RAPPORT DE PROJET DE FIN D'ETUDE :

Pilotage automatique d'un drone

Encadrants :

Aziz NAKRACHI

Claudine LECOCQ

AYOUB Abass
VISCOGLIOSI Alexis

IMA5
Année universitaire 2018-2019

Table des matières

Remerciements

Nous tenons à remercier Mr Aziz NAKRACHI et Mme Claudine LECOCQ, pour nous avoir donné la chance de réaliser ce projet.

Nous les remercions de leurs confiances tout au long de l'année concernant ce projet de fin d'étude. Ils ont apporté une ambiance de travail studieuse mais décontractée à la fois.

La présence de nos encadrants a été primordiale tant dans la finalité mais aussi dans le lancement du projet. Leurs critiques constructives nous ont permis de toujours mieux se situer tout au long de l'aventure.

Nous remercions également le second binôme de projet, Claire VANDAMME et Justine SENELLART, pour leur coopération et l'échange de connaissance dans les domaines sollicités.

Un remerciement sincère à Mr Olivier SCRIVE, pour sa disponibilité et sa confiance.

Nous remercions tous les professeurs de Polytech Lille et ceux de nos cursus précédents pour les connaissances acquises jusqu'aujourd'hui.

Enfin, nous remercions Mr Guy REUMONT, directeur de Polytech Lille, pour sa confiance et l'opportunité d'effectuer notre cursus d'ingénieur en Informatique Microélectronique et Automatique.

Introduction

Dans le cadre de notre dernière année à Polytech Lille en Informatique Microélectronique et Automatique, nous réalisons un projet de fin d'étude tout au long de notre année. Nous avons choisi de réaliser le projet numéro 36 qui a pour but : Le pilotage automatique d'un drone.

Afin de réaliser ce projet, nous avons à disposition un drone Bebop 2 de la marque Parrot. Celui-ci nous a été fourni par l'API à travers nos encadrants.

Le travail que nous devons réaliser en premier lieu sur ce drone est une étude de faisabilité en termes de contrôle, d'asservissement en vitesse ou en position et ce qui, au contraire, n'est pas modifiable. Nous travaillons en collaboration avec le binôme du projet 21 composé de Justine Senellart et Claire Vandamme. Celui-ci qui se concentrera sur la partie traitement d'images. L'intérêt premier de ce projet est de faire un bilan des capacités et de l'ouverture du drone. Dans un second lieu, après validation de l'étude de faisabilité, nous avons réalisé le déplacement automatiquement du drone d'un point A à un point B avec un asservissement en vitesse. Ce travail réalisé donnera une visibilité aux différents encadrants pour des futurs projets de recherche ou de TP.

Dans ce rapport, nous allons dans un premier temps faire une présentation du projet incluant le contexte, la présentation du cahier des charges et la présentation du drone. Dans un deuxième temps, nous détaillerons le travail que nous avons effectué sur les différentes façons de communiquer avec le drone et les conclusions que nous en avons tiré. Enfin, dans une troisième et dernière partie, nous expliciterons le travail effectué sur l'asservissement en vitesse entre deux points GPS.

I) Présentation du projet et son cahier des charges

Dans cette première partie, nous présenterons plus en détails notre projet et son cahier des charges.

A) Présentation du contexte et du matériel

Nous travaillons sur le pilotage et le contrôle automatique d'un drone Bebop 2 de la marque Parrot. Notre travail consistera à effectuer une étude de faisabilité en termes de contrôle et d'asservissement mais également sur d'autres capacités du drone s'il en existe. Le travail réalisé sera source d'inspiration à des fins de recherches pour le laboratoire de recherche CRISTAL mais aussi de mise en place de Travaux Pratiques pour les modules de Robotique et/ou Automatique.

Figure 1 : Photo du drone

Les caractéristiques techniques de ce drone sont :

Capteur :

- Magnétomètre 3 axes (AKM 8963)
- Gyroscope 3 axes (MPU 6050)
- Accéléromètre 3 axes (MPU 6050)
- Capteur optique
- Capteur Ultrason
- Capteur de pression (MS 5607)

Batterie :

- Lithium Polymer 2700 mAh
- Autonomie de vol : 22 min

Processeur :

- Parrot P7 dual-core CPU Cortex A9
- Quad core GPU
- 8Gb flash memory

Autres caractéristiques :

- Poids : 510.7g
- OS : Linux, kernel 3.4.11
- 4 moteurs Brushless
- Camera (définition de la vidéo :1920x1080p ; définition de la photo : 3800x3188p)
- GNSS (GPS + GLONASS + Galileo, Baidu)

B) Présentation du cahier des charges du projet

Partant d'une étude de faisabilité, notre cahier des charges nous laisse une certaine marge de manœuvre sur le lancement du projet :

Matériel utilisé :

Nous utiliserons un drone Parrot Bebop 2. La première étape consiste à savoir comment utiliser. A la suite des résultats obtenus, nous nous concentrerons sur une méthode précise. Nous travaillerons sous Linux Ubuntu 16.04 avec ROS qui est un outil basé sur le SDK, utilisé par les roboticiens et les automaticiens. Celui-ci accepte les langages de programmation de Python et C++. Nous passerons par le langage de programmation Python. Nous envisagerons selon les résultats acquis, de travailler avec Matlab, Oracle et/ou d'autres logiciels afin d'extraire le maximum de potentiel de ces travaux.

Contraintes :

Tout d'abord nous n'avons pas accès à toutes les données du drone. Ce projet est réalisé avec un second binôme, le travail partagé est une légère contrainte mais également un plus s'il est bien fait. Ensuite, nous devons nous adapter aux différents environnements du drone et apprendre les différents outils qui sont à disposition. Le fait d'avoir peu d'informations sur des projets similaires va nous contraindre à réaliser de nombreuses recherches et de nombreux tests. Nous nous contraindrons à travailler en temps réel si possible.

