Rapport de stage S2
-
Planeur Autonome

Julien Le Granvalet

Promotion 2012 - ESSE

3 octobre 2011

Remerciements

Je tiens à remercier tous ceux qui m'ont aidé durant ce stage, en particulier M. Luc JAULIN professeur à l'ENSTA Bretagne, mon maître de stage, pour son aide, ses conseils et sa disponibilité tout au long du stage.

Je remercie également M. Fabrice LE BARS et M. Jan SLIWKA, doctorants à l'ENSTA Bretagne, M. Yvon GALLOU professeur à l'ENSTA Bretagne et les autres étudiants stagiaires travaillant sur des projets robotiques pour leur aide et leur disponibilité.

Résumé

Au cours de ces dernières années, on a vu l'utilisation de systèmes de drones se développer. En effet, les possibilités et l'efficacité déjà prouvée de tels systèmes en font des sujets de recherche de plus en plus importants. Les drones affichent de plus de nombreux avantages. Tout d'abord, ils permettent d'éviter de mettre en danger la vie d'un équipage et peuvent accomplir des missions qui seraient éprouvantes pour un humain comme des missions de surveillance de longue durée. C'est pourquoi il m'a paru intéressant de concevoir un robot planeur qui serait de plus économe en énergie de propulsion. Il utiliserait les courants ascendants pour poursuivre son vol aussi longtemps que possible et chercherait de plus à suivre un parcours déterminé.

Mon étude s'est divisée en 5 parties. J'ai tout d'abord modélisé le planeur sous la forme d'équations d'état. J'ai ensuite fait le choix d'un régulateur permettant au planeur d'effectuer ce qu'on l'on attendait de lui. Puis, j'ai programmé un simulateur pour tester la modélisation. Ensuite, j'ai choisi un modèle réduit de planeur à construire répondant aux critères de simplicité de construction et de mise en œuvre. Il a ensuite fallu créer une partie électronique permettant de réaliser la commande et la régulation du planeur. La partie expérimentale a été l'une des parties les plus importantes du projet. Elle a en effet permis de valider les travaux des parties précédentes et de les améliorer. Mais elle a surtout permis d'évaluer les capacités que l'on peut attendre d'un tel robot.

Enfin, j'ai profité de ce stage pour découvrir et analyser l'organisation d'un laboratoire de recherche.

Abstract

Nowadays, UAV systems are getting more and more used and popular. They offer many advantages. Indeed, they do not endanger the life of a crew and can carry out long duration missions. They also demonstrated their effectiveness especially in military field. However, it seemed interesting to create an energy recourse friendly UAV able to complete several missions as others UAVs.

Therefore the main purpose of my internship was to achieve a simple aerial system, autonomous able to acquire data in a specific area. Thus the study has brought to the realization of an autonomous glider. The glider must be able to reach a defined point and the final goal is to take advantage of updrafts to define its own path and fly as long as possible.

My study was divided into five parts. I first model the problem. Then, I chose a regulator allowing the glider to perform what was expected. Then, I programmed a simulator to test the model. After that, I selected a model of glider easy to build and simple to use. Then, I had to create the electronic part that will achieve the regulation. The experimental part was one of the most important part of the project. Indeed, it permitted to validate the tasks made in the previous parts and to improve them. But, it mainly assessed the capability of the glider.

I could also discover and analyze the organization of a research laboratory.

Table des matières

Introduction

Dans le cadre de la formation à l'ENSTA Bretagne, un stage technique d'assistant ingénieur est à réaliser. Il vise à mettre en pratique les connaissances acquises lors des deux premières années de formation, mais également de les approfondir en vue de la spécialisation de la troisième année. Ce stage est de plus l'occasion d'apporter un avis critique sur l'organisation d'une entreprise ou d'un centre de recherche et de mieux comprendre les enjeux d'un projet sur lequel on est amené à travailler.

J'ai fait le choix de faire mon stage au pôle STIC de l'ENSTA Bretagne pour poursuivre mon projet industriel de deuxième année. Le but du projet était de réaliser un robot planeur autonome. Ce projet avait apporté de nombreux résultats mais n'avait pas pu être entièrement mené à son terme. C'est pourquoi l'objectif de ce stage était donc de réaliser un robot planeur capable de se maintenir pendant une longue période en vol en profitant des courants ascendants pour gagner de l'altitude. Je pouvais ainsi profiter de l'expérience acquise lors de mon projet industriel dans ce domaine pour avancer le projet. L'étude d'un tel système apparait de plus comme étant très complète faisant appel à des connaissances dans de nombreux domaines comme la mécanique et l'électronique.

Dans une première partie, nous verrons les différentes missions du pôle STIC. Puis, nous verrons dans quel cadre s'inscrit un projet tel que celui du robot planeur. Ensuite, une description de l'activité ainsi que les résultats obtenus durant ce stage seront présentés. Enfin, nous ferons une analyse de l'organisation.

1 Pôle STIC

Le pôle Sciences et Technologies de l'Information et de la Communication (STIC) regroupe 90 personnes, enseignants chercheurs, ingénieurs de recherche et post-doctorants, et doctorants qui travaillent sur des problématiques d'acquisition et de traitement d'information. Les études qui y sont effectuées ont pour thématique principale l'environnement marin. En effet, elles peuvent porter sur la surveillance de cet environnement ainsi que sur sa caractérisation. Le pôle se compose de 4 équipes, l'équipe Acoustique Passive, l'équipe Ingénieurie Dirigée par les Modèles, l'équipe Ocean Sensing an Mapping et l'équipe Radar and Electro-Magnetic Sensing.

1.1 L'équipe Acoustique Passive (AP)

Cette équipe est composée de 8 personnes, enseignants chercheurs, ingénieurs post-doctorants et ingénieurs de recherche. L'objectif de l'équipe Acoustique Passive est de faire l'étude des milieux marins par l'écoute des sons qui en proviennent. En effet, cette approche est un moyen discret mais également efficace d'obtenir des informations sur le milieu en fonction des différentes sources de sons répertoriées et ce pendant une longue période d'observation. Le but de ces études est une meilleure compréhension des milieux marins et cela à des fins environnementale et scientifique mais aussi stratégique. Elles se décomposent en plusieurs thèmes. L'un d'eux étant la caractérisation des sources sonores avec par exemple comme application, la détection, la classification et la localisation de mammifères marins. Un autre thème également étudié est l'interaction des activités humaines en mer avec la faune avec comme application possible : l'incidence du trafic maritime sur les mammifères marins. Ainsi, l'écoute passive implique entre autres des recherches dans le domaine de la propagation des ondes acoustiques et du traitement du signal.

1.2 L'équipe Ingénieurie Dirigée par les Modèles (IDM)

L'équipe IDM est composée de 22 personnes. Les travaux effectués par cette équipe sont basés sur l'étude des modèles qui peuvent être utilisés dans de nombreux domaines pour la représentation de projet ou d'organisation. Ils portent cependant en particulier sur l'étude des langages de modélisation. Les techniques de transformations de modèles ainsi que la réalisation d'analyseurs et la gestion de la complexité font également partie des études menées par l'équipe IDM. Cette équipe est de plus amenée à travailler en contact direct avec le milieu industriel. Les recherches s'effectuent donc en coopération avec des équipes industrielles et universitaires. Les travaux visent à améliorer l'utilisation des techniques de modélisation dans le développement logiciel et industriel.

1.3 L'équipe Ocean Sensing and Mapping (OSM)

L'équipe OSM est constituée de 25 personnes. L'objectif des études menées par cette équipe est de « caractériser les fonds marins à l'aide de capteurs intelligents ». Les capteurs acoustiques permettant l'acquisition des données sont généralement utilisés sur des systèmes embarqués. Les études réalisées mettent en œuvre plusieurs compétences comme la modélisation physique, le traitement de signal et de données, la robotique et la fusion et représentation de données. Les trois axes d'étude principaux sont les capteurs intelligents sous-marins, l'automatisation des systèmes de monitoring et le traitement de l'information et la modélisation. On peut distinguer plusieurs domaines visant à remplir les critères proposés par les axes d'études. Tout d'abord, l'automatique appliqué à la robotique sous-marine. Le stage portait sur un sujet de robotique c'est donc dans ce cadre qu'il s'inscrivait. Mais on y retrouve également les systèmes de perception sous-marine et l'hydrographie. Un des objectifs est donc de faire en sorte que les robots forment des plate-formes capables de recueillir des informations en autonomie. Plusieurs problématiques interviennent alors. Il faut en effet traiter de l'automatique, de la localisation des robots mais également de la gestion de meutes de robots. L'équipe participe de plus à des concours de robotique sous-marine comme le concours *SAUC-E*. Une autre partie traite également de la caractérisation des capteurs et de l'hydrographie.

1.4 L'équipe Radar and Electro-Magnetic Sensing (REMS)

L'équipe REMS est composée de 25 chercheurs. Sa mission principale est de modéliser les interactions des ondes électromagnétiques avec l'environnement naturel. Ce qui permet à terme de caractériser l'environnement électromagnétique afin de permettre de l'identification de cibles dans un environnement perturbé. Cela passe par l'étude d'interaction entre des ondes électromagnétiques et le milieu naturel, comme par exemple la surface de la mer. En effet, une modélisation des interactions à ce niveau permet de donner une représentation des perturbations engendrées par ce milieu et ainsi de pouvoir mieux détecter la présence de cibles dans cet environnement. Les travaux réalisés font de de plus intervenir le traitement du signal afin de reconstituer des images radar haute-résolution.

2 Problématisation du thème

Ces dernières années, les recherches technologiques permettant de mieux connaître l'environnement se font de plus en plus nombreuses. Les robots participent en ce sens à l'évolution technologique qui est faite dans ce domaine. Comme présenté précédemment, c'est en effet l'objectif de l'équipe OSM que d'étudier les possibilités d'utilisation de robots dans ce domaine et de les développer. L'objectif du stage est de réaliser un robot planeur capable d'exploiter son environnement afin de se procurer l'énergie nécessaire pour se maintenir en vol. Un tel robot peut répondre à des problématiques similaires posées par l'étude des robots sous-marins qui ont pour but de nous renseigner sur l'environnement dans lequel ils évoluent grâce à leurs capteurs. Le sujet du stage portait essentiellement sur la partie automatique cependant il est envisageable de pourvoir le robot de capteurs, il serait alors capable de recueillir des informations.

De plus, les systèmes de drones [1] sont actuellement de plus en plus utilisés, et ce aussi bien dans le cadre d'applications civiles que militaires. En effet, leur efficacité a largement été démontrée, c'est pourquoi ces dernières années un investissement important a été fait pour développer ces systèmes. Les drones n'affichent pas les inconvénients que peut représenter la présence d'un humain à bord d'un appareil. Tout d'abord, la notion de risque est différente de celle que peut apporter la présence d'un équipage à bord. Cela est particulièrement vrai pour l'utilisation dans le domaine militaire mais aussi dans le domaine civil où certaines missions comme la surveillance, nécessitent une longue période à passer sur zone et qui apparait donc comme physiologiquement difficile. De plus, les moyens technologiques actuels permettent aux drones de réaliser des tâches multiples et complexes pour lesquelles il faudrait un équipage de plusieurs personnes. Les drones sont également reconnus pour leur simplicité d'utilisation. Ils sont en effet utilisables pour de nombreuses missions et il est facile de les envoyer et de les récupérer. Cependant, bien que les drones à vocation militaires aient jusqu'à maintenant fait leurs preuves et voient leur nombre augmenter de jour en jour, les drones civils sont encore assez peu nombreux sur le marché.

C'est pourquoi, il m'a paru intéressant de travailler sur un projet de planeur autonome en projet industriel de deuxième année et de le continuer lors du stage. En effet, un planeur offre l'avantage d'être économe en terme d'énergie. De plus, les robots actuellement utilisés à l'ENSTA Bretagne portaient essentiellement sur le domaine maritime : sous-marins et voilier, terrestre avec robots buggys. Il m'a donc paru intéressant de proposer un robot aérien. En effet, ce stage m'a permis de découvrir et d'acquérir de l'expérience dans ce domaine et de rendre ce projet profitable pour les activités robotiques.

1. Aéronefs sans pilote humain.

3 Réalisation d'un planeur autonome

Le but du stage était de concevoir un robot planeur qui se rendrait autonome en profitant des courants ascendants pour se maintenir en vol. Entre deux phases ascendantes il devrait effectuer un parcours déterminé. Le travail s'est décomposé en cinq parties. Tout d'abord, une approche théorique a permis de déterminer les équations d'état d'un tel robot et d'élaborer les algorithmes de commande et de navigation. Puis, une partie simulation a permis une première validation des tests algorithmes de régulation. Le choix d'un modèle réduit de planeur a par la suite été effectué ainsi que sa construction. Une partie électronique comprenant les différents éléments permettant le pilotage et la régulation du planeur a ensuite été intégrée. Enfin, la partie expérimentation vient valider les éléments des parties précédentes et définit le domaine d'utilisation du planeur. Pour cela, je me suis appuyé sur les résultats précédemment obtenus lors de mon projet industriel de deuxième année portant sur le même sujet. Des précisions sur le matériel utilisé ainsi que sur la réalisation des différentes parties seront apportées en annexes en vue d'une réutilisation.

3.1 Partie théorique

L'approche théorique consiste à représenter le planeur sous forme d'équations d'état qui modélisent alors le système. On peut ensuite choisir à partir de ces équations les méthodes de régulation qui seront adoptées.

3.1.1 Représentation d'état

Dans le cas d'un système mécanique tel que le robot planeur, les équations d'état s'obtiennent par application du principe fondamental de la dynamique. La représentation alors obtenue est :

$$\begin{cases} \dot{x} &= V\cos\alpha\cos\theta \\ \dot{y} &= V\cos\alpha\sin\theta \\ \dot{z} &= V\sin\alpha \\ \dot{\theta} &= Vu_1 \\ \dot{\delta_d} &= u_1 \\ \dot{\delta_p} &= u_2 \\ \dot{V} &= \dfrac{R_z}{m} - g\cos\alpha - f_p\sin\delta_p \\ \dot{\alpha} &= \dfrac{R_z}{mV}\cot\alpha - \dfrac{g\cos^2\alpha}{V\sin\alpha} - f_p\dfrac{\sin\delta_p}{V}\cot\alpha - \dfrac{g}{V} + \dfrac{R_x}{mV\sin\alpha} + \dfrac{f_d\sin\delta_d}{V\sin\alpha} \end{cases}$$

Le vecteur d'état est composé des coordonnées du planeur : x, y et z, lié à son centre de gravité G. Son orientation θ et sa pente α ainsi que les angles que forment les gouvernes avec le fuselage : δ_d pour la gouverne de direction et δ_p pour la gouverne de profondeur, apparaissent également dans le vecteur d'état. Celui-ci comprend également la vitesse V du planeur. Elles vont permettre de fournir un modèle pour simuler les régulateurs que l'on souhaite appliquer au planeur.

