RAPPORT DE MI-PROJET :

Pilotage automatique d’un drone

AYOUB Abass
VISCOGLIOSI Alexis

IMA5
Année universitaire 2018/2019

Table des matières

Introduction

Dans le cadre de notre dernière année à Polytech Lille en Informatique Microélectronique et Automatique, nous devons réaliser un projet de fin d'études tout au long de notre année sur un sujet précis. 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. Ce drone nous a été fourni par le groupe de recherche de monsieur Rochdi MERZOUKI.

Le travail que nous devons réaliser sur ce drone est une étude de faisabilité afin de voir tout ce qui est possible de faire avec le drone 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 groupe numéro 21 composé de Justine Senellart et Claire Vandamme qui se concentreront sur la partie traitement d'image. L'intérêt premier de ce projet est de faire un bilan des capacités et de l'ouverture du drone pour que les différents encadrants puissent reprendre notre travail pour de futurs projets de recherche ou de TP.

Dans ce rapport de mi-projet, nous allons dans un premier temps faire une présentation du projet incluant un contexte, une présentation du cahier des charges et une présentation du drone. Dans un deuxième temps, nous allons détaillé le travail que nous avons effectué et les conclusions que nous en avons tiré d'un point de vue faisabilité. Enfin dans un dernier temps, nous allons explicité le travail restant à effectuer.

I) Présentation du projet et son cahier des charges

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

A) Présentation du contexte et du matériel

Comme nous l'avons précisé dans notre introduction, nous travaillons sur le pilotage automatique et le contrôle d'un drone Bebop 2 de la marque Parrot. Notre travail consiste à faire une étude de faisabilité sur les capacités du drone en termes de contrôle et d'asservissement pour ensuite donner la possibilité à d'autre utilisateurs de réutiliser notre travail pour des projets de recherche (essaim de robots) ou encore pour des Travaux Pratiques de Robotique 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

Etant donné que nous faisons une étude de faisabilité, notre cahier des charges nous laisse une certaine flexibilité :

Matériel utilisé :

Nous allons, pour ce projet, travailler avec un drone Parrot Bebop 2. La première étape du projet consiste à savoir comment nous pouvons utiliser le drone et en fonction des résultats, nous allons nous concentrer sur une méthode précise. Nous allons donc travailler sous Linux Ubuntu 16.04 et utiliser le SDK de Parrot, ROS ainsi que le langage de programmation Python pour gérer la communication avec le drone. Cette liste est une liste non exhaustive et nous envisagerons, dans la suite du projet travailler avec Matlab, Oracle et d'autres langages de programmation.

Contraintes :

Nous avons un certain nombre de contrainte pour la réalisation de ce projet. Tout d'abord nous n'avons pas accès à toutes les données du drone. Une autre des contraintes que nous avons est de travailler avec un autre binôme. Cette collaboration peut etre très utile lors de recherches d'informations mais nous contraint à nous mettre d'accord sur le mode d'utilisation du drone pour ne pas avoir d'incompatibilités quand nous tenterons de mettre les 2 sujets en commun. Ensuite nous devons nous adapter aux différents environnements du drone et apprendre les différents outils qui nous sont mis 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.

Enfin, la plus grosse de nos contraintes va être de devoir travailler en temps réel et donc de recevoir les données à une fréquence suffisante pour pouvoir les exploiter directement.

Résultats attendus :

Quant aux résultats attendus, il nous est demandé de pouvoir faire 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 envoie de commandes basiques
- Récupération des données odomériques
- Récupération des données GPS
- Exploitation des données

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 complète qui nous permet de nous connecter, de piloter, recevoir un 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 et il n'est donc pas possible de piloter directement les moteurs. Néanmoins, Parrot met à dispositions des commandes permettant de réaliser des figures ou des tâches complexes tel que des flips ou le trajet d'un point A à un point B.