Résultats attendus :

Nous effectuerons une étude complète afin de savoir quelles sont les limites du drone et les moyens de le contrôler. Pour ce faire nous avons divisé notre travailler en sous-systèmes qui sont :

- Contrôle et envoi de commandes basiques
- Récupération des données odométriques, GPS, vitesse
- Exploitation des données précédemment citées
- Asservissement en vitesse entre deux points GPS

II) Travail effectué

Dans cette seconde partie, nous allons dans un premier temps vous présenter les différentes solutions que nous avons testé, expliquer leurs modes de fonctionnement et dire ensuite si nous les avons retenus ou pas et pour quelles raisons.

Nous allons tout d'abord présenter le SDK de Parrot, ensuite Pyparrot, viendra après ROS avec Bebop Autonomy.

A) SDK

Le SDK est une bibliothèque de fonctions complètes qui nous permet de nous connecter, de piloter, recevoir en direct de la caméra, sauvegarder et télécharger des médias (photo ou vidéo), envoyer des plans de vol, de piloter automatiquement et de mettre à jour le drone. Peu d'informations issues des capteurs et des actionneurs sont accessibles avec le SDK dont la commande des moteurs. Néanmoins, Parrot met à dispositions des commandes permettant de réaliser des figures ou des tâches complexes tels que des flips ou le trajet d'un point A à un point B.

Nous avons décidé de pousser un peu les recherches sur le SDK afin de voir si nous pouvons réaliser, par la suite, un asservissement en position. Nous nous sommes rendu compte que le SDK est un outil très fermé bien qu'il soit très proche du drone. Nous avons donc concentré nos recherches sur la récupération de données via SDK : Il est, en théorie, possible de récupérer :

- latitude (double): Position en latitude au dixième de degré

- longitude (double): Position en longitude au dixième de degré

- altitude (double): Altitude en mètres

- speedX (float): Vitesse relative au Nord en m/s (Quand le drone se déplace vers le Nord, vitesse > 0)

- speedY (float): Vitesse relative à l'EST en m/s (Quand le drone se déplace vers l'Est, une vitesse > 0)

- speedZ (float): Vitesse sur l'axe Z (Quand le drone passe d'une position haute à une position basse, vitesse > 0) (en m/s)

- roll (float): Valeur du mouvement en roulis (en radian)

- pitch (float): Valeur de tangage (en radian)

- yaw (float): Valeur de lacet (en radian)

- longitude_accuracy (i8): Erreur de localisation en longitude (en mètre)

- latitude_accuracy (i8): Erreur de localisation en latitude (en mètre)

- altitude_accuracy (i8): Erreur de localisation pour l'altitude (en mètre)

- Image en directe grâce à la caméra avant

Nous avons cherché un moyen de récupérer ces données de capteurs. Pour cela, nous avons repris l'exemple de programme proposé par Parrot pour le Bebop. Dans ce programme, on reçoit les valeurs des capteurs lorsqu'il y a un changement d'état cependant ces valeurs ne sont jamais affichées ou sauvegardées. Nous avons donc ajouté l'écriture de ces valeurs dans un fichier. Cependant, lors de la lecture du fichier les données ne sont pas cohérentes : toutes les valeurs proviennent du même capteur et ne changent jamais. De plus, les capteurs sont identifiés par un nombre et non par un nom on ne peut donc pas identifier lequel des capteurs envoie les données.

On peut donc conclure que cet outil est très polyvalent en termes de fonctionnalités de base mais nous n'avons pas pu le garder pour des raisons de temps réel et de récupération de données concrètes comme les données odométriques par exemple ou GPS.

B) PyParrot

Dans cette deuxième sous-partie, nous présentons une des options que nous avons envisagées pour récupérer les données. Cette option est Pyparrot. Pyparrot est basé sur le SDK de Parrot et qui permet de contrôler le drone avec des programmes Python. Afin de tester, dans un premier temps, cette solution, nous avons utilisé les programmes de base, permettant de faire décoller le drone et de le faire atterrir. Les

programmes Python contiennent ici uniquement les commandes de base et ne s'occupent pas de la connexion avec le drone. Ce dernier est géré par le SDK.

Après avoir réussi ces dernières fonctions, nous avons exploré les fonctions disponibles dans le projet Pyparrot.

Parmi les fonctions testées, nous avons trouvé celle-ci :

```
def extract_sensor_values(self, data):
    """
    Extract the sensor values from the data in the BLE packet
    :param data: BLE packet of sensor data
    :return: a list of tuples of (sensor name, sensor value, sensor enum, header_tuple)
    """
    sensor_list = []
    #print("updating sensors with ")
    try:
        header_tuple = struct.unpack_from("<BBH", data)
    except:
        color_print("Error: tried to parse a bad sensor packet", "ERROR")
        return None
```

Figure 2: Fonction Extract_values

Cette fonction nous permettrait, en théorie de récupérer les données de capteurs. Malheureusement, nous n'avons pas reussi à récupérer les données souhaitées.

En effet, nous pouvions récupérer soit les valeurs une fois le drone posé mais ces données ne nous permettent pas de faire un contrôle en temps réel. Sinon, nous récupérons des données non-exploitables comme le pourcentage de charge.

Ce programme se basant sur le SDK de Parrot, nous avons donc rencontré les mêmes problématiques que précédemment, c'est-à-dire, de ne pas recevoir les données des différents capteurs en temps réel.

L'option PyParrot est alors écartée des options envisagées à étudier et approfondir dans le cadre de notre étude de faisabilité.

C) Robot Operating System (ROS)

Nous allons tout d'abord faire une présentation de ROS puis mettre en avant Bebop Autonomy et les résultats obtenus.

