popupCOM, 'Value'); % Recupere le port COM
93  listPortCOM = get(handles.popupCOM, 'String');
94  PortCOM = listPortCOM(numPortCOM);
95
96  numVitesseCOM = get(handles.popupVitesse, 'Value');    % Recupere le BaudRate
97  listVitesseCOM = get(handles.popupVitesse, 'String');
98  BaudRate = listVitesseCOM(numVitesseCOM);
99
100 %disp([PortCOM BaudRate]);
101
102 comresiduel = instrfind;        % matrice des ports com ouverts
103 if (~isempty(comresiduel))          % on les ferme tous
104     fclose(comresiduel);            % Nettoyage si necessaire
105     delete(comresiduel);
106 end
107
108 %if  (length(handles.SerialCOM) == 0)              % test si connexion deja etabli
109 handles.SerialCOM = serial(PortCOM,'baudrate',str2double([BaudRate]));
110 handles.SerialCOM.InputBufferSize = 1024 * 1000;
111 handles.SerialCOM.ReadAsyncMode = 'manual';
112 handles.SerialCOM.Terminator = '';              % Caractere de fin
113 fopen(handles.SerialCOM);
114 fprintf(handles.SerialCOM,'qqchose');
115 handles.SerialCOM.ReadAsyncMode = 'continuous';
116 %end
117
118 % guidata(gcbo,handles);
119 % handles = guidata(gcbo);
120 % clc;
121 % disp(handles.SerialCOM);
122 % if ~(strncmp(get(handles.SerialCOM,Status),'open',4))
123 %         delete(handles.SerialCOM);                  % Si pas de connexion, supprime la classe JAVA
124 %         clear handles.SerialCOM;
125 %         error('Erreur lors de la connexion')
126 % end
127
128 guidata(gcbf,handles);
129
130 disp([PortCOM BaudRate]);
```

```
131
132
133
134 % --- Executes on button press in pushSave.
135 function pushSave_Callback(hObject, eventdata, handles)
136 % hObject    handle to pushSave (see GCBO)
137 % eventdata reserved - to be defined in a future version of MATLAB
138 % handles    structure with handles and user data (see GUIDATA)
139
140 variableglobal = get(handles.VarName,'string'); % nom de la variable tableau des valeurs
141 filename = get(handles.FileName,'string');       % nom du fichier
142 command = ['save ' filename ' ' variableglobal ' t_' variableglobal ';'];
143 evalin('base',command);
144
145
146
147 % --- Executes on button press in pushStart.
148 function pushStart_Callback(hObject, eventdata, handles)
149 % hObject    handle to pushStart (see GCBO)
150 % eventdata reserved - to be defined in a future version of MATLAB
151 % handles    structure with handles and user data (see GUIDATA)
152
153 handles.FlagRun = 1;
154 guidata(gcbf,handles);
155
156 %disp(get(handles.SerialCOM));
157 if (length(instrfind) == 1)                  % si un port est déjà défini,
158     disp('redefinition SerialCOM');
159     handles.SerialCOM = instrfind;                    % on est sure que c'est SerialCOM
160     guidata(gcbf,handles);
161 end
162
163 if (~isempty(handles.SerialCOM))                % si il existe une connexion
164
165     variableglobal = get(handles.VarName,'string');              % nom de la variable tableau des valeurs
166     command = ['clear ' variableglobal ';global ' variableglobal];
167     evalin('base',command);                           % declare variable global dans le workspace
168     eval(command);                                    % dans la fonction
169
170     variableglobalTps = ['t_' variableglobal];
171     command = ['clear ' variableglobalTps ';global ' variableglobalTps];
172     evalin('base',command);                           % declare variable global dans le workspace
173     eval(command);                                    % dans la fonction
174
175
176     variableglobalinter = get(handles.VarNameInter,'string');   % nom de la variable tableau des valeurs
177     command = ['clear ' variableglobalinter ';global ' variableglobalinter];
178     evalin('base',command);                           % declare variable global dans le workspace
179     eval(command);                                    % dans la fonction
180
181     variableglobalinterTps = ['t_' variableglobalinter];
182     command = ['clear ' variableglobalinterTps ';global ' variableglobalinterTps];
183     evalin('base',command);                           % declare variable global dans le workspace
184     eval(command);                                    % dans la fonction
185
186     R = [];
187     t_R = [];
188
189     n = handles.SerialCOM.bytesAvailable;                % vide la cache
190     if (n > 0)
191         s = fread(handles.SerialCOM,n);
192     end
193     tic                                  % Initialisation base de tps