3.1.2 Régulation

Le planeur doit pouvoir profiter des courants ascendants pour réaliser un parcours. Pour cela, plusieurs types de régulation ont été mis en place. Tout d'abord, le planeur est régulé en cap et également suivant sa pente de descente ou de montée en fonction de la phase du vol. En effet, en considérant qu'un courant ascendant occupe une partie de l'espace définie et restreinte, le planeur doit être capable de rester dans le courant afin d'en profiter. Une fois son ascension terminée, il doit pouvoir utiliser l'altitude acquise pour se diriger vers un point défini et d'effectuer ainsi un parcours avant de rejoindre un nouveau courant ascendant. Il a donc fallu mettre en place une régulation permettant au robot de se maintenir dans le courant ascendant et un régulateur lui permettant de rejoindre un point de coordonnées géographiques en ligne droite. Pour que le planeur puisse profiter pleinement de l'ascendance, il a été fait le choix de lui faire décrire des cercles de rayon constant autour d'un point qui serait le centre du courant. La montée s'effectuerait alors sous la forme d'une spirale ascendante. De plus, les phases de vol plané s'effectuent suivant un plan de descente rendant la finesse [1] maximale.

Rejoindre un point en ligne droite

Le planeur évolue en allant de point en point, définis par leurs coordonnées géographiques. Pour aller d'un point à un autre il se dirige en ligne droite. Pour cela, le cap permettant au robot de rejoindre son point d'arrivée est calculé, ainsi que la distance à ce point (annexes B.1 et B.2). Le planeur est alors capable de suivre un cap à l'aide d'une régulation à commande proportionnelle.

Cependant, un simple suivi de cap tel que celui-ci ne permet pas de rejoindre précisément un point. En effet, pour effectuer les calculs aboutissants à la commande à appliquer à la gouverne de direction, on se sert des informations sur la position du planeur et du cap renvoyées par le GPS qui ne transmet des données actualisées que toutes les secondes. Le vent et les différentes perturbations que peut rencontrer le planeur ajoutés au manque de précision dans les informations de localisation le fait dériver et l'écarte donc de la route directe qu'il devrait suivre. Pour remédier à ce problème, il a fallu définir une distance à la route directe que l'on pourrait qualifier de critique au delà de laquelle le planeur se voit redirigé vers sa route, comme cela est expliqué en annexe B.3. Cette partie de régulation s'ajoute à la précédente pour former le régulateur final qui tient compte du cap suivi et de la distance à la route directe. Le planeur peut alors suivre une route en ligne directe malgré les perturbations du milieu dans lequel il évolue.

Rejoindre un cercle

Comme énoncé précédemment, on a considéré que les courants ascendants sont localisés autour d'un point qui en serait alors leur centre et que le planeur doit évoluer en cercle dans cet espace pour prendre de l'altitude. Pour cela, les courants ont été représentés par

1. La finesse est le rapport entre la portance et la traînée. Elle permet également de définir la distance maximale franchissable en vol plané lorsqu'elle est maximale.

un champ de vecteurs donné par :

$$h : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \frac{1}{\sqrt{x^2+y^2}} \begin{pmatrix} x \arctan(\rho(r_0 - \sqrt{x^2+y^2})) + y \\ y \arctan(\rho(r_0 - \sqrt{x^2+y^2})) - x \end{pmatrix}$$

Ici, x et y représentent respectivement les distances du planeur au centre du cercle suivant les abscisses et les ordonnées. r_0 correspond au rayon du cercle et ρ est le coefficient définissant la vitesse à laquelle le planeur tend à se rapprocher du cercle.

Cette régulation utilise la régulation en cap simple présentée précédemment. En effet, étant donnée la fonction représentant le champ de vecteurs, chaque vecteur indique la direction à suivre pour rejoindre le cercle, on peut donc en déduire un cap à suivre pour le planeur pour rejoindre le cercle. Ainsi quelque soit sa position, le planeur définit son cap à partir du calcul du vecteur correspondant à sa position actuelle. Pour entrer en phase ascendante le planeur se dirige donc vers le cercle centré sur le courant ascendant et gagne de l'altitude en cerclant autour de cette position.

3.2 Partie simulation

La simulation vise à valider la régulation théorique choisie en accord avec la représentation d'état qui modélise le système. Tout d'abord, il a fallu modéliser le planeur en 3D, puis l'intégrer à un simulateur en lui affectant les équations d'état. On a pu alors appliquer le régulateur au modèle et simuler le comportement du planeur.

3.2.1 Modélisation du planeur

Pour créer le planeur en 3D, on a fait le choix d'utiliser le logiciel d'infographie 3D *Blender*. C'est un logiciel gratuit, très performant qui permet de créer soit-même les formes que l'on souhaite donner à notre objet à l'aide d'outils puissants. L'objet créé sous *Blender* avant d'être intégré est représenté en figure 3.1.

Il a par la suite pu être utilisé dans le simulateur en étant représenté grâce à la librairie *OpenGL*. En effet, un script s'exécutant en *perl* disponible sur internet a permis à partir de l'objet généré par *Blender* d'obtenir sa représentation équivalente utilisant *OpenGL*.

3.2.2 Le simulateur

Le simulateur avait pour but de tester le modèle défini afin de voir s'il apparait réaliste et exploitable. De plus, il permet de valider les différents régulateurs choisis. La programmation d'un tel simulateur s'effectue donc dans le langage C++ et grâce à la bibliothèque *Qt* qu'il est possible d'utiliser avec l'environnement *Qt Creator*. Cela permet ainsi de réaliser une interface graphique pour mieux interagir avec le simulateur. Pour réaliser le simulateur, je me suis basé sur les travaux de Mohamed Saad IBN SEDDIK, élève en troisième année en option IASE également stagiaire à l'ENSTA Bretagne pendant cette période, qui a réalisé un simulateur pour un robot hydroglisseur. Cependant, le simulateur

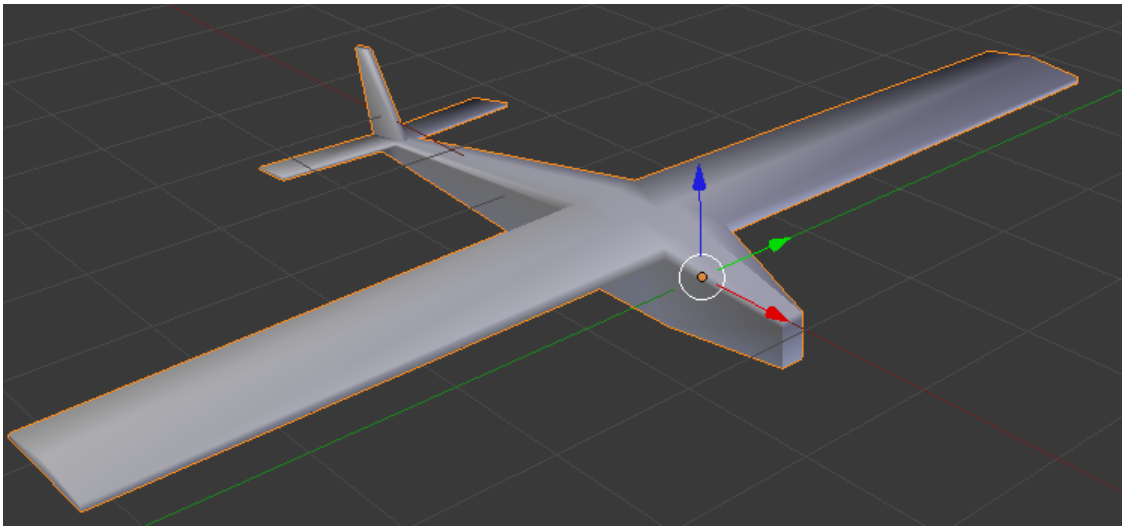

FIGURE 3.1 – Modélisation du planeur

du planeur n'a pu être terminé et les équations d'états n'ont donc pas été validées par une simulation.

3.3 Partie mécanique

Tout d'abord, la réalisation de prototypes tels que des planeurs doit tenir compte de plusieurs paramètres. Ils doivent être légers pour pouvoir espérer rester en l'air pendant une longue période et à la fois robustes pour ne pas être trop endommagés à la suite des chocs auxquels ils pourraient être soumis lors des tests. Il doivent également être de mise en œuvre assez simple pour faciliter leur utilisation. Il peut également être intéressant qu'ils soient faciles et rapides à construire pour pouvoir être reproduits.

3.3.1 Description

Le choix du modèle s'est donc porté vers celui de Jean-Paul THEBAULT, le *POLY 5* d'après un plan de Patrick NICOLAS, disponible sur demande. Ce planeur est apparu facile à construire, il n'utilise que des matériaux aisés à travailler. Le plan est de plus très détaillé et a l'avantage de n'utiliser que des procédés de construction simples, ce qui en fait initialement un modèle facilement réalisable par un modéliste débutant. Il est également rapide à construire, en effet 5 jours de travail suffisent à sa réalisation. Peu de modifications ont été apportées. Il a fallu seulement agrandir la partie centrale du fuselage devant contenir toute la partie électronique (servomoteurs, carte *Arduino*, GPS) que nous verrons par la suite. Il est ainsi possible de part sa conception de répartir le poids de l'électronique au niveau de l'aile ce qui rend le centrage [2] plus évident. Sa mise en œuvre est de plus relativement simple étant donné que la position du matériel est bien définie

2. Le centrage consiste à répartir les charges dans l'appareil afin de respecter les conditions de stabilité et d'équilibre.

pour assurer un centrage correct et qu'une fois les branchements réalisés il ne reste plus qu'à fixer l'aile sur le fuselage à l'aide d'élastiques. Sa robustesse a de plus été démontrée au cours des premiers tests, il n'y eu que peu de dégâts, aisément réparables. Il répond donc finalement aux exigences initiales pour un tel prototype.

Le planeur est contrôlable suivant deux axes : l'axe de lacet et l'axe de tangage. En effet, une gouverne de direction est actionnée et permet au planeur de changer de cap et la gouverne de profondeur permet de le diriger dans le plan vertical. Ce sont les résultats des tests effectués sur de précédents prototypes de mon projet industriel seulement dotés d'un stabilisateur non réglable en vol qui ont montré l'utilité d'une gouverne de profondeur pour mieux maîtriser les plans de descente en vol plané et de montée en phase ascendante.

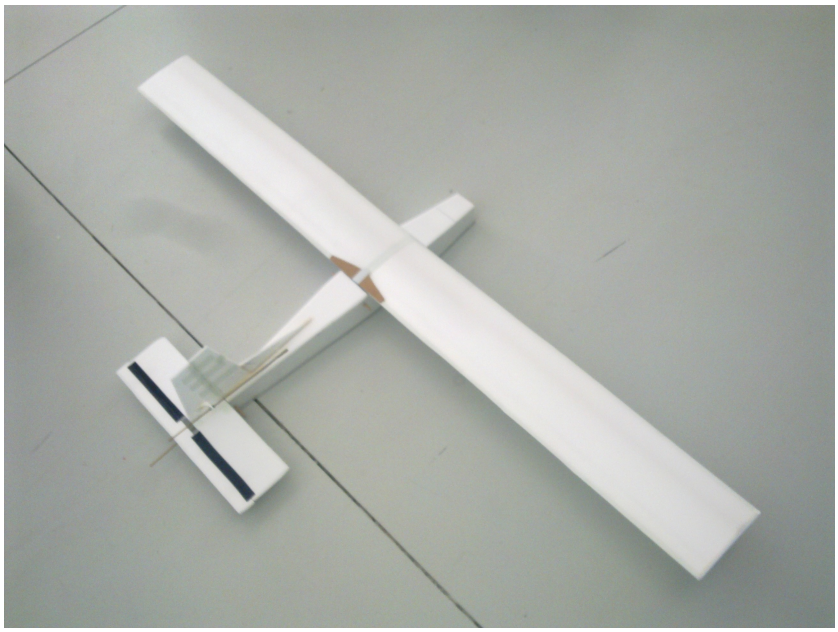

FIGURE 3.2 – Le robot planeur

3.3.2 Caractéristiques

Voici les caractéristiques principales du planeur.

- Envergure : 1600 mm.
- Longueur : 740 mm.
- Poids total : 800 g.
- Autonomie théorique estimée : 1 heure.

3.4 Partie électronique

La partie électronique permet le contrôle du planeur en vol et assure sa régulation. Elle comprend plusieurs éléments, le principal étant la carte *Arduino Uno* qui est une

plate-forme de développement très fonctionnelle. Elle sert d'interface entre les différents actionneurs et contient le code décrivant les algorithmes de régulation. La partie électronique est également composée de deux servomoteurs qui permettent d'actionner les deux gouvernes présentées précédemment. Ils sont connectés à la carte *Arduino* qui assure leur commande. Un module GPS connecté directement à la carte *Arduino* permet d'acquérir de nombreuses informations sur le planeur comme sa position sous forme de coordonnées géographiques, son cap, sa vitesse, son altitude. Les données du GPS sont envoyées à l'*Arduino* et utilisées par le programme de régulation. Une centrale inertielle AHRS (Attitude and Heading Reference System) a également été prévue pour la partie électronique, elle fournit les angles d'Euler du planeur permettant alors de déterminer sa position dans l'espace en temps réel. Il a fallu adapter le code initialement contenu dans la centrale pour que les données traitées soient directement exploitables par le programme de régulation. Il était nécessaire de le modifier pour mettre en place un protocole de communication entre la carte *Arduino* où s'effectuent tous les calculs relatifs à la régulation, et le module AHRS qui effectuent les calculs de navigation inertielle à partir des données brutes de ses capteurs. Les données traitées sont désormais accessibles par le programme de régulation dès que cela est nécessaire. Cependant, la dernière version de la centrale dont nous disposions proposait des capteurs différents de ceux utilisés par le programme réalisé qui correspondait alors à l'ancienne version, de plus nous ne disposions pas du code relatif à ces nouvelles centrales. Ainsi, les modifications du code décrites précédemment concernant la communication entre la centrale et la carte *Arduino* ne purent être effectuées pour la dernière version de centrale. Il ne fut donc pas possible d'intégrer ces nouvelles centrales au planeur. Leur calibration au départ apparaissait de plus difficile, il aurait été de toute façon compliquée de finir leur intégration sur le robot.