ROS est structuré de la manière suivante :

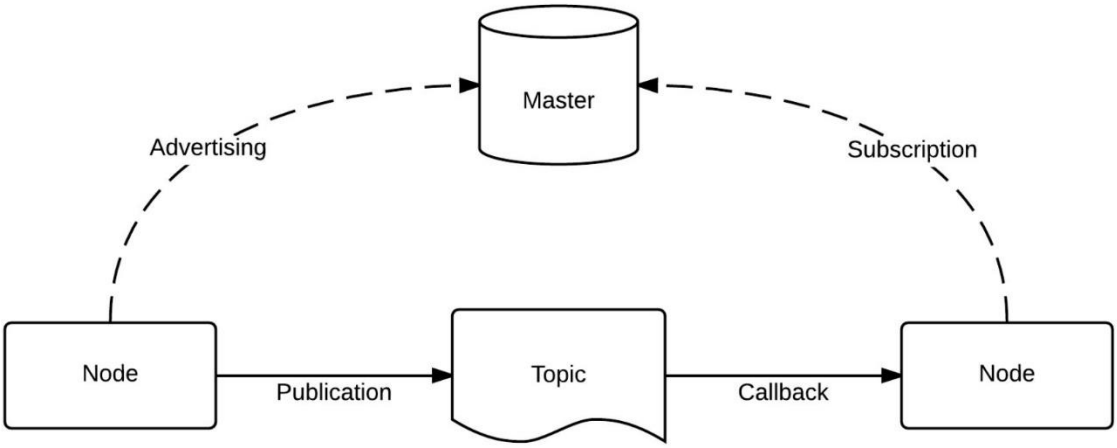

Figure 3 : structure de ROS

ROS propose une architecture souple permettant la communication entre les processus et entre les machines. Ces processus sont appelés « nodes » ou « nœuds ». Un nœud peut être par exemple un capteur, un moteur ou encore un algorithme. Chaque nœud peut être appelé d'une façon ou d'une autre en fonction de l'action qu'il réalise. Un nœud qui publie des données est un « publisher ». Un nœud qui souscrit à des données est un « subscriber ».

La communication entre chaque nœud se fait via des topics. Un topic est un système de transport de l'information basé, comme dit précédemment, sur le système de publisher/subscriber. Un topic est standardisé (le type d'informations qui est publié sur le topic est toujours formé de la même manière) et sert de bus d'informations. Cette communication entre 2 nœuds est gérée par un Master.

Un master est une base de données permettant aux différents nœuds de s'enregistrer et donc de se connaitre entre eux. La communication d'un message se fait comme ceci :

- Le premier nœud avertit le master qu'il a une donnée à publier
- Le deuxième nœud avertit le master qu'il souhaite souscrire à une donnée
- La connexion entre les 2 nœuds est créé

Un nœud peut être à la fois « publisher » et « subscriber ».

Le topic est donc un mode de communication asynchrones permettant la communication entre plusieurs nœuds. Il existe néanmoins un autre mode de communication qui est le Service. Le service est un mode de communication qui permet la communication synchrone entre 2 nœuds.

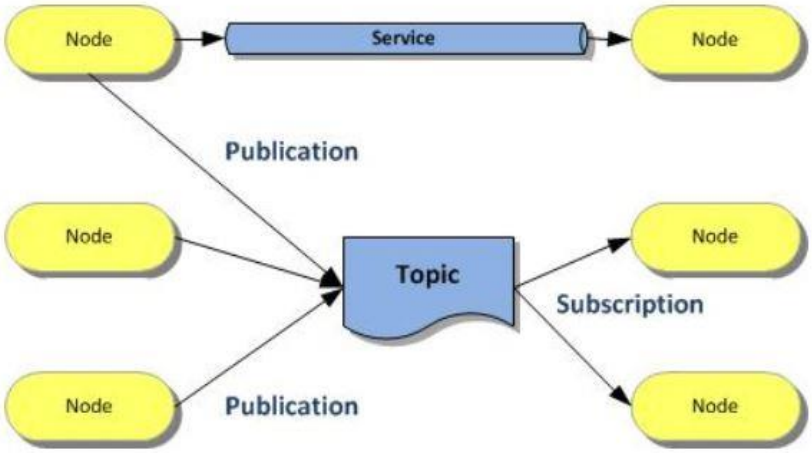

Figure 4 : Fonctionnement de ROS

Concernant l'utilisation de fichier avec ROS, il existe 2 concepts : celui de package et celui de stack.

Le plus couramment utilisé est le package. Un package est un répertoire de nœuds. Il possède également les librairies externes, les données…

Le stack est une collection de package permettant des fonctions plus complexes comme la localisation… L'un des intérêts de ces fichiers est le fait d'être des exécutables. Cela signifie que le non-fonctionnement de l'un d'entre eux pour une quelconque raison n'entraine pas de problèmes sur les autres, vu qu'ils sont tous indépendants les uns des autres.

Après cette introduction sur le fonctionnement de ROS, nous allons développer les points que nous avons étudié. Nous nous y sommes particulièrement intéressés pour plusieurs raisons. Tout d'abord, ROS est utilisé par une très grande communauté de roboticiens et d'automaticiens, il possède alors un très grand nombre de fonctions open source. Il est un standard en termes de robotique et il est donc très intéressant pour nous de nous y intéresser pour acquérir des compétences dans ce domaine.

L'intérêt de ce projet est dans un premier temps d'être capable de récupérer des données odométriques, GPS, etc… en temps réel si possible pour les exploiter.

En complément de ROS, nous utilisons un autre second outil, Bebop Autonomy. Bebop Autonomy est un driver de ROS nous permettant d'exécuter des commandes de bases comme décoller, atterrir, exécuter des flips… Les commandes pour exécuter ces fonctions sont du type :

rostopic pub --once [namespace]/[topic] [typeMessage]/[Message], pour publier sur un topic

rostopic echo [namespace]/[topic], pour souscrire à un topic

On retrouve ci-dessous la liste des différents topics qui sont utilisé pour les « publishers » et les « suscribers ».

```
abass@abass-HP-Pavilion-dv7-Notebook-PC:~/catkin_ws/src/projetbebop/src$ rostopic list
/bebop/autoflight/navigate_home
/bebop/autoflight/pause
/bebop/autoflight/start
/bebop/autoflight/stop
/bebop/bebop_driver/parameter_descriptions
/bebop/bebop_driver/parameter_updates
/bebop/camera_control
/bebop/camera_info
/bebop/cmd_vel
/bebop/fix
/bebop/flattrim
/bebop/flip
/bebop/image_raw
/bebop/image_raw/compressed
/bebop/image_raw/compressed/parameter_descriptions
/bebop/image_raw/compressed/parameter_updates
/bebop/image_raw/compressedDepth
/bebop/image_raw/compressedDepth/parameter_descriptions
/bebop/image_raw/compressedDepth/parameter_updates
/bebop/image_raw/theora
/bebop/image_raw/theora/parameter_descriptions
/bebop/image_raw/theora/parameter_updates
/bebop/joint_states
/bebop/land
/bebop/odom
/bebop/record
/bebop/reset
/bebop/set_exposure
/bebop/snapshot
/bebop/states/ardrone3/CameraState/Orientation
/bebop/states/ardrone3/GPSState/NumberOfSatelliteChanged
/bebop/states/ardrone3/MediaStreamingState/VideoEnableChanged
/bebop/states/ardrone3/PilotingState/AltitudeChanged
/bebop/states/ardrone3/PilotingState/AttitudeChanged
/bebop/states/ardrone3/PilotingState/FlatTrimChanged
```