194     TpsBoucle = 0;                       % variable intermediaire pour calcul tps boucle
195     TpsSr = 0;
196     tps = eps;
197
198
199     ValDernierNoeud = 0;
200     sr = [];
201
202     VarDelay = (str2num(get(handles.VarDelay,'String')));
203     pause(VarDelay);
204     while(handles.FlagRun == 1)
205         if (handles.FlagResetBuffer == 1)
206             R = [];
207             t_R = [];
208             t_Rn = [];
209             Rn = [];
210             handles.FlagResetBuffer = 0;
211             guidata(gcbf,handles);
212         end
213
214
215         n = handles.SerialCOM.bytesAvailable;
216         if (n > 26)
217             s = fread(handles.SerialCOM,n);
218             tps =toc - TpsBoucle;
219             TpsBoucleAnc = TpsBoucle;
220             TpsBoucle = toc;            % prend le tps
221             set(handles.text_buffer,'String',n);
222             set(handles.text_vitesse,'String',n*10/tps);
223
224             Tpss = TpsBoucle + ( (TpsBoucle-TpsBoucleAnc)/n );
225 %disp([TpsSr Tpss]);
```

```
226             [Rn,t_Rn,sr,ValDernierNoeud,TpsSr] = RS232_analyzBloc([sr s'],ValDernierNoeud,TpsSr,Tpss);
227 %disp(Tpss);
228
229
230             [ln cn]=size(Rn);
231             [l c]=size(R);
232
233             t_R(l+1:l+ln,1) = t_Rn;
234
235             set(handles.text_sizeBuffer,'String',ln);
236
237             if (cn ~= c)  % si besion de redimensionner une matrice
238                 K = zeros(ln+l , max(cn,c))*NaN;
239                 K(1:l,1:c) = R;
240                 K(l+1:l+ln , 1:cn) = Rn;
241                 R = K;
242             else
243                 R(l+1:l+ln , 1:cn) = Rn;
244             end
245
246             VarBufferSize = (str2num(get(handles.VarBufferSize,'String')));
247             if ((ln+l) > VarBufferSize)
248                 R = R(l+ln-VarBufferSize+1:l+ln , 1:cn);
249                 t_R = t_R(l+ln-VarBufferSize+1:l+ln);
250             end
251
252             [l c]=size(R);
253             set(handles.text_buffline,'String',l);
254             set(handles.text_buffcol,'String',c);
255
256
257             VarBufferInterSize = (str2num(get(handles.VarBufferInterSize,'String')));
258             if (l > VarBufferInterSize)
259                 command = [variableglobalinter '= R(' num2str(l-VarBufferInterSize+1) ':' num2str(l) ',:);' ];
260                 command = [command variableglobalinterTps '=t_R(' num2str(l-VarBufferInterSize+1) ':' num2str(l) ');'];
261             else
262                 command = [variableglobalinter '= R;' ];
263                 command = [command variableglobalinterTps '= t_R;'];
264             end
265
266             eval(command);                      % exporte le tableauIntermediaire dans le workspace
267
268             command = [variableglobal '= R;'];
269             command = [command variableglobalTps '=t_R;'];
270             eval(command);                      % exporte le tableau dans le workspace
271
272             command = get(handles.inLoopCommand,'String');
273             [nbrcommand,tmp] = size(command);
274
275
276
277             for i = 1:nbrcommand
278                 command2 = char(command(i,:));      % convert cell to char
279                 evalin('base',command2);
280             end
281
282
283             %             axes(handles.axes1);
284             %
285             %             plot(Rn(~isnan(Rn(:,1)),1));
286             %             hold on;
287             %             plot(Rn(~isnan(Rn(:,2)),2));
288             %             hold off;
289             %             drawnow;
290         end % if n > 25
291         set(handles.text_vitesse,'String',n*10/tps);
292         str_text_buffer = get(handles.text_buffer,'String');
293         if n == 0
294             switch str_text_buffer
295                 case '|'
296                     newtext_buffer = '\';
297                 case '\'
298                     newtext_buffer = '-';
299                 case '-'
300                     newtext_buffer = '/';
301                 otherwise
302                     newtext_buffer = '|';
303             end
304             set(handles.text_buffer,'String',newtext_buffer);
305         else
306             set(handles.text_buffer,'String',n);
307
308         end
309     VarDelay = (str2num(get(handles.VarDelay,'String')));
310         pause(VarDelay)
311         handles = guidata(gcbo);    % recupere la condition de stop : handles.FlagRun = 0!