La description et le fonctionnement de cette partie électronique sont repris plus en détails en annexe C. La figure 3.3 montre une vue d'ensemble de l'électronique présent dans le planeur ainsi que sa répartition dans le planeur.

3.5 Partie expérimentation

La partie expérimentale occupe une part majeure dans la réalisation d'un projet de robotique. Elle vise à valider l'ensemble des points techniques du projet : les algorithmes, la mécanique et l'électronique. Il y a très souvent de nombreux points parfois prévisibles à améliorer et c'est surtout grâce aux différents tests effectués que l'on peut adapter les algorithmes qui dépendent des éléments extérieurs au cadre réel pour former un système cohérent. Mais cette partie est au final destinée à évaluer les capacités réelles d'un tel robot. Il est en effet important de déterminer ce que l'on peut réellement en attendre pour définir ses utilisations futures. Il est également important de rendre compte de la faisabilité de ce type de système. Cette partie s'est décomposée en plusieurs points. Il a en effet fallu tester la partie mécanique et les algorithmes de régulation avec l'électronique prévu.

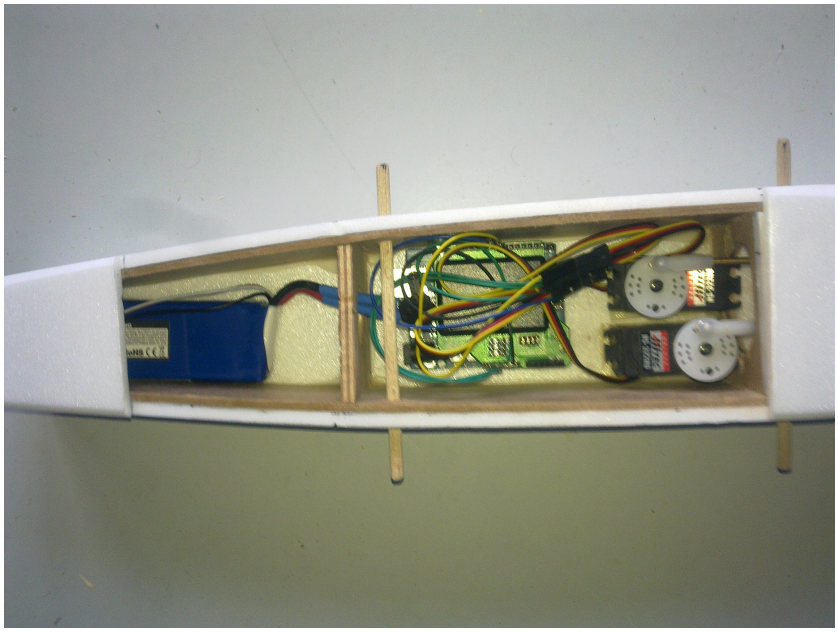

FIGURE 3.3 – L'installation de l'électronique

3.5.1 Test de la partie mécanique

Les tests de la partie mécanique se décomposent en plusieurs parties. Les premiers visaient tout d'abord à observer les possibilités du planeur à voler ainsi que sa réaction aux commandes appliquées sur les gouvernes. Ceux-ci ont été réalisés en radio-commandé. Le planeur était lancé du sol, à partir d'un vélo pour lui donner une vitesse initiale la plus importante possible et qu'il puisse planer sur quelques mètres. Ces premiers essais ont démontré les capacités du planeur à voler et à réagir convenablement aux commandes. Cependant, il est aussi apparu que la vitesse de sustentation minimale du planeur était élevée comme cela avait été également observé lors des essais effectués dans le cadre de mon projet industriel. Cette vitesse pouvait être acquise à la suite d'un lancé à vitesse initiale élevée ou également à la suite d'un lancé d'une hauteur suffisante pour qu'elle puisse l'être lors de sa chute puis de débuter le vol. Disposant d'un moteur adaptable au planeur, la seconde partie des tests consista à utiliser le moteur pour amener le planeur à une bonne altitude et de le couper pour pouvoir l'observer planer et ainsi effectuer les dernières observations relatives à la partie mécanique. Cependant, il est apparu que le planeur étant très léger (800 g) était très sensible au vent et à ses variations. C'est pourquoi en raison d'un vent trop élevé, les tests n'ont pas apporté entière satisfaction et les dernières observations n'ont pu être faites. Il en résulte donc que le domaine d'utilisation de ce planeur reste restreint par les conditions climatiques. En effet, le vent peut aider à mettre l'avion en vol, notamment si on le lance fait au vent mais il doit être très faible pour ne pas le déséquilibrer.

3.5.2 Test des algorithmes de régulation

Les premiers tests on été effectués sur des robots buggys avec l'électronique prévu pour le planeur c'est-à-dire : la carte *Arduino* ainsi que le GPS. Ils ont permis de valider le fonctionnement des algorithmes en conditions réelles. Ils ont également montrer les limites que peuvent avoir ces algorithmes dans certaines conditions d'utilisations.

Pour tester le régulateur permettant de rejoindre un point en ligne, des parcours ont été mis en place avec différents types de trajectoires faisant intervenir de nombreuses possibilités de cap avec des changements de direction suffisamment francs à des vitesses différentes pour évaluer la robustesse des algorithmes. Un exemple de parcours utilisé pour les tests est représenté en figure 3.4, une vidéo de validation de ce parcours est également disponible. Les premiers tests ont notamment permis de fixer des premières valeurs pour les coefficients intervenant dans les commandes proportionnelles, adaptées au robot. Puis, au fur et à mesure des tests, ces valeurs ont été affinées pour aboutir à des valeurs qui rendent la régulation optimale. En effet, les valeurs de ces coefficients apportant le meilleur compromis entre le suivi de cap et l'écart à la route directe ne peuvent être déterminées que par les tests en conditions réelles. Les limites observées d'un tel système proviennent essentiellement du GPS. En effet, l'imprécision du GPS peut aller jusqu'à 2,5 mètres. Dans le cas d'une faible vitesse du robot, entre deux actualisations de ses données, le GPS n'est pas suffisamment précis pour détecter un changement de position du robot, les informations concernant la vitesse et le cap fournies par le GPS ne sont pas correctes. Alors, l'algorithme qui nécessite une information sur le cap ne fonctionne que si le GPS est couplé avec une boussole lorsque son cap n'est pas valide. Cependant, la boussole utilisée doit être compensée pour que le cap qu'elle donne ne soit pas altéré par les mouvements en roulis et tangage du planeur. De plus, le GPS n'actualise ses données que toutes les secondes. Ce qui ne permet pas au planeur d'effectuer des corrections rapides néanmoins sur un parcours de longue distance ce phénomène ne pose pas de problème puisque le robot dispose de suffisamment de temps pour appliquer les corrections nécessaires afin de suivre sa route prévue.

Les tests concernant le régulateur permettant au planeur de rejoindre un cercle se sont déroulés de façon similaire. En effet, le test de cet algorithme visait à vérifier son bon fonctionnement mais aussi à déterminer les coefficients intervenants dans la régulation qui sont fonction du robot. Pour cela, le robot a fait de nombreux départs à différentes positions par rapport au cercle comme le montre la figure 3.5. Pour les tests, le cercle de 10 mètres de rayon est centré sur le terrain. Les résultats ont montré que le cercle était rejoint dans tous les cas, une vidéo validant le test est disponible. Cependant, une fois le cercle intercepté, le robot ne restait pas à la même distance du centre du cercle, l'imprécision pouvait être de 5 mètres. Néanmoins, l'objectif de cet algorithme est rempli, en effet le robot est capable de rester dans un espace restreint en tournant autour d'une position centrale et l'écart de 5 mètres est largement acceptable pour l'utilisation voulue. Cette imprécision est, pour les mêmes raisons que montré précédemment, due à l'imprécision même du GPS et à son temps d'actualisation.

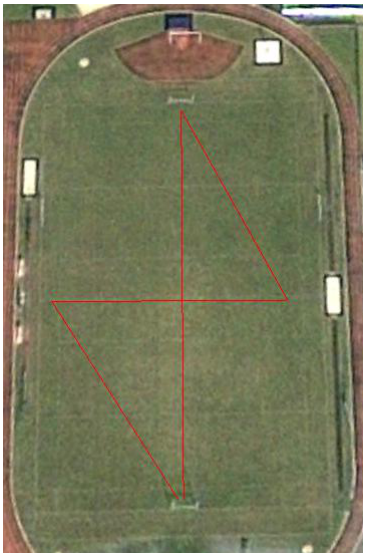

FIGURE 3.4 – Parcours de test du suivi de route

3.5.3 Conclusion

Les observations liées aux différents tests ont permis de définir un domaine d'utilisation de ce type de robot. Tout d'abord, le planeur ne peut-être utilisé que dans des conditions météorologique précises. Le vent doit être faible (moins de 5 nœuds). Une solution qui pourrait être envisagée pour le cas où l'on voudrait que son utilisation soit moins contraignante serait de concevoir un modèle plus lourd mais qui serait en conséquence plus grand pour conserver de bonnes performances en terme de vol plané. De plus, il est apparu indispensable de disposer d'un moyen permettant de mettre en vol le planeur. Il est possible pour cela d'utiliser un lanceur qui permettrait, du sol, de lui imprimer une vitesse initiale suffisante pour qu'il puisse débuter son vol. Mais il est également envisageable de l'utiliser en tant que moto-planeur et se servir d'un moteur uniquement pour le décollage, pour gagner de l'altitude, ce qui peut être également intéressant en l'absence de courants ascendants.

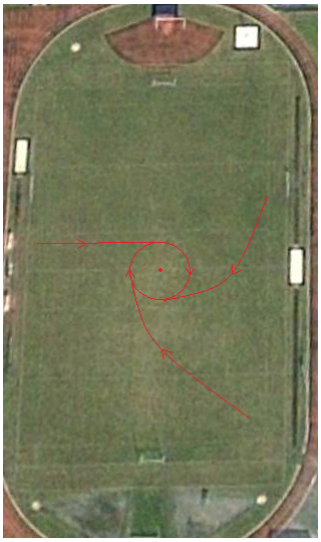

FIGURE 3.5 – Test de la régulation en cercle

4 Analyse de l'organisation

Le cadre dans lequel le stage a été effectué peut s'apparenter à celui d'un centre de recherche. On peut de plus représenter l'école selon le modèle de Mintzberg en considérant également l'aspect universitaire de l'organisation, comme le montre la figure 4.1. En effet, le centre opérationnel représente la majeure partie de l'organisation, il correspond aux chercheurs qui peuvent être des enseignants chercheurs, des ingénieurs post-doctorants, des ingénieurs de recherche, des doctorants. On est dans le cadre d'une organisation professionnelle où les personnes s'accordent parfois entre elles au profit des résultats, ce qui correspond dans certains cas à de l'ajustement mutuel. Ils participent tous à l'activité principale de l'école en dehors de l'enseignement : la recherche. La ligne hiérarchique constituée des intermédiaires faisant le lien entre le sommet stratégique et de la production peut être représentée avec moins d'importance puisqu'elle est composée de personnes intervenant également au niveau du centre opérationnel ou du sommet stratégique. En effet, on retrouve là un point commun avec les centres de recherche où de nombreuses personnes participent à l'activité de recherche et à qui on laisse une bonne autonomie qui découle de la standardisation des résultats qui permet ainsi à chacun de faire valoir ses idées, ce qui peut être très bénéfique en terme de recherche. Le sommet stratégique supervise tout de même en évoquant les grandes lignes d'action. Les fonctions de support logistique sont également très développées. En effet, elles assurent au centre opérationnel de bonnes conditions de travail. La technostructure apparait quand à elle moins développée. Comme évoqué précédemment la majorité des personnes contribuent aux activités de recherche. L'idéologie est également belle et bien présente. En effet, l'école possède au niveau de la recherche des domaines de compétences dans lesquels elle s'efforce de progresser et qui font sa renommée depuis de nombreuses années. L'ENSTA Bretagne est en effet reconnue dans le domaine de l'architecture navale et offshore ainsi que pour l'hydrographie et pour le milieu maritime en général, que l'on retrouve dans tous les laboratoires de recherche. De plus, l'ENSTA Bretagne a un statut particulier en étant sous tutelle de la Direction Générale de l'Armement (DGA) du ministère de la Défense.

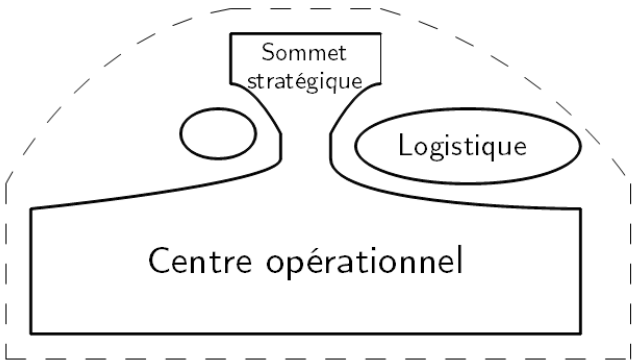

FIGURE 4.1 – Modèle de Mintzberg

Ainsi, l'organisation peut être rapportée à celle d'un centre de recherche où les chercheurs peuvent faire preuve d'une certaines autonomie pour répondre aux demandes du sommet hiérarchique. De plus, le fait que l'ENSTA Bretagne soit reconnue pour ses compétences dans le domaine maritime apporte une culture du milieu qui donne encore plus de sens aux travaux réalisés.

Conclusion

Ainsi, ce stage m'a permis de mettre en pratique des connaissances techniques dans des domaines variés. J'ai en effet pu travailler sur la réalisation d'un système complet. Ce fut de plus l'occasion d'avoir un regard critique sur l'organisation d'un laboratoire de recherche.

L'objectif du stage était de réaliser un robot planeur capable de puiser l'énergie nécessaire au vol dans son environnement en profitant des courants ascendants. Dans la partie théorique une modélisation du planeur sous forme d'équations d'état a pu être faite ainsi que le choix et la mise en place des différents régulateurs. La partie simulation consistait en la modélisation en 3D du planeur puis à son intégration dans un simulateur muni d'une interface graphique. Cependant, la représentation d'état n'a pu être testée car le simulateur n'a pas pu être entièrement terminé. La partie régulation a quant à elle pu être validée par des tests effectués sur des robots roulants. Un prototype de planeur autonome a par la suite été construit en se basant sur un plan de modèle réduit remplissant des exigences particulières pour sa fabrication et sa robustesse. La partie électronique est constituée de plusieurs composants et capteurs qui ont été mis en relation pour assurer la régulation du planeur. Enfin, la partie expérimentale du stage, la plus conséquente a permis de valider les parties précédemment effectuées en particulier la partie électronique associée à la régulation et la partie mécanique, et d'évaluer les capacités du robot et de définir ses possibilités d'utilisation.