Figure 5: : Liste des topics disponibles

II) Déplacement entre 2 points avec asservissement en vitesse

A) Traitement des données du drone

Données odométriques :

L'odométrie est une technique permettant d'estimer la position d'un objet en mouvement. Elle permet alors de mesurer le déplacement du drone en fonction de son point de départ. Ces données nous étaient relativement utile car elle nous permettait de savoir par exemple la hauteur en mètre à laquelle se trouve le drone ou encore ses autres déplacements. Nous pensions grâce à ces données effectuer un asservissement en position. Il se trouve que cette caractéristique est déjà intégrée lorsque le drone est en l'air mais sans consigne de commande sur les moteurs. Nous avons étudié le type de donnée que nous recevons lorsqu'on souscrit au topic /bebop/odom.

```
abass@abass-HP-Pavilion-dv7-Notebook-PC:~/catkin_ws/src/projetbebop/src$ rosmsg show Odometry
[nav_msgs/Odometry]:
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

Figure 6: Types de donnée dans le topic /bebop/odom

Afin de de récupérer dans une variable la valeur de « position x » issue du message du topic /bebop/odom, on passe par les différents attributs entourés en rouge sur l'image. Cela donne alors : *x = msg.pose.pose.position.x*

Néanmoins, les données odométriques ne pouvaient pas à elles seules garantir le déplacement automatique du drone car nous n'avions aucun moyen de repérer le drone dans l'espace. Nous remarquons également qu'il était possible de récupérer, sur ce même topic, les valeurs angulaires du drone.

Donnée GPS :

Nous avons décidé de passer par les coordonnées GPS afin de localiser le drone avec la longitude et la latitude. Afin de récupérer les données GPS, nous avons utilisé 2 méthodes. La première était de réaliser une écoute, donc une souscription à un topic. Nous effectuons cette tâche avec la commande *rostopic echo* sur le topic /bebop/fix (topic contenant les données GPS) et d'afficher la latitude et la longitude grâce à la commande suivante :

Rostopic echo /bebop/fix/latitude && Rostopic echo /bebop/fix/longitude

Cette commande nous permettait, lors de différents tests de visualiser les données GPS sans les stocker. Afin de pouvoir exploiter les données GPS, nous avons le choix d'utiliser la même méthode précédente utilisée pour les données odométriques c'est-à-dire d'utiliser le « suscriber » dans le programme python ou sinon d'exporter ses données dans un fichier CSV que nous ouvrons avec Excel : *Rostopic echo /bebop/fix/latitude > excel.csv*

Dans la suite du projet, nous avons utilisé ces 2 méthodes selon la finalité. Soit pour exploiter, soit pour visualiser un résultat.

À la suite des données récupérées, nous avons localisé ces coordonnées sur google Map afin de s'assurer de la véracité de l'information. Il s'avère que celle-ci est correcte. A compter de cet instant, nous savons qu'il est possible de récupérer les coordonnées GPS. Nous restons tout de même à une approximation de quelques mètres.

De la même manière que précédente, nous souscrivons au topic /bebop/fix et nous récupérons les valeurs de la latitude et la longitude. Celles-ci restent tout de même difficilement exploitable lorsqu'il s'agit de commander le drones sur ces axes

X, Y et Z. Pour ce faire, nous avons décidé de transposer ces positions GPS sur un repère XYZ.

Transposition des données GPS :

Avant de transposer ces données sur un repère, nous avons mesuré la distance qui séparait 2 points GPS. Pour cela, nous avons utilisé la formule suivante :