312
313     end % while(handles.FlatRun == 1)
314
315     %    command = [variableglobal '= R;']
316     %    eval(command);                      % exporte le tableau dans le workspace
317
318 else % si existe une connexion ( ~isempty(handles.SerialCOM))
319     error('Pas de connexion ouverte');
320 end % si existe une connexion ( ~isempty(handles.SerialCOM))
```

```
321
322
323
324
325
326 % --- Executes on button press in pushStop.
327 function pushStop_Callback(hObject, eventdata, handles)
328 % hObject    handle to pushStop (see GCBO)
329 % eventdata reserved - to be defined in a future version of MATLAB
330 % handles    structure with handles and user data (see GUIDATA)
331 handles.FlagRun = 0;
332 guidata(gcbf,handles);
333
334 % --- Executes on button press in pushReset.
335 function pushReset_Callback(hObject, eventdata, handles)
336 % hObject    handle to pushReset (see GCBO)
337 % eventdata reserved - to be defined in a future version of MATLAB
338 % handles    structure with handles and user data (see GUIDATA)
339
340 comresiduel = instrfind;        % matrice des ports com ouverts
341 if (~isempty(comresiduel))
342     fclose(comresiduel);              % Nettoyage si necessaire
343     delete(comresiduel);
344 end
345
346 handles.SerialCOM = [];
347 guidata(gcbf,handles);
348
349 % --- Executes on button press in pushplotBuffer.
350 function pushplotBuffer_Callback(hObject, eventdata, handles)
351 % hObject    handle to pushplotBuffer (see GCBO)
352 % eventdata reserved - to be defined in a future version of MATLAB
353 % handles    structure with handles and user data (see GUIDATA)
354
355 variableglobal = get(handles.VarName,'string'); % nom de la variable tableau des valeurs
356
357 figure(50);
358 command = ['plot(t_' variableglobal ',' variableglobal '); axis tight;' ];
359 evalin('base',command);
360
361 % --- Executes on button press in pushResetBuffer.
362 function pushResetBuffer_Callback(hObject, eventdata, handles)
363 % hObject    handle to pushResetBuffer (see GCBO)
364 % eventdata reserved - to be defined in a future version of MATLAB
365 % handles    structure with handles and user data (see GUIDATA)
366
367 handles.FlagResetBuffer = 1;      % flag reset buffer
368 guidata(gcbf,handles);
369
370
371 %--------------------------------------------------------------------------------------------
372 %--------------------------------------------------------------------------------------------
373 %--Fcts non utilisés-------------------------------------------------------------
374 %--------------------------------------------------------------------------------------------
375 %--------------------------------------------------------------------------------------------
376
377 % --- Executes during object creation, after setting all properties.
378 function popupCOM_CreateFcn(hObject, eventdata, handles)
379 % hObject    handle to popupCOM (see GCBO)
380 % eventdata reserved - to be defined in a future version of MATLAB
381 % handles    empty - handles not created until after all CreateFcns called
382
383 % Hint: listbox controls usually have a white background on Windows.
384 %       See ISPC and COMPUTER.
385 if ispc
386     set(hObject,'BackgroundColor','white');
387 else
388     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
389 end
390
391
392 % --- Executes on selection change in popupCOM.
393 function popupCOM_Callback(hObject, eventdata, handles)
394 % hObject    handle to popupCOM (see GCBO)
395 % eventdata reserved - to be defined in a future version of MATLAB
396 % handles    structure with handles and user data (see GUIDATA)
397
398 % Hints: contents = get(hObject,'String') returns popupCOM contents as cell array
399 %        contents{get(hObject,'Value')} returns selected item from popupCOM
400
401
402 % --- Executes during object creation, after setting all properties.
403 function popupVitesse_CreateFcn(hObject, eventdata, handles)
404 % hObject    handle to popupVitesse (see GCBO)
405 % eventdata reserved - to be defined in a future version of MATLAB
406 % handles    empty - handles not created until after all CreateFcns called
407
408 % Hint: listbox controls usually have a white background on Windows.
409 %       See ISPC and COMPUTER.
410 if ispc
411     set(hObject,'BackgroundColor','white');
412 else
413     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
414 end
415
```

```
416
417 % --- Executes on selection change in popupVitesse.