De par notre sujet, 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 très vite 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 degrés

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

- 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) (in m/s)

- roll (float): Valeur du mouvement en roulade (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 grâce à la caméra avant

- Vidéo, accès en directe de la caméra si un écran est connecté

- État de la batterie : Pourcentage de batterie

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 affiché ou sauvegardé. 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 fonctionnalité 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 allons présenter une des solutions que nous avons envisagées pour récupérer les données. Cette solution est Pyparrot. Pyparrot est une solution qui se base 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 testé 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 étant géré par le SDK. Après avoir réussis 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)
    """
```

Figure 2 : Fonction extract_values

Celle fonction nous permettrait, en théorie de récupérer les données d'un capteur voulu provenant d'un paquet BLE. 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 batterie restant.

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. Néanmoins, ce mode de fonctionnement permet tout de même de contrôler le drone d'une manière plus précises qu'avec le SDK.

C) ROS

Dans cette partie concernant ROS, nous allons tout d'abord faire une présentation sur ROS puis ensuite présenter le Bebop Autonomy et les résultats que nous avons eu grâce à lui.

ROS est structuré de la manière suivante :

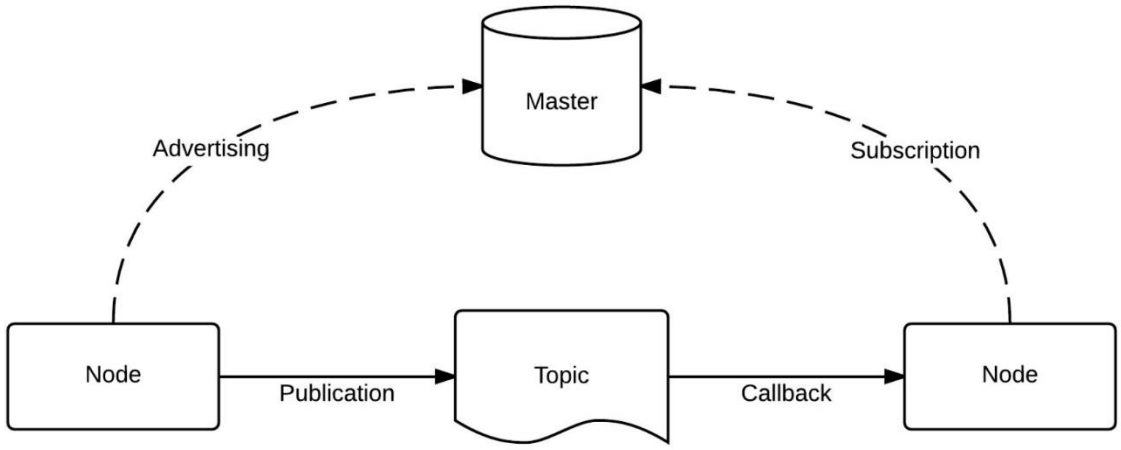

Figure 3 : structure de ROS

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 etre par exemple un capteur, un moteur ou encore un algorithme. Chaque node peut être appelé d'une façon ou d'une autre en fonction de l'action qu'il est train de réaliser. En effet, un node qui publie des données est un « publisher ».

A contrario, un node qui souscrit à des données est un « subscriber ». La communication entre chaque node 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 entre guillemet de bus d'informations. Cette communication entre 2 nœuds est gérée par un Master.

Un master est une sorte de 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 créés

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.

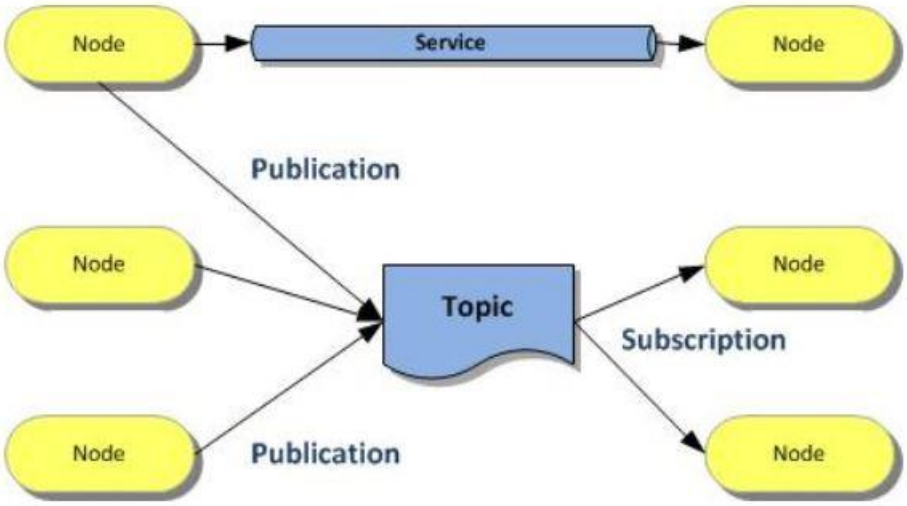

Figure 4 : Fonctionnement de ROS

Concernant l'utilisation de fichier 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 (décrit précédemment). Il possède également les librairies externes, les données…

Quant à lui, le stack est une collection de package permettant des fonctions plus complexes comme la localisation… L'un des intérêts de ces fichiers c'est que ce sont tous 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 plus étudié concernant cet outil. Nous nous sommes plus particulièrement intéressés à cet outil pour plusieurs raisons. La première est ROS est utilisé par une très grande communité de roboticien et d'automaticien et, de ce fait, il possède un très grand nombre de fonctions open source. Il est dorénavant devenu 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 depuis le début de ce projet est dans un premier temps d'être capable de récupérer des données odométriques, GPS ou autres en temps réel pour ensuite pouvoir les exploiter. Cette tâche n'a pas pu être réaliser avec Pyparrot et avec le SDK.

Pour ROS, nous nous sommes aidés de Bebop Autonomy. Bebop Autonomy est un gros fichier, utilisant ROS, et nous permettant d'exécuter des commandes de bases comme décoller, atterrir, faire des flips … Les commandes pour exécuter ces fonctions sont du type :

rostopic pub --once /bebop/land std_msgs/Empty (atterissage)

Dans cette fonction atterrissage, nous publions sur le topic bebop/land et nous envoyons un message vide du type std_msgs.

À la suite de cela, nous avons voulu avoir plus d'informations sur les différents topics et plus précisément les topics contenants les données odométriques. Nous avons donc utilisé la commande Rostopic info sur le topic bebop/odom

Nous avons eu les retours suivants :