*Distance(A, B) = R * arccos (sin (lata) sin * (LATB) + cos (lata) cos * (LATB) cos * (Lona-lonB))*

Où lata est latitude du point A, LATB celle du point B, Lona la longitude du point A, lonB celle du point B et R le rayon de la terre en km = 6372.

Cette formule nous retourne une distance en kilomètre que nous avons donc converti en mètre pour avoir des résultats plus agréables à exploiter.

Concernant la transposition des données GPS, nous nous sommes basés sur le fait que tout point géographique sur un globe terrestre (défini par sa longitude et sa latitude) peut être représenté dans un maillage quadrillé (un repère cartésien en 3D), avec 3 coordonnées spatiales : X, Y, Z.

Nous avons donc pu grâce au système d'équations suivant convertir la longitude et la latitude en coordonnées cartésiennes :

$$X = R \times \cos l \times \sin L$$

$$Y = R \times \cos l \times \cos L$$

$$Z = R \times \sin l$$

L = longitude

l = latitude

R = rayon terrestre = 6371 km (rayon de référence en sciences de la Terre)

Les angles sont en degrés, ils sont convertis en radians, en multipliant les angles par le coefficient pi/180. Dans les calculs, l'unité de distance utilisée est le

kilomètre. Comme pour la distance, nous les avons convertis en mètre pour la suite du projet.

Cette conversion nous permet de transposer ces coordonnées GPS sur un axe X, Y et Z où l'origine est le centre de la terre. Elle nous permet également de mieux jouer sur les axes du drone pour contrôler les déplacements.

B) Exploitation des données du drone

Nous allons maintenant commencer la deuxième sous-partie concernant le traitement des données du drone.

Détermination de la fonction de transfert

Une fois les données de vitesse récoltées, nous avons tenté de trouver un lien entre notre consigne d'entre comprise entre -1 et 1 et la vitesse en m/s du drone. Nous avons donc exporté ces données sur Excel afin de voir l'évolution de la vitesse au cours du temps pour une consigne fixe.

Nous avons décidé, pour avoir des données plus précises, de réaliser les exportations pour plusieurs valeurs de consigne pour ensuite moyenner les résultats et limiter les erreurs dues aux perturbations extérieures.

Les résultats sont les suivants :

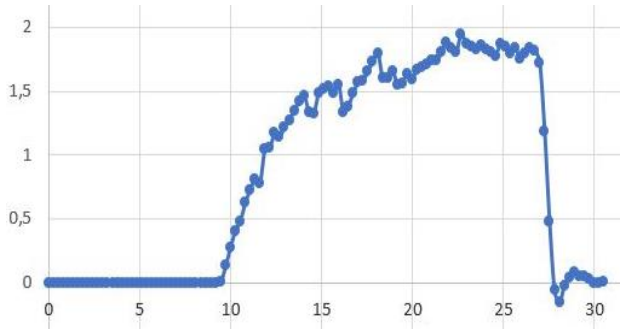

Figure 7: Réponse de la vitesse à un échelon de 0.20

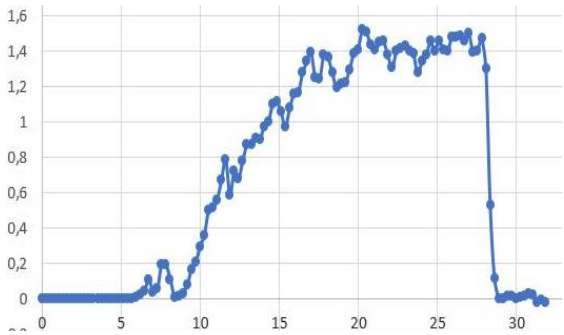

Figure 8: Réponse de la vitesse à un échelon de 0.15

Nous remarquons que pour différentes valeurs de consigne, nous obtenons des courbes qui semblent similaires, ce qui nous permet d'en déduire que le lien entre

notre consigne et la réponse en sortie est une fonction de transfert et en observant l'allure de la courbe, ce serait dont une fonction de transfert du premier ordre.

On rappelle qu'une fonction de transfert du premier ordre est de cette forme ci :

$$\frac{Y(p)}{X(p)} = \frac{K}{1 + \tau * p}$$

Où K est le gain statique

T = la constant de temps du système

Afin de déterminer graphiquement ces constantes, nous procédons de la manière suivante :

- Pour déterminer K, nous regardons la valeur à laquelle tend le système puis nous divisons par la consigne d'entrée
- Pour déterminer T, nous cherchons le temps que met le système à se stabiliser entre 95% de K et 105% de K. Cette valeur nous permet d'avoir 3 T, nous divisons ensuite juste par 3 pour avoir la constante de temps.

  Nous obtenons donc la fonction de transfert suivante :

$$\frac{Y(p)}{X(p)} = \frac{9}{1 + 3.1 * p}$$

Evolution de X, Y et Z