Enfin, ce stage m'a permis de m'intégrer au sein d'une équipe travaillant dans le domaine de la robotique. J'ai en effet pu participer à l'étude d'autres systèmes comme le robot voilier. Cela démontre l'intérêt de l'échange de connaissances, pouvant découlées de l'expérience acquise lors de travaux sur un projet particulier, pour les appliquées à son travail et faire inversement profiter de son expérience.

A Représentation d'état

Les figures A.1 et A.2 représentent le planeur à modéliser.

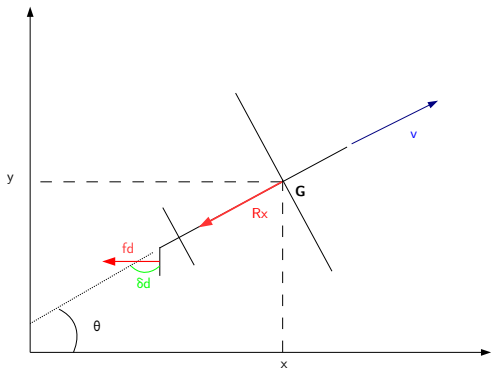

FIGURE A.1 – Planeur que l'on souhaite modéliser (1)

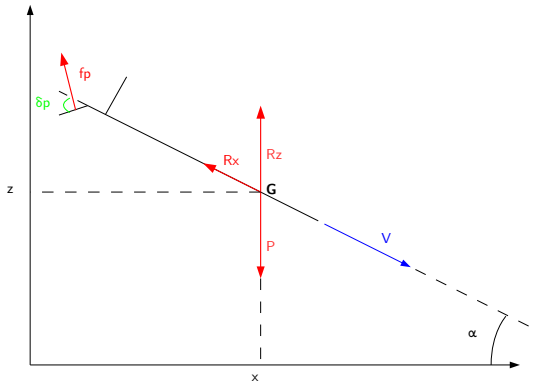

FIGURE A.2 – Planeur que l'on souhaite modéliser (2)

Le vecteur d'état est composé des coordonnées du planeur : x, y et z, lié à son centre de gravité G. Son orientation θ et sa pente α ainsi que les angles que forment les gouvernes avec le fuselage : δ_d pour la gouverne de direction et δ_p pour la gouverne de profondeur, apparaissent également dans le vecteur d'état. Celui-ci comprend également la vitesse V du planeur.

Les entrées du systèmes sont notées u_1 et u_2, elles correspondent aux dérivés respectives des angles δ_d et δ_p. Les paramètres supplémentaires qui ont été pris en compte sont :

- α_d la portance de la gouverne de direction
- α_p la portance de la gouverne de profondeur
- C_z le coefficient de portance des ailes ici égal à 1 correspondant à celui d'un profil *Clark Y* auquel le profil de l'aile de ce planeur s'apparente
- C_x le coefficient de trainée des ailes ici égal à 0,021 pour obtenir une finesse de 40 en accord avec le profil utilisé
- $\rho = 1,293 kg/m^3$ la densité de l'air
- S la surface totale de l'aile

Pour obtenir un système d'équations d'état du type :

$$\dot{x} = f(x, u)$$

on applique le principe fondamental de la dynamique au planeur.

Les forces s'exerçant sur le planeur sont :

- $R_z = \frac{1}{2}\rho S V^2 C_z$ la portance
- $R_x = \frac{1}{2}\rho S V^2 C_x$ la trainée
- $P = mg$ le poids de l'avion
- $f_d = \alpha_d V \sin\delta_d$ la force exercée par l'air sur la gouverne de direction
- $f_p = \alpha_p V \sin\delta_p$ la force exercée par l'air sur la gouverne de profondeur

Le principe fondamental de la dynamique appliqué au planeur s'écrit dans les deux cas schématisés précédemment et donne :

$$\begin{cases} \dot{V} &= \dfrac{R_z}{m} - g\cos\alpha - f_p \sin\delta_p \\ \dot{v} &= g\sin\alpha - \dfrac{R_x}{m} - f_d \sin\delta_d \\ \dot{\alpha} &= \dfrac{R_z}{mV}\cot\alpha - \dfrac{g\cos^2\alpha}{V\sin\alpha} - f_p\dfrac{\sin\delta_p}{V}\cot\alpha - \dfrac{g}{V} + \dfrac{R_x}{mV\sin\alpha} + \dfrac{f_d\sin\delta_d}{V\sin\alpha} \end{cases}$$

On obtient alors les équations d'état suivantes :

$$\begin{cases} \dot{x} &= V\cos\alpha\cos\theta \\ \dot{y} &= V\cos\alpha\sin\theta \\ \dot{z} &= V\sin\alpha \\ \dot{\theta} &= Vu_1 \\ \dot{\delta_d} &= u_1 \\ \dot{\delta_p} &= u_2 \\ \dot{V} &= \dfrac{R_z}{m} - g\cos\alpha - f_p\sin\delta_p \\ \dot{\alpha} &= \dfrac{R_z}{mV}\cot\alpha - \dfrac{g\cos^2\alpha}{V\sin\alpha} - f_p\dfrac{\sin\delta_p}{V}\cot\alpha - \dfrac{g}{V} + \dfrac{R_x}{mV\sin\alpha} + \dfrac{f_d\sin\delta_d}{V\sin\alpha} \end{cases}$$

B Les algorithmes de régulation

B.1 La régulation suivant un cap

La régulation suivant un cap était une première étape avant de réaliser une fonction permettant au planeur de rejoindre un point GPS puisqu'un calcul de cap est effectué pour atteindre ce point à partir de sa position initiale.

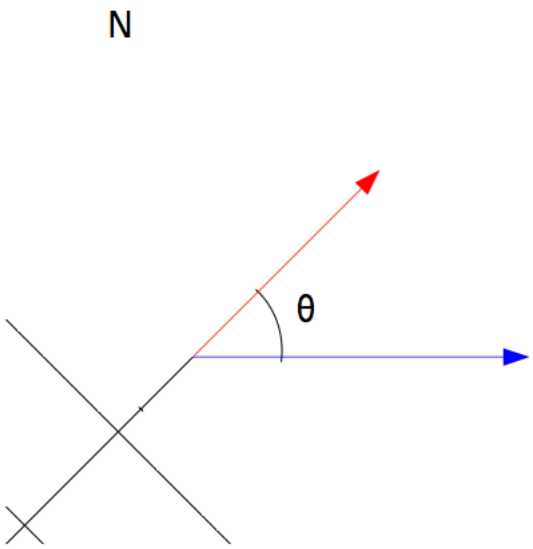

FIGURE B.1 – Différence entre les caps

La méthode retenue pour le régulateur est une commande proportionnelle suivant la valeur du déterminant des deux vecteurs représentants les deux caps, celui actuellement suivi et le cap à suivre (Figure B.1).

Les différents cas traités sont répertoriés sur la machine d'état de la figure B.2 page 27.

La fonction utilisée est *regulationCap(float capVoulu)*. Le produit scalaire et le déterminant entre les deux vecteurs ont tout d'abord été calculés. Cependant, étant donné que les calculs s'effectuent dans le repère indirect (0, Nord, Est), le déterminant se voit alors affecté d'un signe moins. Puis les différents cas sont alors testés par d'abord une condition sur le produit scalaire. S'il est positif, l'angle θ est aigu et la commande se fait proportionnellement au déterminant, en effet celui-ci diminue lorsque l'angle diminue et implique donc une commande appliquée au servomoteur plus faible pour les faibles valeurs de θ. Le déterminant variant entre -1 et 1, le coefficient de proportionnalité appelé ici K représente donc l'angle maximum qui peut être envoyé au servomoteur, sa valeur a été choisie à 20°. Cette valeur apparait comme étant la valeur maximale évitant les mouvements trop brusques qui auraient risqué de déséquilibrer l'avion ou même de le ralentir fortement alors que la vitesse est un facteur important dans le vol plané. Lorsque le produit scalaire

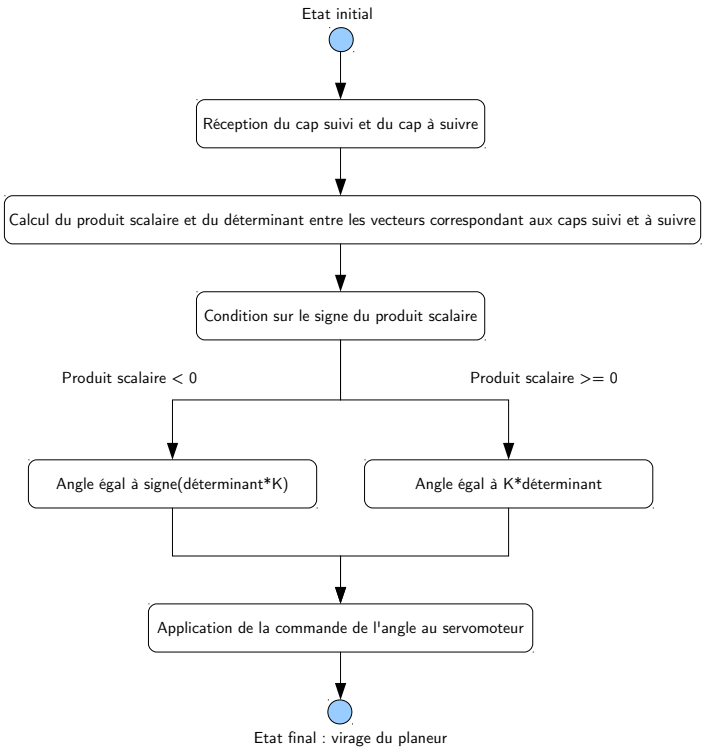

FIGURE B.2 – Automate de régulation de cap

est négatif, l'angle θ est alors supérieur à 90°, le cap à rejoindre est donc dans le secteur arrière du planeur. Le virage doit donc commencer par se faire à l'angle maximum du servomoteur. Il n'y a donc pas dans ce cas de commande proportionnelle, la commande appliquée au servomoteur sera donc K ou -K suivant le sens dans lequel doit être effectué le virage.

Cette régulation sera utilisée pour suivre un cap, notamment pour le suivi de route et pour la régulation en cercle.

B.2 Rejoindre un point de coordonnées GPS

Tout d'abord, il fallait créer une fonction permettant de calculer le cap pour rejoindre un point GPS à partir de sa position actuelle, et également la distance entre ces deux points. Ces fonctions sont répertoriées dans le fichier *Waypoint.pde*. La première calcule le cap à prendre pour atteindre le deuxième point entré en paramètre à partir du premier

point passé aussi en paramètre. Elle effectue le calcul suivant :

$$Cap = atan2(\sin(long1 - long2)\cos(lat2), \cos(lat1)\sin(lat2) - \sin(lat1)\cos(lat2)\cos(long2 - long1))$$

Le calcul de distance se fait à partir de la formule :

$$distance = 2\arcsin(\sqrt{\sin^2(\frac{lat1 - lat2}{2}) + \cos(lat1)\cos(lat2)\sin^2(\frac{lon1 - lon2}{2})})$$

Des formules utilisées cette dernière est présentée comme étant celle qui est la moins sensible aux erreurs d'arrondi. Ces formules sont valables quelque soit la distance séparant les points en raison des termes faisant apparaitre les différences de latitude et longitude.

B.3 Régulation en écart par rapport à la route

La régulation en cap présentée précédemment ne suffit pas à suivre une route définie entre deux points. En effet, le planeur dérive et s'éloigne de sa route. Cela peut être dû au vent, et à l'imprécision du GPS et de son temps d'actualisation de ses données. Pour remédier à ce problème, il a fallu ajouter à la régulation en cap, une régulation en distance à la route. Le principe de cette dernière est d'appliquer une commande supplémentaire, au servomoteur de direction, qui serait fonction de l'écart du planeur à sa route. Pour calculer cet écart, on utilise les angles correspondant au cap à suivre initial et au cap qui permettrait au planeur de rejoindre son point d'arrivée en ligne droite à partir de sa position actuelle. On utilise également la distance qui sépare le planeur de son point d'arrivée, comme cela est représenté sur la figure B.3.

L'expression de la distance à la route s'exprime donc :

$$d = D\sin(|capDirect - capASuivre|)$$

La fonction réalisant ces calculs est la fonction : *regulation(float latitude, float longitude, float latitudeWaypoint, float longitudeWaypoint, float distance, float capasuivre)* du fichier *Planeur.pde*. Elle prend en paramètres la position actuelle du planeur, les coordonnées du waypoint à atteindre, la distance à ce point ainsi que le cap à suivre initial tous deux calculés à l'aide des fonctions du fichier *Waypoint.pde* présentées précédemment. Les valeurs des caps qui seront utilisés pour les calculs sont tout d'abord ramenés entre -180° et 180° :

```
1  // On ramene les valeurs de cap entre -180 et + 180 degres
2  capDirect = conversionCap(capDirect);
3  capASuivre = conversionCap(capasuivre);
```

Puis, les calculs de la distance à la route peuvent s'effectuer en utilisant la formule trouvée.

```
1  // Calcul de la distance a la route prevue
2  differenceCap = abs(capDirect - capASuivre);
3  differenceCap = radians(differenceCap);
4  distanceRoute = distance*sin(differenceCap);
```

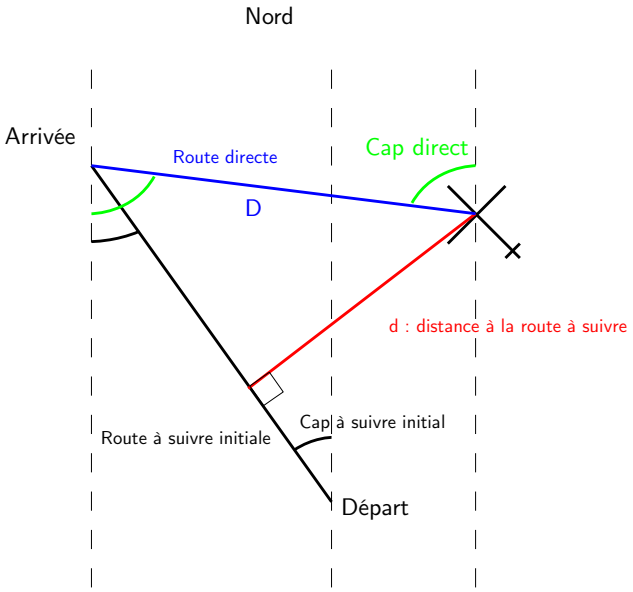

FIGURE B.3 – Schéma du calcul de la distance à la route

Ce calcul ne tient pas compte de la position du planeur par rapport à la route. Le côté duquel se situe le robot va en effet influer sur la commande à appliquer puisque le sens du virage sera différent. De plus, suite aux observations réalisées après les premiers tests de ce régulateur, il est apparu indispensable de ne pas appliquer cette régulation en écart lorsque le robot est suffisamment près de sa route. En effet, cette régulation engendre des oscillations du robot autour de la route et ne progresse plus, lorsqu'il arrive près de celle-ci.

```
1   /* Si le planeur est a moins de 3 metres de sa
2   route il ne regule alors plus qu'en cap */
3   if (abs(distanceRoute) < 3)
4   {
5     angle = regulationCap(capasuivre);
6   }
7
8   else
9   {
10    angle = regulationCap(capasuivre); //application de la regulation en cap
11
12    angle = angle + K1*atan(distanceRoute); /* ajout de la regulation
13
14    en distance a la route*/
15
16  }
```

La commande appliquée est donc la somme de la commande de la régulation en cap et d'une fonction de l'écart à la route. Le choix s'est porté sur la fonction arctanpuisque dans le cadre de notre utilisation elle fait apparaitre une saturation de ses valeurs au-delà d'un certain écart à la route (ici 5 mètres). Le coefficient K_1 permet d'adapter la commande en une valeur angulaire pouvant correspondre à la position du servomoteur. Il est possible de modifier sa valeur pour changer la réactivité du robot. Cependant, il n'apparaissait pas possible de trouver une valeur supprimant les oscillations évoquées et garantissant une bonne réactivité du robot à grande distance de la ligne, c'est pourquoi seule la régulation en cap est active lorsque le robot se situe à moins de 3 mètres de part et d'autre de la route.