418 function popupVitesse_Callback(hObject, eventdata, handles)
419 % hObject    handle to popupVitesse (see GCBO)
420 % eventdata reserved - to be defined in a future version of MATLAB
421 % handles    structure with handles and user data (see GUIDATA)
422
423 % Hints: contents = get(hObject,'String') returns popupVitesse contents as cell array
424 %        contents{get(hObject,'Value')} returns selected item from popupVitesse
425
426
427 % --- Executes during object creation, after setting all properties.
428 function VarName_CreateFcn(hObject, eventdata, handles)
429 % hObject    handle to VarName (see GCBO)
430 % eventdata reserved - to be defined in a future version of MATLAB
431 % handles    empty - handles not created until after all CreateFcns called
432
433 % Hint: edit controls usually have a white background on Windows.
434 %       See ISPC and COMPUTER.
435 if ispc
436     set(hObject,'BackgroundColor','white');
437 else
438     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
439 end
440
441
442
443 function VarName_Callback(hObject, eventdata, handles)
444 % hObject    handle to VarName (see GCBO)
445 % eventdata reserved - to be defined in a future version of MATLAB
446 % handles    structure with handles and user data (see GUIDATA)
447
448 % Hints: get(hObject,'String') returns contents of VarName as text
449 %        str2double(get(hObject,'String')) returns contents of VarName as a double
450
451
452 % --- Executes during object creation, after setting all properties.
453 function FileName_CreateFcn(hObject, eventdata, handles)
454 % hObject    handle to FileName (see GCBO)
455 % eventdata reserved - to be defined in a future version of MATLAB
456 % handles    empty - handles not created until after all CreateFcns called
457
458 % Hint: edit controls usually have a white background on Windows.
459 %       See ISPC and COMPUTER.
460 if ispc
461     set(hObject,'BackgroundColor','white');
462 else
463     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
464 end
465
466
467
468 function FileName_Callback(hObject, eventdata, handles)
469 % hObject    handle to FileName (see GCBO)
470 % eventdata reserved - to be defined in a future version of MATLAB
471 % handles    structure with handles and user data (see GUIDATA)
472
473 % Hints: get(hObject,'String') returns contents of FileName as text
474 %        str2double(get(hObject,'String')) returns contents of FileName as a double
475
476
477
478 % --- Executes during object creation, after setting all properties.
479 function VarBufferSize_CreateFcn(hObject, eventdata, handles)
480 % hObject    handle to VarBufferSize (see GCBO)
481 % eventdata reserved - to be defined in a future version of MATLAB
482 % handles    empty - handles not created until after all CreateFcns called
483
484 % Hint: edit controls usually have a white background on Windows.
485 %       See ISPC and COMPUTER.
486 if ispc
487     set(hObject,'BackgroundColor','white');
488 else
489     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
490 end
491
492
493
494 function VarBufferSize_Callback(hObject, eventdata, handles)
495 % hObject    handle to VarBufferSize (see GCBO)
496 % eventdata reserved - to be defined in a future version of MATLAB
497 % handles    structure with handles and user data (see GUIDATA)
498
499 % Hints: get(hObject,'String') returns contents of VarBufferSize as text
500 %        str2double(get(hObject,'String')) returns contents of VarBufferSize as a double
501
502
503 % --- Executes during object creation, after setting all properties.
504 function inLoopCommand_CreateFcn(hObject, eventdata, handles)
505 % hObject    handle to inLoopCommand (see GCBO)
506 % eventdata reserved - to be defined in a future version of MATLAB
507 % handles    empty - handles not created until after all CreateFcns called
508
509 % Hint: edit controls usually have a white background on Windows.
510 %       See ISPC and COMPUTER.
```

```
511 if ispc
512     set(hObject,'BackgroundColor','white');
513 else
514     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
515 end
516
517
518
519 function inLoopCommand_Callback(hObject, eventdata, handles)
520 % hObject    handle to inLoopCommand (see GCBO)
521 % eventdata  reserved - to be defined in a future version of MATLAB
522 % handles    structure with handles and user data (see GUIDATA)
523
524 % Hints: get(hObject,'String') returns contents of inLoopCommand as text
525 %        str2double(get(hObject,'String')) returns contents of inLoopCommand as a double
526
527
528
529
530 % --- Executes during object creation, after setting all properties.