```
abass@abass-HP-Pavilion-dv7-Notebook-PC:~/catkin_ws/src/projetbebop/src$ rostopic info /bebop/odom
Type: nav_msgs/Odometry

Publishers:
 * /bebop/bebop_driver (http://abass-HP-Pavilion-dv7-Notebook-PC:34379/)

Subscribers: None
```

Figure 5 : Rostopic Info

Comme on peut constater, ce topic reçois un message Odometry du type nav_msgs. Nous avons donc fait un "rosmsg echo" de Odometry afin de voir les informations qui transitent dans ce message.

```
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 : Rosmsg show

On peut remarquer que nous pouvons recevoir, grâce à ce topic, les données x, y et z en position, en vectoriel, en angulaire et d'autres données précises. Nous avons donc créé un programme python nous permettant de souscrire sur ce topic, et d'afficher les données désirées. Nous obtenons les résultats suivants :

```
abass@abass-HP-Pavilion-dv7-Notebook-PC:~/catkin_ws/src/projetbebop/src$ rosrun projetbebop donnee.py
[INFO] [1543935181.182111]: la valeur de x est : 0.000000
[INFO] [1543935181.449038]: la valeur de x est : 0.000000
[INFO] [1543935181.582382]: la valeur de x est : 0.007430
[INFO] [1543935181.848720]: la valeur de x est : 0.029334
[INFO] [1543935181.982194]: la valeur de x est : 0.039674
[INFO] [1543935182.182334]: la valeur de x est : 0.056542
[INFO] [1543935182.449170]: la valeur de x est : 0.072674
[INFO] [1543935182.582293]: la valeur de x est : 0.077984
[INFO] [1543935182.848975]: la valeur de x est : 0.082742
[INFO] [1543935182.982327]: la valeur de x est : 0.083884
```