Après avoir déterminé la fonction de transfert liant la consigne et la vitesse, nous nous sommes concentrés sur l'évolution des coordonnées X, Y et Z déterminées par les équations utilisant les points GPS.

Le but est de d'analyser l'évolution de l'odométrie du drone en fonction des consignes que nous lui imposons. Cela nous permettra par la suite de contrôler les déplacements du drone.

Voici les résultats obtenus :

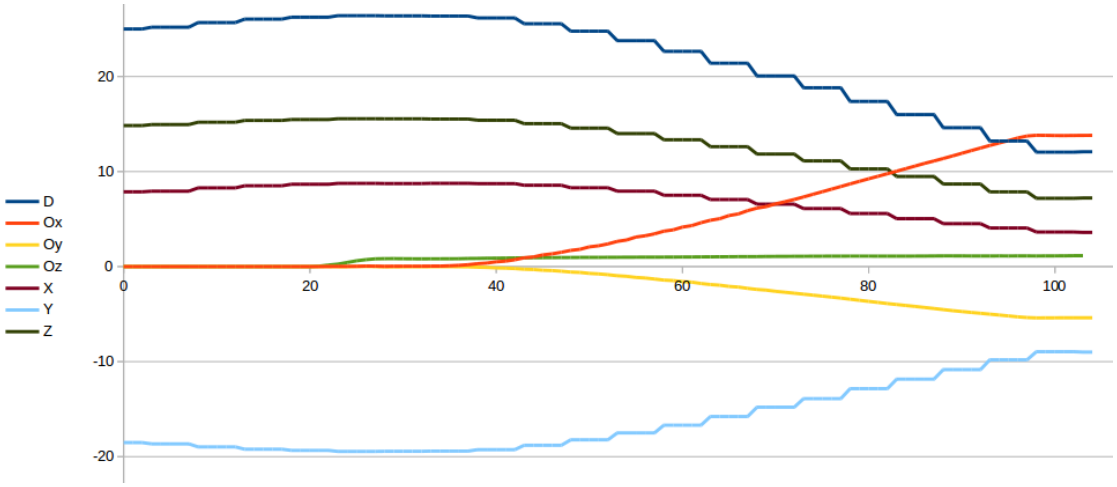

Figure 9 : Consigne de 0.15 sur x

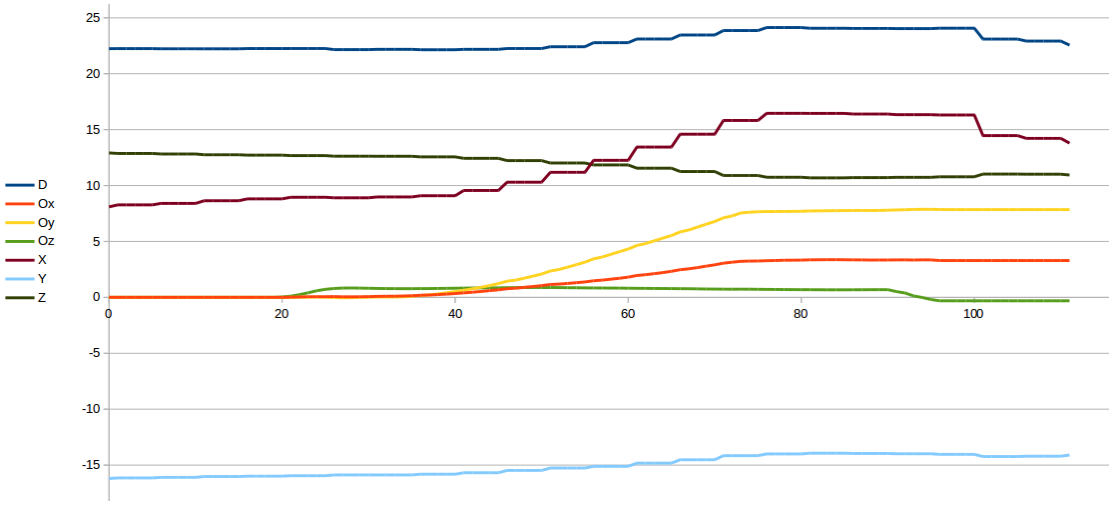

Figure 10 : Consigne de 0.15 sur y

Sur ces 2 figures, nous imposons une consigne de 0.15 sur x puis sur y. Nous observons donc la distance restante pour arriver à un point donné D, les données odométriques du drone Ox, Oy et Oz ainsi que l'évolution des données cartésiennes que nous avons déterminé X, Y et Z.

On remarque dans la première figure qu'en faisant avancer le drone sur x, sa valeur odométrique Ox, évolue naturellement mais aussi que nous avons une évolution sur X et Y. De même avec la seconde figure.

On peut donc constater que nous devons donc travailler sur 2 repères cartésiens différents. En effet, le drone possède son propre repère et nous souhaitons travailler

sur un autre repère. L'origine du repère où l'on transpose les coordonnées GPS est le centre de la Terre. Nous avons émis l'hypothèse que l'un des axes X ou Y de ce repère correspond à un point cardinal.

Nous avons effectué les mêmes tests en orientant cette fois ci le drone vers le Nord.

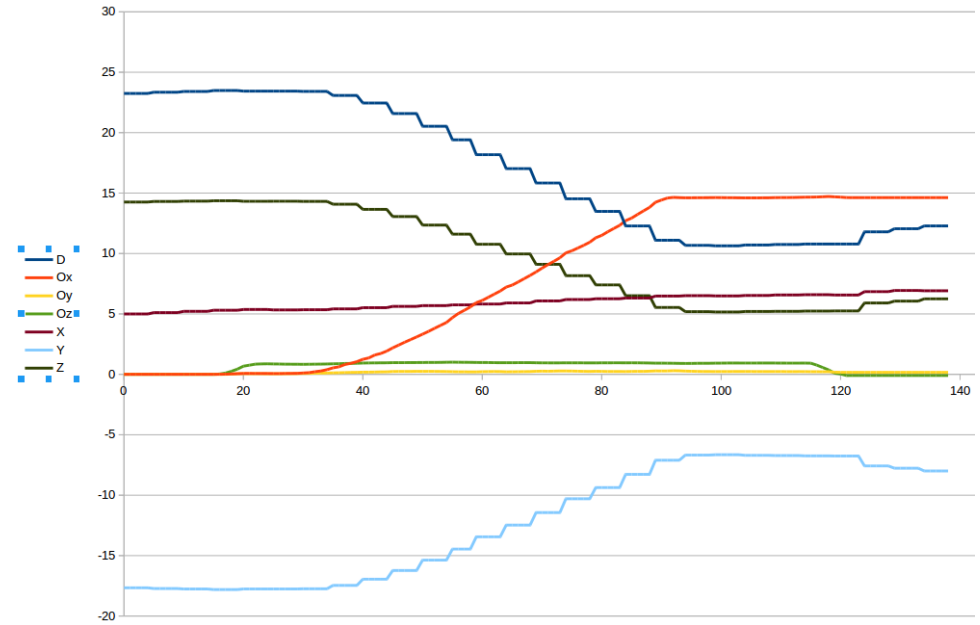

Figure 11 : Consigne de 0.15 sur x avec le drone orienté vers le Nord

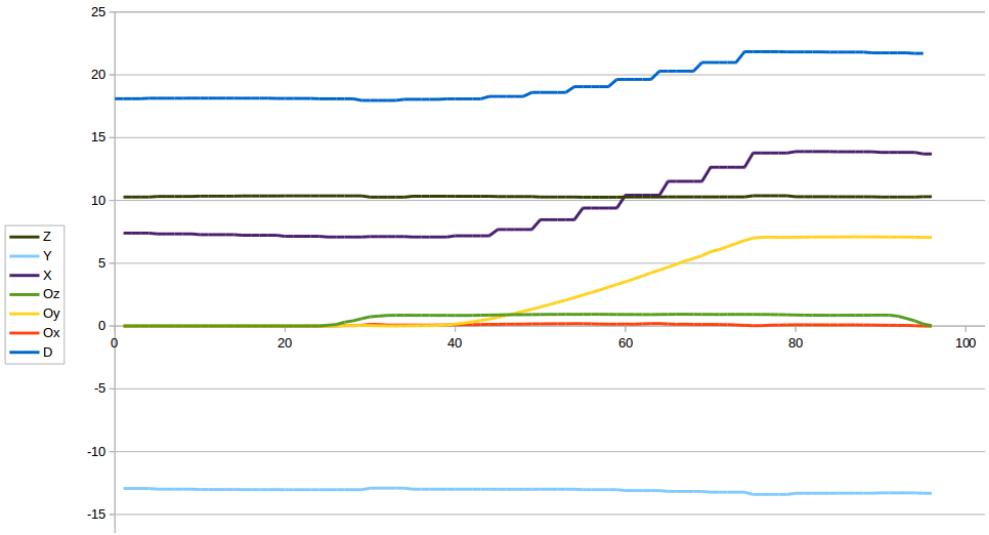

Figure 12 : Consigne de 0.15 sur y avec le drone orienté vers le Nord

En observant les 2 figures, nous constatons que notre hypothèse s'avère être correct. Un seul axe évolue lorsque nous imposons une consigne sur X ou Y. Nous concluons donc que notre repère possède son axe Y orienté vers le nord.

C) Les différentes fonctions

Dans cette troisième et dernière partie, nous présentons les différentes fonction mises en place. Nous avons travaillé sur l'asservissement en vitesse du drone. Le drone adapte sa vitesse en fonction de la distance qui lui reste à parcourir. L'une des premières fonctions que nous avons donc dû réaliser est le déplacement du drone entre 2 points.

Déplacement entre 2 points

Pour réaliser ce déplacement, nous avons décidé d'utiliser les données GPS. L'intérêt d'utiliser les données GPS est que nous pouvons facilement les récupérer sans forcément avoir besoin du drone pour cela. Comme nous l'avons expliqué précédemment, nous récupérons la positions GPS du drone en utilisant le topic /bebop/fix qui nous fournit la position du drone toutes les secondes. Concernant la position GPS d'arrivée, nous la récupérons simplement avec Google Map.

Afin de réaliser le déplacement, nous avons mis en place le diagramme fonctionnel suivant :

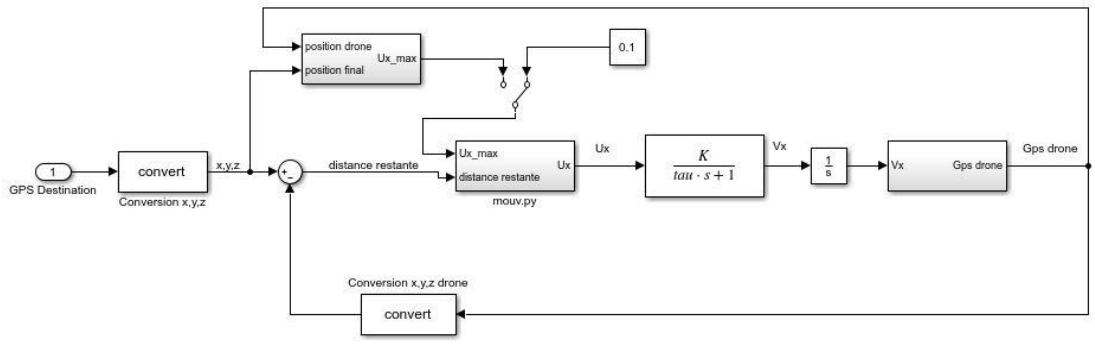

Figure 13 : Schéma bloc du programme

Nous utilisons les transpositions des points GPS sur le repère X Y Z pour effectuer cette tâche. Nous faisons la différence entre les 2 points convertis sur le même repère afin d'avoir la distance à parcourir sur X, Y et Z. Une fois ces données collectées, nous comparons les valeurs absolues de X et Y afin de déterminer sur quel axe nous devons parcourir la plus grande distance.