531 function VarNameInter_CreateFcn(hObject, eventdata, handles)
532 % hObject    handle to VarNameInter (see GCBO)
533 % eventdata  reserved - to be defined in a future version of MATLAB
534 % handles    empty - handles not created until after all CreateFcns called
535
536 % Hint: edit controls usually have a white background on Windows.
537 %       See ISPC and COMPUTER.
538 if ispc
539     set(hObject,'BackgroundColor','white');
540 else
541     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
542 end
543
544
545
546 function VarNameInter_Callback(hObject, eventdata, handles)
547 % hObject    handle to VarNameInter (see GCBO)
548 % eventdata  reserved - to be defined in a future version of MATLAB
549 % handles    structure with handles and user data (see GUIDATA)
550
551 % Hints: get(hObject,'String') returns contents of VarNameInter as text
552 %        str2double(get(hObject,'String')) returns contents of VarNameInter as a double
553
554
555 % --- Executes during object creation, after setting all properties.
556 function VarBufferInterSize_CreateFcn(hObject, eventdata, handles)
557 % hObject    handle to VarBufferInterSize (see GCBO)
558 % eventdata  reserved - to be defined in a future version of MATLAB
559 % handles    empty - handles not created until after all CreateFcns called
560
561 % Hint: edit controls usually have a white background on Windows.
562 %       See ISPC and COMPUTER.
563 if ispc
564     set(hObject,'BackgroundColor','white');
565 else
566     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
567 end
568
569
570
571 function VarBufferInterSize_Callback(hObject, eventdata, handles)
572 % hObject    handle to VarBufferInterSize (see GCBO)
573 % eventdata  reserved - to be defined in a future version of MATLAB
574 % handles    structure with handles and user data (see GUIDATA)
575
576 % Hints: get(hObject,'String') returns contents of VarBufferInterSize as text
577 %        str2double(get(hObject,'String')) returns contents of VarBufferInterSize as a double
578
579
580 % --- Executes during object creation, after setting all properties.
581 function VarDelay_CreateFcn(hObject, eventdata, handles)
582 % hObject    handle to VarDelay (see GCBO)
583 % eventdata  reserved - to be defined in a future version of MATLAB
584 % handles    empty - handles not created until after all CreateFcns called
585
586 % Hint: edit controls usually have a white background on Windows.
587 %       See ISPC and COMPUTER.
588 if ispc
589     set(hObject,'BackgroundColor','white');
590 else
591     set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
592 end
593
594
595
596 function VarDelay_Callback(hObject, eventdata, handles)
597 % hObject    handle to VarDelay (see GCBO)
598 % eventdata  reserved - to be defined in a future version of MATLAB
599 % handles    structure with handles and user data (see GUIDATA)
600
601 % Hints: get(hObject,'String') returns contents of VarDelay as text
602 %        str2double(get(hObject,'String')) returns contents of VarDelay as a double
603
604
605 % --- Executes on button press in ProgDefault.
```

```
606 function ProgDefault_Callback(hObject, eventdata, handles)
607
608 Command = [ {'figure(50);'}
609             {'plot(t_Rn,Rn);'}
610             {'axis tight;' }];
611 set(handles.inLoopCommand,'String',Command);
612
613 % --- Executes on button press in ProgDefaultSigne.
614 function ProgDefaultSigne_Callback(hObject, eventdata, handles)
615 Command = [ {'figure(50);'}
616             {'idx = find(Rn > 65535 / 2);'}
617             {'Rn(idx)= Rn(idx)-65536;'}
618             {'plot(t_Rn,Rn);'}
619             {'axis tight;' }];
620 set(handles.inLoopCommand,'String',Command);
621
622
623 % --- Executes on button press in ProgSimulink.
624 function ProgSimulink_Callback(hObject, eventdata, handles)
625
626 Command = [ {'[Rnfull t_Rnfull] = padr(Rn,1,t_Rn);'}
627             {'[l c] = size(Rnfull);'}
628             {''}
629             {'simin.signals.values = Rnfull;'}
630             {'simin.signals.dimensions = c;'}
631             {'simin.time = t  Rnfull - min(t  Rnfull);'}
632             {''}
633             {'[T,X Y1] = sim(''untitled'',[min(simin.time) max(simin.time)],'''',simin);'}
634             {''}
635             {'figure(50);'}
636             {'plot(T,Y1);'}
637             {'axis tight;' }];
638
639 set(handles.inLoopCommand,'String',Command);
```

Listing 6: *rs232gui.m*