La fonction *regulation()* présentée ici offre donc un régulateur permettant un suivi de route.

B.4 La régulation en cercle

Le principe du régulateur permettant au planeur de tourner autour d'un point suivant un cercle repose sur le calcul d'un cap qu'il devra suivre pour rejoindre ce cercle à partir de l'expression d'un champ de vecteur. L'expression du champ choisi est la suivante :

$$h : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \frac{1}{\sqrt{x^2+y^2}} \begin{pmatrix} x \arctan(\rho(r_0 - \sqrt{x^2+y^2})) + y \\ y \arctan(\rho(r_0 - \sqrt{x^2+y^2})) - x \end{pmatrix}$$

Il est représenté sur la figure B.4.

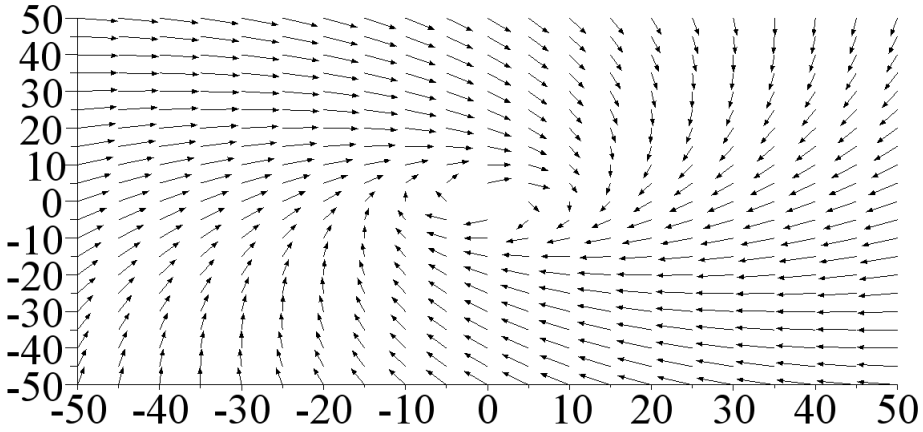

FIGURE B.4 – Le champ de vecteur choisi pour $r_0 = 10$

La fonction *regulationCercle(float latitude, float longitude, float latitudeWaypoint, float longitudeWaypoint, float distance)* réalise cette régulation. Elle prend en paramètres la position du planeur en coordonnées géographiques, les coordonnées du point autour duquel il doit cercler, et la distance qui le sépare de ce point. Les premiers calculs reviennent à déterminer la distance du planeur au cercle et des coordonnées x et y du planeur dans le repère centré sur le waypoint qui serviront pour le calcul du vecteur du champ.

```
1  // Calcul de la distance au cercle
2  differenceDistance = r0 - distance;
3
4  // Calcul de la position du planeur par rapport au waypoint
5  x = k0*(radians(longitude) - radians(longitudeWaypoint));
6  y = k0*(radians(latitude) - radians(latitudeWaypoint));
```

r_0 correspond au rayon du cercle, ici on a choisi 10 mètres. Le coefficient $k_0 = 361.54818 * 1000.0 * RAYON$ permet de ramener les différences des angles de latitude et de longitude à l'échelle du repère, en mètres. La valeur $RAYON$ correspond au rayon moyen de la Terre en kilomètres (6371 km) et le terme numérique a été déterminer expérimentalement en comparant des valeurs de distance obtenues par calcul à partir des coordonnées géographiques grâce à la formule de calcul de distance énoncé précédemment. Puis on calcul les composantes du vecteur du champ correspondant à la position du robot.

```
1  // Calcul des deux composantes du vecteur definissant le cap
2  cap1 = x*atan(k2*differenceDistance) + y;
3  cap2 = y*atan(k2*differenceDistance) - x;
```

Le coefficient k_2 indique la vitesse à laquelle le robot se rapproche du cercle. Ensuite, à partir du vecteur obtenu, on en déduit un cap, c'est celui que le planeur devra suivre pour intercepter le cercle.

```
1  // Calcul du cap a suivre pour rejoindre le cercle
2  capASuivre = atan2(cap1, cap2);
3  capASuivre = degrees(capASuivre);
```

Ce cap est ensuite ramené de 0° à 360° pour être utilisé par la fonction de régulation en cap :

```
1  angle = regulationCap(capASuivre); //application de la regulation en cap
```

Ainsi, le robot calcul à partir de sa position par rapport au cercle un cap qui lui permet de l'atteindre et de le suivre.

C L'électronique

La partie électronique permet la réalisation de la régulation du planeur, le rendant autonome. Elle comprend plusieurs parties. Tout d'abord, nous verrons la plate-forme de développement utilisée, *Arduino Uno* qui sert d'interface entre les actionneurs et les capteurs et où est également chargé le code permettant le contrôle du planeur. Puis nous verrons le GPS utilisé qui est connecté directement sur la carte *Arduino* et qui fournit les coordonnées géographiques de l'avion ainsi que son cap. Ensuite, nous verrons les caractéristiques des servomoteur utilisés ainsi que leur fonctionnement via la carte.

C.1 La plate-forme de développement *Arduino*

Les cartes *Arduino* sont des cartes de développement open-source, très fonctionnelles grâce aux multiples librairies que comporte le langage de programmation utilisé. Ce dernier s'apparente d'ailleurs comme annoncé ci-dessus à du langage C. Mon choix s'est porté sur l'*Arduino Uno* qui de petite taille et possède toutes les caractéristiques requises pour contrôler les servomoteurs et capteurs nécessaires à la commande du planeur comme en particulier le GPS.

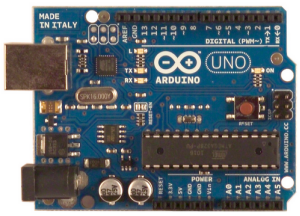

FIGURE C.1 – La carte *Arduino Uno*

La carte *Arduino Uno* se compose d'un micro-contrôleur *ATmega 328* équipé d'une mémoire flash de 32KB pour charger le programme, d'une mémoire EEPROM de 1KB accessible grâce à une librairie adaptée. Elle peut être alimentée par son port USB servant également d'interface avec l'ordinateur notamment pour charger le programme, ou par une alimentation externe. Dans le cas de connexion de plusieurs sources d'alimentation, elle choisit automatiquement la plus adaptée. Ici l'alimentation se fera par une batterie Lithium-Polymère de 7,4 V et 1800 mA.h, qui correspond à la plage nominale d'alimentation qui est va de 7 V à 12 V, la limite supérieure étant de 20 V. L'*Arduino Uno* peut fournir 6 PWM [1] et 8 autres entrées/sorties numériques. Ce sont sur ces broches que sont connectés les servomoteurs, et la disponibilité de plusieurs autres offre des perspectives d'ajouts d'actionneurs. Un port série est également disponible mais n'est utilisé ici que pour l'affichage dans la console de l'IDE par le biais du port série du micro-contrôleur.

1. *Pulse Width Modulation* : modulation de largeur d'impulsion permettant entre autres de contrôler des moteurs à courant continu

De plus, la carte dispose de 6 entrées analogiques. Pour la communication I2C [2] on utilise deux de celles-ci pour les fils SDA [3] et SCL [4].

Cette carte est parfaitement adaptée au prototype grâce à l'adaptation et la compatibilité de différents capteurs et actionneurs comme le GPS et les servomoteur utilisés. Elle permet de plus d'envisager une évolution du prototype en ajoutant d'autres capteurs comme des accéléromètres et des gyromètres, en effet de nombreuses autres cartes de capteurs sont disponibles et adaptées à l'*Arduino Uno*.

C.2 Le GPS

Le GPS a été choisi comme le seul composant fournissant des données. Il donne avec précision les coordonnées GPS du planeur et son cap.

Le GPS utilisé est le *DS-GPM.S*. Il a été conçu pour l'*Arduino Uno*, il s'adapte donc parfaitement à la carte. Il a cependant fallu souder des *headers* le positionnant ainsi sur le dessus de l'*Arduino*. Une fois montée sur la carte, les correspondances des connexions se font de la manière suivante, le signal SDA est relié à l'entrée analogique 4 de l'*Arduino* et le signal SCL à l'entrée analogique 5. La communication entre le GPS et la carte se fait par le biais d'un bus I2C. Lors de la communication, la carte *Arduino* est toujours considérée comme le périphérique maître, et génère donc l'horloge (le signal SCL) et fait les demandes d'envoi de données de la part du GPS qui est alors le périphérique esclave. Il n'y a pas d'autres périphériques connectés sur le bus, cependant il est intéressant de disposer d'un bus I2C, cela permettrait un ajout de plusieurs périphériques d'autant que cette solution laisse le port série libre. L'utilisation du GPS se fait principalement en mode lecture bien qu'il soit possible d'écrire dans certains de ses registres, pour lire les données stockées dans ses registres. Pour cela, une librairie *Wire* est disponible pour interagir en I2C avec les périphériques. Tout d'abord, il faut déclarer la carte comme étant le périphérique maître, lors de son initialisation qui se fait par l'exécution d'une fonction *setup()*, en appliquant l'instruction *Wire.begin()* qui initialise par la même occasion la communication en I2C. Pour effectuer la lecture des registres du GPS, l'*Arduino* doit initier la communication en envoyant à destination de l'adresse du GPS, l'adresse du registre qu'elle souhaite lire. Cela se fait de la façon suivante :

```
1  // Initialisation de la communication
2  Wire.beginTransmission(GPSADDRESS);
3  Wire.send(registre);
4  Wire.endTransmission();
```

La réception des données par paquets d'un octet peut être effectuée sur la demande du maître qui spécifie le nombre d'octets qu'il souhaite recevoir. Un registre contenant un octet si la demande excède un octet, les octets suivants envoyés correspondent à ceux stockés dans les registres suivants consécutivement. Cela est utile comme nous le verrons par la suite, car une information peut être répartie sur plusieurs registres. L'acquisition

2. *Inter Integrated Circuit*, bus série et synchrone
3. Le signal de données
4. Le signal d'horloge

d'un octet se fait de la façon suivante :

```
1  // Demande d'acquisition d'un octet
2  Wire.requestFrom(GPSADDRESS, 1);
3  while(Wire.available() < 1);  // attente d'un byte disponible
4  val = Wire.receive();  // reception et stockage dans la variable val
```

Une fonction *acquire()* a été créée regroupant les instructions précédentes qui permettent la réception d'un octet donc la lecture d'un registre, l'adresse de ce dernier étant déclarée en variable globale du programme permettant de la modifier simplement. C'est cette dernière qui est utilisée dans les fonctions mentionnées ci-dessous. Un diagramme de séquence présenté en figure C.2 résume le protocole de communication entre le GPS et l'*Arduino*.

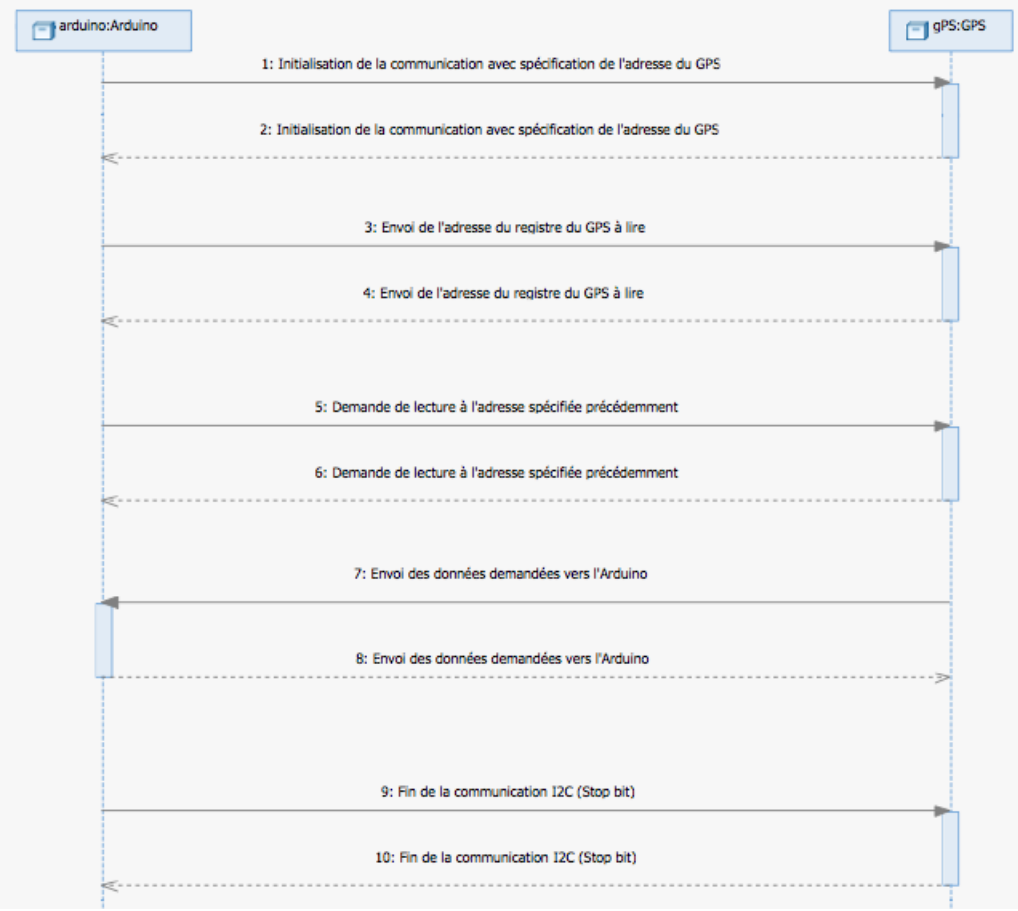

FIGURE C.2 – Diagramme de séquence de la communication I2C

Le GPS peut fournir de nombreuses informations sur la position du planeur. En effet, il est possible de connaître, sa position géographique (latitude et longitude) avec une précision allant de 1 à 2,5 mètres, son altitude, le cap suivi (selon le nord vrai ou le nord magnétique) et sa vitesse. Des informations telles que la date et la position des satellites sont également disponibles. Il est également possible de connaître le mode de fonctionnement actuel du GPS, il peut être en fonctionnement normal autonome, en mode

d'estimation en cas de perte de signal ou il peut indiquer que les données envoyées ne sont pas valides. Toutes ces données sont stockées dans 112 registres qui sont mis à jour toutes les secondes. Les données complètes d'une information sont réparties sur plusieurs registres consécutifs. Pour que les données reçues soient donc directement exploitables par le programme, il a donc fallu créer des fonctions faisant des acquisitions des données des registres successifs en stockant dans des variables intermédiaires les valeurs obtenues entre chaque lecture de registre, en les mettant au fur et à mesure au bon format pour retourner une valeur finale dans l'unité attendue. C'est le cas par exemple des fonctions : *getLatitude()*, *getLongitude()*, *getHeading()* et *getAltitude()*. Ces fonctions sont regroupées dans le fichier *GPS.pde*. D'autres fonctions permettant l'affichage des informations dans la console ont également été ajoutées. Elles ont principalement servi au *debug*, pour des tests unitaires permettant de valider les fonctions d'acquisition ainsi qu'à créer des fichiers log.