L'étape d'après est de faire le ratio entre la plus grande et la plus petite valeur entre x et y. Ce ratio influera la vitesse de l'axe sur lequel nous devons parcourir une plus petite distance. Dans un premier temps, on impose la consigne max du système afin de ne pas aller trop vite et d'adapter la seconde consigne pour aller dans la bonne direction. Afin de permettre au drone d'aller dans n'importe quelle direction, nous avons adapté cette solution pour les différents cas de figure.

Nous envoyons les différentes consignes de commandes sur les différents axes du drone. La consigne est utilisée dans la fonction de transfert interne au drone afin de la transformer en vitesse en m/s. Nous faisons donc bien un bouclage car nous exécutons ces instructions constamment afin de régler le drone en cas de perturbation et pour atteindre la position finale.

Le drone possède son propre repère et donc une évolution du drone sur son axe X ne correspond pas à une évolution sur l'axe X de la terre. Nous avons donc fait une série d'essai afin de déterminer les différentes influences des mouvements du drone sur le repère où les positions GPS sont transposées. (voir partie B)

Pour pouvoir donc avoir un programme valable dans toutes les circonstances nous avons donc choisi d'avoir l'axe X du drone orienté vers le Nord. Un mouvement effectué direction Nord correspond à un déplacement sur l'axe Y du repère terrestre. Nous devons donc avant de lancer le programme, orienter le drone vers le Nord pour ne pas avoir de déplacements incohérents.

Une fois ses différentes données récupérées, converties et exploitées, nous avons dû gérer les différents moteurs pour faire bouger le drone dans la bonne direction.

Asservissement en vitesse

On effectue une lecture de variable pour la vitesse de déplacement. Cette variable est utilisée ensuite dans les fonctions de déplacement. On joue sur les déplacements en X (avancer reculer) et Y (droite gauche) du drone.

En théorie, on effectue asservissement de la vitesse par une fonction affine décroissante, la vitesse en fonction de la distance.

En pratique, on se retrouve avec des paliers dues à la limite de transmission de 1 seconde du GPS.

Pour réaliser l'asservissement en vitesse, on mesure la distance initiale entre les 2 points et on effectue ensuite une différence entre cette distance et la distance restante pour avoir la distance parcourue. Le but est donc de faire diminuer la vitesse du drone en fonction de la distance qui lui reste à parcourir

On se retrouve avec une fonction de la forme F(x) = b – ax

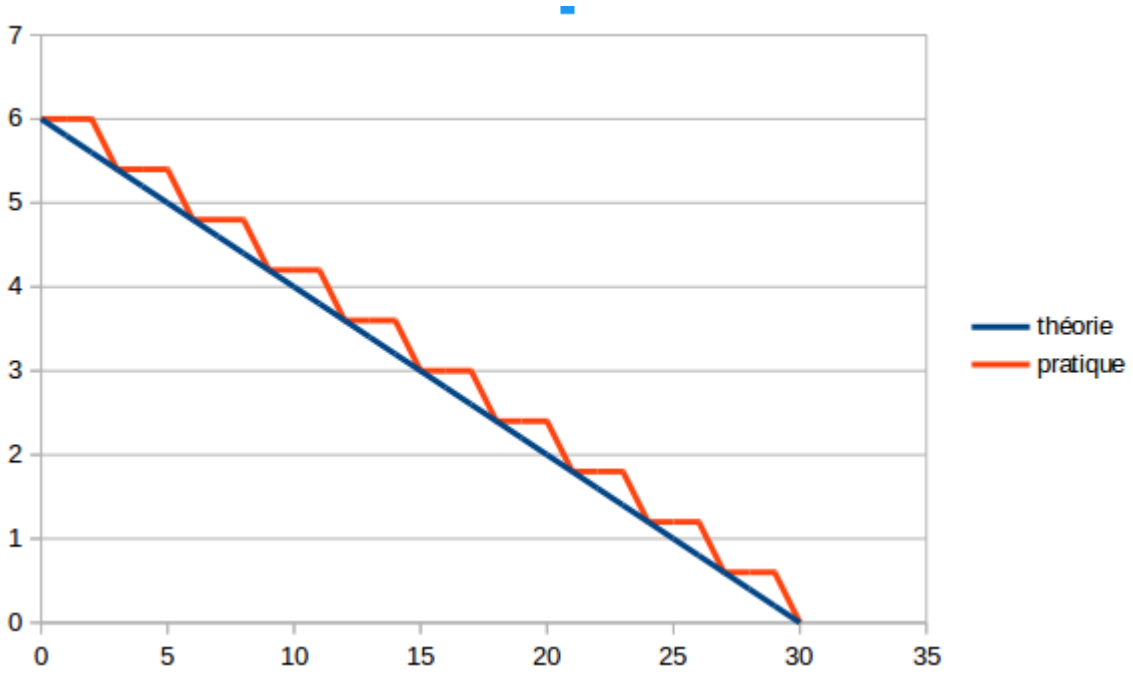

Figure 14: Représentation de la théorie et de la pratique

Comme nous l'avons précisé, nous sommes limités avec la fréquence de récupération des données GPS, toutes les secondes, donc nous ne pouvons pas réaliser un asservissement linéaire. Raison pour laquelle, nous fixons des vitesses de déplacement faibles. Néanmoins, en affichant les vitesses du drone sur le terminal,

on s'est rendu compte que le drone ralentissait bien que ce soit difficile de l'observer à l'œil nu.

Récupération des données Bluetooth

Afin d'enrichir ce projet, nous avons décidé de récupérer via le téléphone, notre position GPS. Il s'agit d'envoyer la position de l'utilisateur par Bluetooth au PC. Etant donné que l'ordinateur que nous utilisons ne possède pas de carte Bluetooth. Nous utilisons alors une clef Bluetooth. Le fichier envoyé par Bluetooth au PC est un fichier HTML. Dans ce fichier HTML, nous retrouvons la longitude et la latitude que nous extrayions à partir d'une fonction d'extraction. Cette fonction nous retournera les valeurs que nous insérons dans des variables. On envoi la position GPS par le mobile avec l'application AndLocation.