Figure 7 : Données reçues

Nous pouvions donc commencer l'asservissement en position grâce à ces données en temps réel. Mais lors d'un essai, nous avons perturbé le drone afin de voir l'évolution de ces données et nous nous sommes donc rendu compte que le drone était naturellement déjà asservi (pour des mesures de sécurité et de qualité) et quil nous était impossible de modifier cet asservissement.

Après cette découverte nous nous sommes donc concentrés sur le déplacement du drone d'un point GPS à un autre. De la même manière que précédemment, nous avons donc pu récupérer, en temps réel les données GPS du drone. Nous avons donc pu recentrer notre sujet de projet et commencer à travailler sur le déplacement du drone entre 2 points GPS précis. (En cours)

Il est à noter que ces commandes entrées sur le terminal pour récupérer des données ou envoyer des actions peuvent être insérées dans un fichier « .py ».

```
---
header:
  seq: 371
  stamp:
    secs: 1544710227
    nsecs: 559619490
  frame_id: "base_link"
latitude: 50.6076988879
longitude: 3.13721804196
altitude: 45.2000007629
---
```

Figure 8 : Récupération donnée GPS du drone

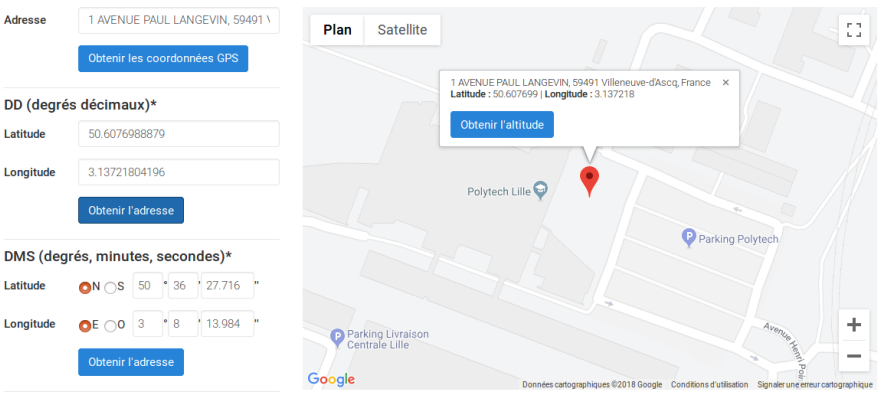

Figure 9 : Localisation sur GoogleMap des données GPS récupérées

III) Travail restant à effectuer

Dans cette dernière partie du rapport nous allons développer le travail restant à effectuer jusqu'à la date finale du rendu du projet.

Comme précisé dans la partie 2, nous allons donc utiliser ROS, Python et des commandes de Bebop Autonomy pour la suite du projet. Nous nous sommes donc rendu compte que le drone possède, de base, un asservissement interne et que nous n'avons pas accès à ces données. Nous avons donc décidé de poursuivre l'étude de faisabilité sur d'autres aspect puisque l'asservissement en position est déjà fait.

Comme nous l'avons présenté, nous nous sommes ensuite concentrés sur le GPS pour pouvoir réaliser divers mouvements avec le drone. Nous avons reussi, grâce à des lignes de commandes à nous abonner au topic du GPS pour pouvoir récupérer en temps réel les diverses valeurs de longitude, de latitude et d'altitude.

L'étape suivante serait de réaliser soit un programme python ou avec des lignes de commande ROS le déplacement du drone sur des points précis dans l'espace en lui imposant une latitude, une longitude et une altitude. Cette étape sera la plus compliqué et la plus complète du projet car nous devrons probablement faire des contrôles sur les axes odométriques x, y et z dans l'espace en même temps pour pouvoir gérer la position du drone.