Comme énoncé précédemment, le GPS sert essentiellement à connaître la position du planeur et son cap. Ces informations sont directement utilisées par les fonctions de régulation après leur acquisition par les fonctions présentées ci-dessus. Il nécessite un endroit bien dégagé pour qu'il puisse entrer en contact avec les satellites. A l'extérieur un démarrage « à froid » peut nécessiter une dizaine de minutes (30 minutes maximum si sa dernière position connue est très loin de sa position de réinitialisation) mais quelques minutes voire quelques secondes suffisent si sa dernière utilisation est récente.

C.3 Le servomoteur

Le servomoteur choisi pour la gouverne de direction mais également pour la gouverne de profondeur est le *HITEC HS-322HD*, il est commandé en position. Il offre un couple de 3,0 kg.cm, et une vitesse de 0,19 s/60° et pèse 43 g. Ils sont connectés sur la carte électronique, *Arduino Uno*. Ils sont connectés sur les pins d'alimentation 5V et GND, et le signal PWM, la broche de sortie PWM numéro 9 correspond au servomoteur de direction et la numéro 10 à celui de la profondeur. Ils sont commandés à l'aide de la librairie *Servo* fournissant les méthodes permettant l'utilisation directe de servomoteurs. La méthode *attach(numéro de pin de PWM)* permet d'assigner le pin de PWM de la carte au servomoteur. La méthode permettant de lui imposer une valeur de PWM est *write(valeur en degrés)*, elle retranscrit la valeur passée en degrés en valeur correspondante de PWM. Il est également possible de retrouver la valeur précédemment affectée au servomoteur par la méthode : *read()*.

Pour commander le servomoteur, on lui envoie une valeur de position angulaire variant entre 0 et 180°. Ces deux valeurs correspondent aux valeurs théoriques maximales pouvant être atteintes par le servomoteur, la position neutre du servomoteur doit donc se situer à 90°. Cependant, en pratique, ces valeurs ne sont pas exactement les mêmes, il a donc fallu étalonner le servomoteur. Pour cela, j'ai déterminé à l'aide de tests unitaires, les positions du neutre et les positions des butées. Les valeurs alors trouvées sont pour le neutre : 88°, pour le maximum 172° et 10° pour le minimum. Son débattement maximal théorique de 90° de part et d'autre de la position neutre n'est donc pas obtenue donc le cas pratique, cependant cela n'a pas de conséquences pour la suite étant donné qu'un débattement aussi important n'était pas nécessaire. Il a d'ailleurs été comme nous le verrons par la suite restreint à 20° qui paraissait être une limite pour éviter les évolutions trop brusques.

C'est également du servomoteur de direction que découlent les fonctions permettant de guider le planeur. En l'occurrence ici, la fonction *tourner(float angle)* du fichier *Direction.pde* est utilisée pour faire tourner le planeur. Elle prend en paramètre la valeur de la position du servomoteur par rapport au servomoteur, les valeurs positives permettent un virage à gauche et les valeurs négatives, un virage à droite. Elle ajoute l'angle demandé à la position neutre et compare la valeur ainsi obtenue avec les positions maximales déterminées pour éviter de mettre le servomoteur en butée. Puis elle envoie la commande obtenue au servomoteur. Une fonction fonctionnant de façon analogue à la précédente permet d'appliquer une commande au servomoteur agissant sur la gouverne de profondeur.

C.4 La centrale inertielle AHRS

L'utilisation d'une centrale inertielle s'avérait intéressante puisqu'elle permettrait de fournir des informations sur la position du planeur à tout instant contrairement au GPS qui ne donne de nouvelles données que toutes les secondes. En effet, une boussole qui n'est pas compensée apparait inutilisable en virage où les valeurs indiquées seront fausses, elle devrait rester bien à plat. Une centrale offre l'avantage d'intégrer des gyroscopes qui permettent les corrections nécessaires pour conserver des informations de cap correctes. La centrale inertielle prévue pour le planeur est la *9DOF Razor IMU*. Elle est composée de trois capteurs, un gyromètre 3 axes *ITG-3200*, un accéléromètre 3 axes *ADXL345* et un magnétomètre 3 axes *HMC5883L*. Elle est également pourvue d'un micro-contrôleur *ATmega328* dans lequel il est possible de charger un programme qui permet de fournir les angles d'Euler du robot après calculs à partir des données brutes des capteurs. Ce programme disponible ne permet cependant que d'afficher les valeurs des valeurs des angles. Or dans notre cas, il paraissait intéressant de pouvoir également utiliser ces données. Il a donc fallu mettre en place un protocole de communication entre la carte *Arduino Uno* et la centrale inertielle. La carte et la centrale sont par connectées sur le port série. La librairie *Wire* permet de gérer la communication sur le port série. Pour recevoir les informations de la centrale, on utilise la fonction *getAttitude(float *roll, float *pitch, float *yaw)* du fichier *AHRS.pde* qui écrit sur le port série un caractère indiquant à la centrale qui reçoit ce caractère qu'elle doit écrire sur le port série les données (les angles d'Euler) sous un format particulier : une chaîne de caractère qui sera traitée à sa réception pour en extraire les données voulues. Voici le code de la fonction du programme contenu dans la carte *Arduino*.

```
1  /* Fonction permettant de recevoir les angles d'Euler */
2  void getAttitude(float *roll, float *pitch, float *yaw)
3  {
4    char commandbuffer[128];
5    /* caractere envoye par l'Arduino sur le port serie pour
6    demander a la centrale d'envoyer des donnees*/
7    char start = '0';
8
9    // demande de reception de donnees
10   Serial.write(start);
11
12   int i = 0; //compteur du nombre de byte dans le buffer du port serie
13
14   // recuperation des donnes des qu'elles sont disponibles
```

```
15    if (Serial.available())
16    {
17        delay(100);
18        while ( Serial.available() && i < 127)
19        {
20            commandbuffer[i++] = Serial.read();
21        }
22        commandbuffer[i++]='\0';
23    }
24
25    /* variables intermediaires pour l'extraction des donnees
26    a partir de la chaine de caracteres recue*/
27    int r, p, y;
28
29    // traitement des donnees presentes dans la chaine de caracteres
30    sscanf(commandbuffer,"%i %i %i",&r,&p,&y);
31
32    // stockage des donnees extraites et mises au bon format
33    *roll = (float) r/100.0;
34    *pitch = (float) p/100.0;
35    *yaw = (float) y/100.0;
36
37 }
```

Tout d'abord, on prépare un tableau dans lequel seront stockés les données reçues. Puis on écrit un caractère sur le port série qui signale à la centrale qu'elle doit envoyer la chaîne de caractères contenant les informations voulues. Ensuite, les données reçues sont stockées dès qu'elles sont disponibles. Enfin, la chaîne de caractères reçue est traitée, on en extrait les valeurs des angles grâce à la fonction *sscanf* qui sont alors stockées dans les variables correspondantes (*roll*, *pitch* et *yaw*).

Voici, les lignes qui ont été ajoutées au programme initialement contenu dans la centrale inertielle.

```
1 if(Serial.available()){
2       delay(100);
3       while(  i < Serial.available()) {
4           Serial.read();
5           i++;
6       }
7       printdata();
8   }
```

Cela permet d'envoyer les données uniquement lors de la réception d'un caractère provenant de la carte *Arduino*. Les données sont envoyées sous la forme d'une chaîne de caractères comprenant d'abord la valeur de l'angle de roulis (roll), puis celle de l'angle de tangage (pitch) et enfin l'angle de lacet (yaw) qui n'est autre que le cap. Ces valeurs sont séparées par des espaces. Il est ainsi possible d'accéder à partir du programme où s'effectuent tous les calculs concernant la régulation du robot aux valeurs des angles d'Euler calculées par la centrale.

Cependant, la centrale est apparue très difficile d'utilisation puisqu'elle nécessite un calibrage au démarrage. En effet, elle possède une période de calibration à son initialisation durant laquelle elle doit être maintenue parfaitement à plat ce qui est très difficile à mettre en œuvre compte tenu de la sensibilité des capteurs.

D Le code

Le langage utilisé s'apparentant au langage C, une programmation modulaire a été adoptée avec un fichier *header* correspondant à chaque fichier de fonctions.

D.1 Planeur.pde

```
1  /* Planeur : fichier principal d'execution du code */
2
3  #include <Wire.h>
4  #include <Servo.h>
5  #include "Defines.h"
6  #include "GPS.h"
7  #include "Direction.h"
8  #include "Waypoint.h"
9  #include "AHRS.h"
10
11 Servo servoDirection; // correspond au servomoteur de direction du planeur
12 Servo servoProfondeur; /* correspond au servomoteur de la gouverne de
13 profondeur du planeur*/
14
15 /* Fonction calculant la position du planeur */
16 void getPosition(float *latitude, float *longitude, float latitudeWaypoint,
17 float longitudeWaypoint, float *distance,
18 float *roll, float *pitch, float *yaw)
19 {
20   *latitude = getLatitude();
21   *longitude = getLongitude();
22   *distance = 1000*getDistance(*latitude, *longitude,
23   latitudeWaypoint, longitudeWaypoint);
24   //getAttitude(roll, pitch, yaw);
25 }
26
27 /* Fonction de regulation en cap du planeur - capVoulu en degres */
28 float regulationCap(float capVoulu)
29 {
30   float commande = 0.0;
31   float capSuivi = 0.0;
32   float det = 0.0;
33   float scal = 0.0;
34   float angle = 0.0;
35
36   commande = radians(capVoulu);
37
38   // demande du cap suivi actuellement suivi et conversion en radians
39   capSuivi = getHeading();
40
41   capSuivi = radians(capSuivi);
42
43   // Calcul du determinant et du produit scalaire
44   det = -(cos(capSuivi)*sin(commande) - cos(commande)*sin(capSuivi));
45   scal = cos(capSuivi)*cos(commande) + sin(capSuivi)*sin(commande);
```

```
46
47    // Test sur le signe du produit scalaire pour determiner l'angle a
48    //appliquer au servomoteur
49    if (scal >= 0) {
50      angle = K*det;
51    }
52    if (scal < 0) {
53      if (det < 0) {
54        angle = -K;
55      }
56      else if (det >= 0) {
57        angle = K;
58      }
59    }
60
61    return angle;
62  }
63
64  /* Fonction de regulation permettant au planeur de rejoindre un point GPS
65  suivant une route */
66  void regulation(float latitude, float longitude, float latitudeWaypoint,
67  float longitudeWaypoint, float distance, float capasuivre)
68  {
69    float distanceRoute = 0.0;
70    float capDirect = 0.0;
71    float capASuivre = 0.0;
72    float angle = 0.0;
73    float differenceCap = 0.0;
74
75    // Calcul du cap direct
76    capDirect = getBearing(latitude, longitude, latitudeWaypoint,
77    longitudeWaypoint);
78
79    // On ramene les valeurs de cap entre -180 et + 180 degres
80    capDirect = conversionCap(capDirect);
81    capASuivre = conversionCap(capasuivre);
82
83    // Calcul de la distance a la route prevue
84    differenceCap = abs(capDirect - capASuivre);
85    differenceCap = radians(differenceCap);
86    distanceRoute = distance*sin(differenceCap);
87
88    // Test permmettant de savoir de quel cote de la route se
89    //trouve le planeur
90    if (capDirect > capASuivre)
91    {
92      distanceRoute = -distanceRoute;
93    }
94
95    /* Si le planeur est a moins de 3 metres de sa route
96    il ne regule alors plus qu'en cap */
97    if (abs(distanceRoute) < 3)
98    {
99      angle = regulationCap(capasuivre);
100   }
101
```

```
102   else
103   {
104     angle = regulationCap(capasuivre); //application de la regulation en cap
105
106     angle = angle + K1*atan(distanceRoute); /*ajout de la regulation en
107     distance a la route*/
108
109   }
110
111   tourner(angle); // virage
112
113 }
114
115 /*Fonction permettant au planeur de cercler autour d'un waypoint */
116 void regulationCercle(float latitude, float longitude,
117 float latitudeWaypoint, float longitudeWaypoint, float distance)
118 {
119   int r0 = 10;
120   int v0 = 1;
121   float k0 = 361.54818*1000.0*RAYON;
122   float k2 = 0.1;
123   float cap1 = 0.0;
124   float cap2 = 0.0;
125   float capASuivre = 0.0;
126   float angle = 0.0;
127   float differenceDistance = 0.0;
128   float x = 0.0;
129   float y = 0.0;
130
131   // Calcul de la distance au cercle
132   differenceDistance = r0 - distance;
133
134   // Calcul de la position du planeur par rapport au waypoint
135   x = k0*(radians(longitude) - radians(longitudeWaypoint));
136   y = k0*(radians(latitude) - radians(latitudeWaypoint));
137
138   // Calcul des deux composantes du vecteur definissant le cap
139   cap1 = x*atan(k2*differenceDistance) + v0*y;
140   cap2 = y*atan(k2*differenceDistance) - v0*x;
141
142   // Calcul du cap a suivre pour rejoindre le cercle
143   capASuivre = atan2(cap1, cap2);
144   capASuivre = degrees(capASuivre);
145
146   if (capASuivre < 0)
147   {
148     capASuivre = capASuivre + 360;
149   }
150
151   // Application de la regulation en cap
152   angle = regulationCap(capASuivre); //application de la regulation en cap
153
154   tourner(angle); // virage
155
156 }
157
```

```
158
159 /* Initialisation */
160 void setup()
161
162 {
163   // Initialisation pour la communication par le bus I2C : Arduino maitre
164   Wire.begin();
165
166   Serial.begin(57600);
167   Serial.flush();
168
169   // Assignement des pins correspondants aux servomoteurs
170   //et mise au neutre des servos pendant 1 seconde
171   servoDirection.attach(PINSERVODIRECTION);
172   servoDirection.write(NEUTREDIRECTION);
173   servoProfondeur.attach(PINSERVOPROFONDEUR);
174   servoProfondeur.write(NEUTREPROFONDEUR);
175
176   delay(5000);
177
178   char mode;
179
180   mode = getMode();
181
182   // Les coordonnees initiales du planeur ne sont pas acquises
183   // tant que le GPS n'a pas actualise sa position et ne
184   // fonctionne en mode autonome
185   while (mode == 'N')
186   {
187     mode = getMode();
188     //Serial.println("Le GPS n'est pas encore initialise.");
189   }
190 }
191
192
193 /* Parcours */
194
195 void loop()
196 {
197   // Position du planeur
198   float latitudePlaneur = 0.0;
199   float longitudePlaneur = 0.0;
200   float roll = 0.0; // angle de roulis
201   float pitch = 0.0; // angle de tangage
202   float yaw = 0.0; // angle de lacet
203
204   // Coordonnees du point a atteindre
205   float latitudeWaypoint = 48.418471; //latitude du point GPS a atteindre
206   float longitudeWaypoint = -4.473938; //longitude du point GPS a atteindre
207
208   float distanceAuBut = 0.0; // distance au point a atteindre
209   float capASuivre = 0.0; // cap a suivre pour atteindre le point GPS
210
211   capASuivre = getBearing(48.418637, -4.473026, latitudeWaypoint,
212   longitudeWaypoint);
213
```

```
214     getPosition(&latitudePlaneur, &longitudePlaneur, latitudeWaypoint,
215     longitudeWaypoint, &distanceAuBut, &roll, &pitch, &yaw);
216     distanceAuBut = 1000*getDistance(latitudePlaneur,
217      longitudePlaneur, latitudeWaypoint, longitudeWaypoint);
218
219     while (distanceAuBut > 10)
220     {
221       regulation(latitudePlaneur, longitudePlaneur, latitudeWaypoint,
222       longitudeWaypoint, distanceAuBut, capASuivre);
223       getPosition(&latitudePlaneur, &longitudePlaneur, latitudeWaypoint,
224       longitudeWaypoint, &distanceAuBut, &roll, &pitch, &yaw);
225
226       delay(15);
227     }
228
229     while (1)
230     {
231       getPosition(&latitudePlaneur, &longitudePlaneur,
232       latitudeWaypoint, longitudeWaypoint, &distanceAuBut, &roll, &pitch, &yaw);
233       regulationCercle(latitudePlaneur, longitudePlaneur,
234       latitudeWaypoint, longitudeWaypoint, distanceAuBut);
235
236       delay(15);
237     }
238 }
```