```
global xa, xb
f=open ("bluetooth_content_share.html","r")
if f.mode == "r":
        contents = f.read()
f.close()
print contents,'\n'
gps = re.findall(r'([0-9.]+[0-9.]+)',contents)
xa=gps[0]
xb=gps[1]
print xa,'\n',xb
return xa,xb
```

Figure 15: Fonction extraction donnée GPS

Le fichier doit être reçu sur l'ordinateur avant d'exécuter le programme. On effectue une recherche de caractère dans le fichier et on ne garde que les caractères à succession de chiffre et séparé ou non par une virgule. La suite de caractère est enregistrée dans une variable.

On retrouve ci-dessous dans les premières lignes le contenu du fichier HTML et les 2 dernières lignes correspondent aux valeurs que nous extrayions.

```
<html><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head><body><br> <br> <a href="http://maps.google.com/?q=50.607951,3.134831">ma
ps.google.com/?q=50.607951,3.134831</a><br> <br> Location shared with AndLocatio
n app</body></html>

50.607951

3.134831
```

Figure 16: Extraction donnée GPS du fichier HTML

Contrôle manuel du drone

Pour cette dernière fonction, nous avons proposé de pouvoir reprendre les commande sur le drone par mesure de sécurité. Nous avons remarqué qu'à certains moments le drone venait à atterrir à des endroits où il pouvait subir des dégâts matériels à la fin de son parcours dû à l'incertitude du GPS. Nous demandons alors à l'utilisateur l'autorisation d'atterrir, dans le cas négatif, il est possible alors pour l'utilisateur de contrôler le drone manuellement avec les touches du clavier pour avancer, reculer, aller à gaucher ou à droite. Avec une fonction getch(), on récupère la touche appuyée par l'utilisateur. La touche s détermine l'arrêt de la commande manuelle, elle lui donnera l'autorisation d'atterrir à ce moment.

```
def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)

    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch
```

Figure 17: Fonction acquisition d'un caractère

Conclusion

Pour conclure ce projet de fin d'études, nous sommes satisfaits du travail que nous avons accomplis tout au long de l'année. Nous avons réalisé de nombreuses recherches concernant le contrôle et l'utilisation du drone. Grâce à ces recherches, nous avons rencontré et enrichi nos connaissances sur différents modes de communication ou de contrôle comme le SDK de Parrot, Pyparrot ou encore ROS.

Nous avons d'avantage approfondi nos compétences en programmation, en automatique mais également dans la recherche et dans l'auto-critique du travail accompli. Nous avons également développé un sens de la gestion projet que nous avons aiguisé grâce à nos encadrants. Nous avons appris à se fixer des limites et des conditions d'utilisations dans le cadre du bon fonctionnement du projet.

Ce projet sera éventuellement repris par d'autres étudiants dans le cadre de projet ou TP et/ou aussi par des chercheurs du laboratoire de CRISTAL pour leurs projets en laboratoire. Cela nous a réconforté dans le travail réalisé et nous a poussé à donner le meilleur de nous-même.

Nous remercions à nouveau nos différents encadrants de nous avoir permis de réaliser ce projet.

Bibliographie

Documents de recherche fournis par CRISTAL

SDK :

- https://developer.parrot.com/

ROS :

- https://wiki.paparazziuav.org/wiki/Bebop#Bebop_2
- http://wiki.ros.org/fr/ROS/Tutorials
- https://github.com/AutonomyLab/bebop_autonomy

PyParrot :

- https://pyparrot.readthedocs.io/en/latest/
- https://github.com/amymcgovern/pyparrot