L'intérêt de ce travail serait ensuite de faire un asservissement en vitesse soit entre ces 2 points dans l'espace ou sur un parcours précis.

Et enfin, en fonction de l'avancement du second binôme sur le traitement d'image, nous pourrions assembler nos 2 projets pour faire un évitement d'obstacles lors d'un parcours précis.

Détails du projet	
Projet	PFE
Chef de projet	Alexis Viscogliosi
Jours ouvrés uniquement	OUI
Date de début	17/12/2018
Commentaires	

	Tâches	Date de début	Durée	Date de fin
1	Tâche 1 : GPS	01/01/2019	17	23/01/2019
1.1	Récupération d'un point B	01/01/2019	10	14/01/2019
1.2	Déplacement d'un point A vers B	15/01/2019	7	23/01/2019
2	Tache 2 : Contrôle en vitesse	24/01/2019	12	08/02/2019
3	Tache 3 : Evitement d'obstacle	09/02/2019	14	28/02/2019

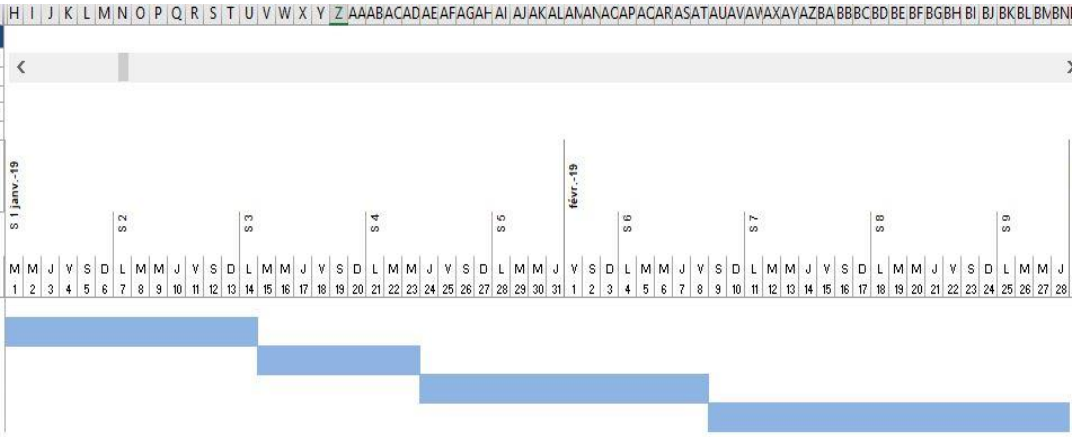

Figure 10 : Planning prévisionnel

Conclusion

Pour conclure sur la mi-parcours de ce projet de fin d'études, nous pouvons dire que pour le moment, nous sommes satisfaits du travail que nous avons accomplis durant ces 4 premiers mois. Nous avons pu réaliser de nombreuses recherches concernant le contrôle et l'utilisation du drone. Grâce à ces recherches, nous avons pu, sur notre WIKI réaliser des présentations complètes des différents modes d'utilisation du drone comme par exemple le SDK de Parrot, Pyparrot ou encore ROS

De ce fait, nous avons pu approfondir nos compétences dans différents langages de programmatino comme Python et apprendre à utiliser ROS.

Bien que notre projet de base soit très orienté Automatique, nous nous sommes rendu compte qu'il est plus orienté Informatique Industrielle. Cela va nous permettre de développer de nombreuses compétences en informatique ainsi qu'en gestion de projet. Bien entendu, les drones étant des robots comme les autres, nous devrons, dans la suite du projet, utiliser toutes nos connaissances en automatique pour pouvoir mener à bien ce projet.

Bibliographie

SDK :

- https://developer.parrot.com/

Pyparrot :

- https://pyparrot.readthedocs.io/en/latest/bebopcommands.html#bebop-commands
- https://github.com/amymcgovern/pyparrot

ROS :

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