D.2 Defines.h

```
1  #ifndef DEF_DEFINES
2  #define DEF_DEFINES
3
4  #define GPSADDRESS 0x68 // B1101000
5
6  #define PINSERVODIRECTION 9 //numero de broche du servomoteur de direction
7  #define NEUTREDIRECTION 100 //Position neutre du servomoteur
8  #define POSMAXDIRECTION 172 //Position maximale du servomoteur de direction
9  #define POSMINDIRECTION 10 //Position minimale du servomoteur de direction
10
11 #define PINSERVOPROFONDEUR 10 //numero de broche du servomoteur de profondeur
12 #define NEUTREPROFONDEUR 90 //Position neutre du servomoteur
13 #define POSMAXPROFONDEUR 172 //Position maximale du servomoteur
14 #define POSMINPROFONDEUR 10 //Position minimale du servomoteur
15
16 #define K 10 //coefficient du regulateur de cap
17 #define K1 4  //coefficient de la regulation en distance a la route
18 #define RAYON 6371 // Rayon moyen de la Terre
19
20 #endif
```

D.3 AHRS.h

```
#ifndef DEF_AHRS
#define DEF_AHRS

void getAttitude(float *roll, float *pitch, float *yaw);

#endif
```

D.4 AHRS.pde

```
#include "Defines.h"
#include "AHRS.h"

/* Fonction permettant de recevoir les angles d'Euler */
void getAttitude(float *roll, float *pitch, float *yaw)
{
  char commandbuffer[128];

  char start = '0'; // caractere envoye par l'Arduino sur le port
  //serie pour demander a la centrale d'envoyer des donnees

  // demande de reception de donnees
  Serial.write(start);

  int i = 0; // compteur du nombre de byte dans le buffer du port serie

  // recuperation des donnes des qu'elles sont disponibles
  if (Serial.available())
  {
     delay(100);
     while ( Serial.available() && i < 127)
     {
        commandbuffer[i++] = Serial.read();
     }
     commandbuffer[i++]='\0';
  }

  int r, p, y; // variables intermediaires pour l'extraction des
  //donnees a partir de la chaine de caracteres recue

  // traitement des donnees presentes dans la chaine de caracteres
  sscanf(commandbuffer,"%i %i %i",&r,&p,&y);

  // stockage des donnees extraites et mises au bon format
  *roll = (float) r/100.0;
  *pitch = (float) p/100.0;
  *yaw = (float) y/100.0;

}
```

D.5 Direction.h

```
#ifndef DEF_DIRECTION
#define DEF_DIRECTION

void tourner(float angle);

int conversionMicroseconds(float angle);

void tournerMicroseconds(int angle);

#endif
```

D.6 Direction.pde

```
#include "Direction.h"
#include "Defines.h"

/* Fonction permettant au planeur de virer */
void tourner (float angle)
{
  float pos = 0.0;

  pos = NEUTREDIRECTION + angle;

  if (pos > POSMAXDIRECTION)
    pos = POSMAXDIRECTION;

  if (pos < POSMINDIRECTION)
    pos = POSMINDIRECTION;

  // Envoi de la commande au servomoteur
  servoDirection.write(pos);

}
```

D.7 GPS.h

```
#ifndef DEF_GPS
#define DEF_GPS

int acquire(char registre);

char getMode();

void displayTime();

void displayLatitude();

float getLatitude();
```

```
13
14 void displayLongitude();
15
16 float getLongitude();
17
18 void displayHeading();
19
20 float getHeading();
21
22 void displaySpeed();
23
24 float getSpeed();
25
26 void displayAltitude();
27
28 float getAltitude();
29
30 #endif
```

D.8 GPS.pde

```
1  /* GPS DS-GPM.S */
2
3  #include "Defines.h"
4  #include "GPS.h"
5
6  /* Fonction d'acquisition d'un octet */
7  int acquire(byte registre)
8  {
9    int val = 0;
10
11   // Initialisation de la communication
12   Wire.beginTransmission(GPSADDRESS);
13   Wire.send(registre);
14   Wire.endTransmission();
15
16   // Demande d'acquisition d'un octet
17   Wire.requestFrom(GPSADDRESS, 1);
18   while(Wire.available() < 1);  // attente d'un byte disponible
19   val = Wire.receive();  // reception du byte
20
21   return val;
22
23 }
24
25
26 /* Affichage du mode du GPS */
27 char getMode()
28 {
29   /* A = Autonomous Mode, D = Differential Mode,
30    E = Estimated Mode (Dead Reckoning), M = Manual input Mode,
31    S = Simulated Mode, N = Data Not Valid */
32   char mode;
```

```
33    byte registre; // registre du GPS a lire
34
35    registre = 56;
36
37    mode = acquire(registre);
38
39    //Serial.println(mode, BYTE);
40    //Serial.println(" ");
41
42    return mode;
43  }
44
45  /* Affichage de l'heure */
46  void displayTime()
47  {
48    byte hourTen, hourUn, minuteTen, minuteUn, secondTen, secondUn;
49
50    // On initialise la lecture au niveau du registre 0
51    Wire.beginTransmission(GPSADDRESS);
52    Wire.send(0);
53    Wire.endTransmission();
54
55    // Demande de reception de la part du maitre
56    Wire.requestFrom(GPSADDRESS, 6);
57
58    // Lecture des registres
59    hourTen = Wire.receive();
60    hourUn = Wire.receive();
61    minuteTen = Wire.receive();
62    minuteUn = Wire.receive();
63    secondTen = Wire.receive();
64    secondUn = Wire.receive();
65
66    // Affichage de l'heure
67    Serial.print(hourTen, DEC);
68    Serial.print(hourUn, DEC);
69    Serial.print(":");
70    Serial.print(minuteTen, DEC);
71    Serial.print(minuteUn, DEC);
72    Serial.print(":");
73    Serial.print(secondTen, DEC);
74    Serial.print(secondUn, DEC);
75    Serial.println("␣UTC");
76
77  }
78
79  /*Affichage de la lattitude */
80  void displayLatitude()
81  {
82    byte latDegTen, latDegUn, latMinTen, latMinUn, latMinTenths,
83    latMinHundredths, latMinThousandths, latMinTenThousandths, latDir;
84
85    // On initialise la lecture au niveau du registre 14
86    Wire.beginTransmission(GPSADDRESS);
87    Wire.send(14);
88    Wire.endTransmission();
```

```
89
90    // Demande de reception de la part du maitre
91    Wire.requestFrom(GPSADDRESS, 9);
92
93    // Reception
94    latDegTen = Wire.receive();
95    latDegUn = Wire.receive();
96    latMinTen = Wire.receive();
97    latMinUn = Wire.receive();
98    latMinTenths = Wire.receive();
99    latMinHundredths = Wire.receive();
100   latMinThousandths = Wire.receive();
101   latMinTenThousandths = Wire.receive();
102   latDir = Wire.receive();
103
104   // Affichage de la lattitude
105   Serial.print(latDegTen, DEC);
106   Serial.print(latDegUn, DEC);
107   Serial.print("␣degrees␣");
108   Serial.print(latMinTen, DEC);
109   Serial.print(latMinUn, DEC);
110   Serial.print(",");
111   Serial.print(latMinTenths, DEC);
112   Serial.print(latMinHundredths, DEC);
113   Serial.print(latMinThousandths, DEC);
114   Serial.print(latMinTenThousandths, DEC);
115   Serial.print("␣minutes␣");
116   Serial.println(latDir);
117
118 }
119
120 /* Fonction renvoyant la latitude en degres */
121 float getLatitude()
122 {
123   float latitude;
124   float minutes;
125   byte letter;
126   byte registre; // registre du GPS a lire
127   float data = 0.0; // donnees lues en provenance du GPS
128
129   registre = 14;
130
131   data = acquire(registre);
132   latitude = data*10;
133   registre++;
134
135   data = acquire(registre);
136   latitude = latitude + data;
137   registre++;
138
139   data = acquire(registre);
140   minutes = data*10;
141   registre++;
142
143   data = acquire(registre);
144   minutes = minutes + data;
```

```
145     registre++;
146
147     data = acquire(registre);
148     minutes = minutes + data/10;
149     registre++;
150
151     data = acquire(registre);
152     minutes = minutes + data/100;
153     registre++;
154
155     data = acquire(registre);
156     minutes = minutes + data/1000;
157     registre++;
158
159     data = acquire(registre);
160     minutes = minutes + data/10000;
161     latitude = latitude + (minutes/60);
162     registre++;
163
164     data = acquire(registre);
165     letter = data;
166     if (letter == 'S')
167       latitude = -latitude;
168
169     return latitude;
170
171 }
172
173 /*Affichage de la longitude */
174 void displayLongitude()
175 {
176     byte longDegHundred, longDegTen, longDegUn, longMinTen,
177     longMinUn, longMinTenths, longMinHundredths, longMinThousandths,
178     longMinTenThousandths, longDir;
179
180     // On initialise la lecture au niveau du registre 23
181     Wire.beginTransmission(GPSADDRESS);
182     Wire.send(23);
183     Wire.endTransmission();
184
185     // Demande de reception de la part du maitre
186     Wire.requestFrom(GPSADDRESS, 10);
187
188     // Reception
189     longDegHundred = Wire.receive();
190     longDegTen = Wire.receive();
191     longDegUn = Wire.receive();
192     longMinTen = Wire.receive();
193     longMinUn = Wire.receive();
194     longMinTenths = Wire.receive();
195     longMinHundredths = Wire.receive();
196     longMinThousandths = Wire.receive();
197     longMinTenThousandths = Wire.receive();
198     longDir = Wire.receive();
199
200     // Affichage de la lattitude
```

```
201    Serial.print(longDegHundred, DEC);
202    Serial.print(longDegTen, DEC);
203    Serial.print(longDegUn, DEC);
204    Serial.print("␣degrees␣");
205    Serial.print(longMinTen, DEC);
206    Serial.print(longMinUn, DEC);
207    Serial.print(",");
208    Serial.print(longMinTenths, DEC);
209    Serial.print(longMinHundredths, DEC);
210    Serial.print(longMinThousandths, DEC);
211    Serial.print(longMinTenThousandths, DEC);
212    Serial.print("␣minutes␣");
213    Serial.println(longDir);
214
215 }
216
217 /* Fonction renvoyant la longitude en degres */
218 float getLongitude()
219 {
220    float longitude;
221    float minutes;
222    byte letter;
223    byte registre; // registre du GPS a lire
224    float data = 0.0; // donnees lues en provenance du GPS
225
226    registre = 23;
227
228    data = acquire(registre);
229    longitude = data*100;
230    registre++;
231
232    data = acquire(registre);
233    longitude += data*10;
234    registre++;
235
236    data = acquire(registre);
237    longitude += data;
238    registre++;
239
240    data = acquire(registre);
241    minutes = data*10;
242    registre++;
243
244    data = acquire(registre);
245    minutes += data;
246    registre++;
247
248    data = acquire(registre);
249    minutes += data/10;
250    registre++;
251
252    data = acquire(registre);
253    minutes += data/100;
254    registre++;
255
256    data = acquire(registre);
```

```
257   minutes += data/1000;
258   registre++;
259
260   data = acquire(registre);
261   minutes += data/10000;
262   longitude = longitude + (minutes/60);
263   registre++;
264
265   data = acquire(registre);
266   letter = data;
267   if (letter == 'W')
268     longitude = -longitude;
269
270   return longitude;
271
272 }
273
274 /* Affichage du cap par rapport au Nord vrai */
275 void displayHeading()
276 {
277   byte headingDegHun, headingDegTen, headingDegUn, headingDegTenths;
278
279   // On initialise la lecture au niveau du registre 44
280   Wire.beginTransmission(GPSADDRESS);
281   Wire.send(44);
282   Wire.endTransmission();
283
284   // Demande de reception de la part du maitre
285   Wire.requestFrom(GPSADDRESS, 4);
286
287   // Lecture des registres
288   headingDegHun = Wire.receive();
289   headingDegTen = Wire.receive();
290   headingDegUn = Wire.receive();
291   headingDegTenths = Wire.receive();
292
293   // Affichage du cap
294   Serial.print("Heading : ");
295   Serial.print(headingDegHun, DEC);
296   Serial.print(headingDegTen, DEC);
297   Serial.print(headingDegUn, DEC);
298   Serial.print(",");
299   Serial.println(headingDegTenths, DEC);
300
301 }
302
303 /* Fonction renvoyant le cap en degres*/
304 float getHeading()
305 {
306   float heading = 0.0;
307   byte registre; // registre du GPS a lire
308   float data = 0.0; // donnees lues en provenance du GPS
309
310   registre = 44;
311
312   data = acquire(registre);
```

```
313   heading = data*100;
314   registre++;
315
316   data = acquire(registre);
317   heading += data*10;
318   registre++;
319
320   data = acquire(registre);
321   heading += data;
322   registre++;
323
324   data = acquire(registre);
325   heading += (data)/10;
326
327   return heading;
328
329 }
330
331
332 /* Affichage de la vitesse */
333 void displaySpeed()
334 {
335   byte speedHun, speedTen, speedUn, speedTenths;
336
337   // On initialise la lecture au niveau du registre 52
338   Wire.beginTransmission(GPSADDRESS);
339   Wire.send(52);
340   Wire.endTransmission();
341
342   // Demande de reception de la part du maitre
343   Wire.requestFrom(GPSADDRESS, 4);
344
345   // Lecture des registres
346   speedHun = Wire.receive();
347   speedTen = Wire.receive();
348   speedUn = Wire.receive();
349   speedTenths = Wire.receive();
350
351   // Affichage de la vitesse
352   Serial.print("Speed : ");
353   Serial.print(speedHun, DEC);
354   Serial.print(speedTen, DEC);
355   Serial.print(speedUn, DEC);
356   Serial.print(",");
357   Serial.print(speedTenths, DEC);
358   Serial.println(" km/h");
359
360 }
361
362 /* Fonction renvoyant la vitesse */
363 float getSpeed()
364 {
365   float vitesse;
366   byte registre; // registre du GPS a lire
367   float data = 0.0; // donnees lues en provenance du GPS
368
```

```
369   registre = 52;
370
371   data = acquire(registre);
372   vitesse = data*100;
373   registre++;
374
375   data = acquire(registre);
376   vitesse += data*10;
377   registre++;
378
379   data = acquire(registre);
380   vitesse += data;
381   registre++;
382
383   data = acquire(registre);
384   vitesse += data/10;
385
386   return vitesse;
387
388 }
389
390 /* Affichage de l'altitude */
391 void displayAltitude()
392 {
393   byte altTenThousands, altThousands, altHun, altTen, altUn;
394
395   // On initialise la lecture au niveau du registre 39
396   Wire.beginTransmission(GPSADDRESS);
397   Wire.send(39);
398   Wire.endTransmission();
399
400   // Demande de reception de la part du maitre
401   Wire.requestFrom(GPSADDRESS, 5);
402
403   // Lecture des registres
404   altTenThousands = Wire.receive();
405   altThousands = Wire.receive();
406   altHun = Wire.receive();
407   altTen = Wire.receive();
408   altUn = Wire.receive();
409
410   // Affichage de l'altitude
411   Serial.print("Altitude : ");
412   Serial.print(altTenThousands, DEC);
413   Serial.print(altThousands, DEC);
414   Serial.print(altHun, DEC);
415   Serial.print(altTen, DEC);
416   Serial.print(altUn, DEC);
417   Serial.println(" metres");
418
419 }
420
421 /* Fonction renvoyant l'altitude en metres */
422 float getAltitude()
423 {
424   float altitude = 0.0;
```

```
425   byte registre; // registre du GPS a lire
426   float data = 0.0; // donnees lues en provenance du GPS
427
428   registre = 39;
429
430   data = acquire(registre);
431   altitude = data*10000;
432   registre++;
433
434   data = acquire(registre);
435   altitude += data*1000;
436   registre++;
437
438   data = acquire(registre);
439   altitude += data*100;
440   registre++;
441
442   data = acquire(registre);
443   altitude += data*10;
444   registre++;
445
446   data = acquire(registre);
447   altitude += data;
448
449   return altitude;
450
451 }
```

D.9 Waypoint.h

```
1  #ifndef DEF_WAYPOINT
2  #define DEF_WAYPOINT
3
4  float getBearing(float latPlaneur, float longPlaneur,
5   float latWaypoint, float longWaypoint);
6
7  float getDistance(float latPlaneur, float longPlaneur,
8  float latWaypoint, float longWaypoint);
9
10 float conversionCap(float cap);
11
12 #endif
```

D.10 Waypoint.pde

```
1  /* Waypoint */
2
3  #include "Defines.h"
4  #include "Waypoint.h"
5  #include "GPS.h"
6
```

```
7  /* Fonction de calcul de cap a suivre pour rejoindre un point GPS */
8  float getBearing(float latPlaneur, float longPlaneur,
9  float latWaypoint, float longWaypoint)
10 {
11   float latPos, longPos, latWay, longWay;
12   float bearing = 0.0;
13
14   // Conversion en radians des coordonnees geographiques
15   latPos = radians(latPlaneur);
16   longPos = radians(longPlaneur);
17   latWay = radians(latWaypoint);
18   longWay = radians(longWaypoint);
19
20   bearing = atan2((sin(longWay - longPos)*cos(latWay)),
21   (cos(latPos)*sin(latWay) - sin(latPos)*cos(latWay)*cos(longWay - longPos)));
22
23   bearing = degrees(bearing);
24
25   if (bearing < 0)
26   {
27     bearing = 360 + bearing;
28   }
29
30   return bearing;
31 }
32
33 /* Fonction calculant la distance entre deux points GPS,
34 la valeur retournée est en km */
35 float getDistance(float latPlaneur, float longPlaneur,
36  float latWaypoint, float longWaypoint)
37 {
38   double latPos, longPos, latWay, longWay, delta, x, y;
39   double dist = 0.0;
40
41   // Conversion en radians des coordonnées géographiques
42   latPos = radians(latPlaneur);
43   longPos = radians(longPlaneur);
44   latWay = radians(latWaypoint);
45   longWay = radians(longWaypoint);
46
47   delta = longPos - longWay;
48
49   dist = 2*asin(sqrt(pow(sin((latPos - latWay)/2),2) +
50   cos(latPos)*cos(latWay)*pow(sin(delta/2),2)));
51
52   dist = RAYON*dist;
53   dist = (float) dist;
54
55   return dist;
56 }
57
58 /* Fonction convertissant le cap en un angle
59 compris entre -180 et + 180 degres */
60 float conversionCap(float cap)
61 {
62   float capConverti = 0.0;
```

```
63
64    if (cap > 180)
65    {
66      capConverti = -(360 - cap);
67    }
68
69    else
70    {
71      capConverti = cap;
72    }
73
74    return capConverti;
75  }
```

ENSIETA – ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS	**FICHE DE VALIDATION DU STAGE S2**	Stage technique d'assistant ingénieur d'une durée de 6 semaines minimum à réaliser entre 20 juin et 16 septembre 2011

NOM : **LE GRANVALET** Prénom : **Julien**

Promotion : **ENSI2012** Nationalité : **Française**

Branche : ISE

Statut : ☐ IETA (*) ☒ Civil ☐ Auditeur

E- mail : julien.le_granvalet @ensta-bretagne.fr
julien.lgv @gmail.com

Tél / portable (pour être joint rapidement si besoin) : 0689355629

Nom de l'organisme d'accueil :
ENSTA BRETAGNE

☐ Entreprise ☐ Université ☒ Autre

Pays : **FRANCE**

Impératif : 5 semaines mini, hors stage linguistique, à réaliser en pays non-francophone au cours du cursus ENSI)

Domaine d'activité:

Langue parlée du stage : Français

Adresse complète et précise (demander à votre contact s'il y a lieu de préciser le nom d'une personne ou un bureau qui centralise les conventions au sein de l'organisme) :

Téléphone :

Portable :

Fax :

Réfs d'un contact administratif (RH) : ☐ M. ☐ Mme ☐ Mlle

NOM : Prénom :

Mail :

Sujet de stage : (synthèse en 2 lignes) puis agrafer un descriptif plus détaillé si nécessaire

Le stage a pour but de concevoir, construire et tester un robot planeur. Un tel robot doit puiser son énergie dans son environnement en captant les courants ascendants. Entre deux phases ascendantes, le robot devra chercher à se déplacer afin d'effectuer le parcours requis.

(*) Pour les élèves IETA, joindre l'Ordre de Mission

Tuteur de l'organisme d'accueil

☒ M. ☐ Mme ☐ Mlle

NOM : **JAULIN** Prénom : **Luc**

Téléphone: 0298348910

Mail : luc.jaulin@ensta-bretagne.fr

Dates précises du stage (JJ/MM/AAAA)

Date de début : **12/06/2011**

Date de fin : **22/07/2011**
(dates de fermeture de l'organisme d'accueil…) : **22/07/2011**

Stage rémunéré ou gratifié

☐ oui

☒ non

Avis du responsable Langues et culture (si pays non-francophone)	Commentaires :	Date :	Signature :
Avis du responsable de branche	*Vérifie avec l'élève la cohérence des dates du stage.* Commentaires :	Date :	Signature :

Fiche informatique : l'élève saisit (champs), imprime, récupère les visas nécessaires puis dépose la fiche au bureau des stages C006 à la Direction de la Formation (DE/Sco/Stages) afin d'éditer la convention de stage officielle.

000000222 V1.6
Notifié le 30/09/2010

ENSIETA
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport dès la fin du stage à / *Please return this report at the end of the internship to :*
ENSIETA – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – France
☎ 00.33 (0) 2.98.34.87.70 - Fax 00.33 (0) 2.98.38.87.46 - stages@ensieta.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* ENSTA BRETAGNE

Adresse / *Address* _____

_____

Tél / *Phone (with country and area code)* _____

Fax / *Fax (with country and area code)* _____

Nom du superviseur / *Name of the person in charge of the placement*
M. Luc JAULIN

Fonction Professeur des universités

Adresse e-mail / *E-mail address* _____

Nom du stagiaire accueilli / *Name of the trainee* Julien LE GRANVALET

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une cote, en encerclant le chiffre approprié, pour chacune des caractéristiques suivantes. Cette note devra se situer entre 0 (très faible) et 5 (très bien) /
Please give a mark between 0 (very weak) and 5 (very good).

MISSION / TASK

- La mission de départ a-t-elle été remplie ? 0 1 2 3 4 (5)
  Has the task been carried out well ?
- Le stagiaire a-t-il apporté des connaissances nouvelles à l'organisme d'accueil ?
  Did the trainee bring news skills to the host organisation ? ☒ oui/*yes* ☐ non/*no*
  Lesquelles ? *Which ones ?* Compétence planeur + robotique
- Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☒ non/*no*

5.1.1.1.1.1 Was the trainee lacking skills ?

Si oui, lesquelles ? / *Which ones ?* _____

ESPRIT D'EQUIPE / TEAM SPIRIT

- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil / Did the trainee easily integrate into the host organisation? oui

5.1.1.1.1.2 (disponible, sérieux, adaptation au travail de groupe) ***0 1 2 3 4 (5)***
(flexible, serious, adapt himself (herself) to the team work)

Avez vous des observations ou suggestions à nous faire part / Have you any remark or suggest to comment ?

COMPORTEMENT AU TRAVAIL / BEHAVIOUR TOWARDS WORKS

Le comportement du stagiaire était-il conforme à vos attentes *(Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances)* ?
Was the trainee in concordance with your attend (*Punctual, methodical , responsive to management instructions, concerned with quality, concerned about gaining new skills) ?*

0 1 2 3 4 (5)

Avez vous des observations ou suggestions à nous faire part / Have you any remark or suggest to comment ?

INITIATIVE – AUTONOMIE / INITIATIVE – AUTONOMY

Le stagiaire s'adaptait vite à de nouvelles situations ? 0 1 2 3 4 (5)
(Proposer des solutions aux problèmes rencontrés, autonome dans son travail,..)

Did the trainee adapted himself (herself)very well to new situations ? 0 1 2 3 4 (5)
(Suggest solutions to problems met, Was independent in his/her job…)

Avez vous des observations ou suggestions à nous faire part / Have you any remark or suggest to comment ?

CULTUREL – COMMUNICATION / CULTURAL – COMMUNICATION

Le stagiaire était-il ouvert, d'une manière générale, à la communication 0 1 2 3 4 (5)
Was the trainee open to listen and express himself (herself)

Avez vous des observations ou suggestions à nous faire part / Have you any remark or suggest to comment ?

OPINION GLOBALE / OVERALL ASSESSMENT

❖ La valeur technique du stagiaire était : 0 1 2 3 4 (5)
The technical skills of the trainee were :

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?
Are you ready to welcome another trainee next year ? ☒ oui/*yes* ☐ non/*no*

Fait à Brest ,le 26/09/2011
In ,On

Signature

Merci pour votre coopération
We thank you very much for your cooperation

Table des figures

Bibliographie

[1] http://www.arduino.cc

[2] http://a190754.free.fr/PROFILS.HTM

[3] http://aerodynamique.chez.com/

[4] http://diydrones.com/profiles/blogs/ardupilot-main-page

[5] http://williams.best.vwh.net/avform.htm

[6] http://soaring.goosetechnologies.com/

[7] http://www.ensta-bretagne.fr/