Étudiant :
PROFIT Étienne
HART Tristan

Encadrants :
REDON Xavier
VANTROYS Thomas
BOÉ Alexandre

RAPPORT DE PROJET IMA4 :

P31 : Accueil personnalisé par drone.

Université de Lille 1 SCIENCES ET TECHNOLOGIES
POLYTECH LILLE

Année Universitaire
2016-2017
IMA 4

SOMMAIRE

INTRODUCTION

Nous nous appelons Tristan Hart et Étienne Profit, nous sommes actuellement en 2ème année de cycle ingénieur à l'école Polytech Lille. Dans le cadre de notre cursus, nous devons travailler sur un projet de notre choix et notre choix s'est porté sur le projet P31 : Accueil personnalisé par drone.

Nous avons décidé de nous pencher sur ce sujet pour plusieurs raisons, tout d'abord le fait de travailler sur un drone est intéressant car ce domaine est en expansion et de plus en plus d'entreprise s'orientent vers l'automatisation d'un drone autonome. Ce projet était donc une bonne occasion d'enrichir notre expérience sur ce genre de sujet. De plus, ce projet allie à la fois une partie logicielle et une partie matérielle, ce qui allait demander d'appliquer un plus large spectre de connaissances vues en cours.

Le but de ce rapport est donc de décrire et de présenter l'ensemble du travail effectué lors de ce module, étape par étape. Nous commencerons donc par une première partie, décrivant le cahier des charges de notre projet. La deuxième partie traitera des premières solutions apportées à notre problème. Dans la troisième partie, nous parlerons de la partie matérielle et de la réalisation de la partie logicielle. Nous parlerons ensuite dans la quatrième partie des problèmes rencontrés et les éventuelles solutions apportées. Enfin, en guise de conclusion, nous ferons le bilan de ce projet et nous l'ouvrirons vers d'autres applications possibles.

I) DESCRIPTION DU CAHIER DES CHARGES

L'enceinte de Polytech Lille est bien connue de ses occupants, élèves ou professeurs. Cependant, pour un nouvel arrivant, qu'il soit simple visiteur ou conférencier, se rendre dans la salle d'Arsonval ou même trouver une place de parking peut se révéler être une épreuve. L'intérêt de notre projet est de faciliter à ces derniers l'accès à tous les recoins des bâtiments de Polytech Lille grâce à l'utilisation d'un drone d'accueil personnalisé.

L'objectif est donc, à terme, de pouvoir guider un utilisateur de l'entrée du parking jusque à une salle à l'intérieur de l'école. Cela passe par la réalisation de 2 objectifs intermédiaires :

- Accueillir un visiteur depuis l'entrée du parking et le mener jusque à une place de parking libre.

- Guider ce même visiteur depuis sa place de parking jusque à une salle.

Nous disposons d'un drone drone Phantom3 d'1,2 Kg capable de supporter une charge utile d'un Kg. Nous devons utiliser cette charge utile à bon escient afin de ne pas perdre en autonomie. Nous envisageons donc d'utiliser 500g de matériel à monter sur le drone afin de remplir les fonctions demandées suivantes :

- L'utilisateur devra pouvoir choisir sa destination depuis une interface portable soit par Bluetooth avec un téléphone mobile sur lequel sera installé une application soit par le biais d'une page internet hébergée par une Raspberry.

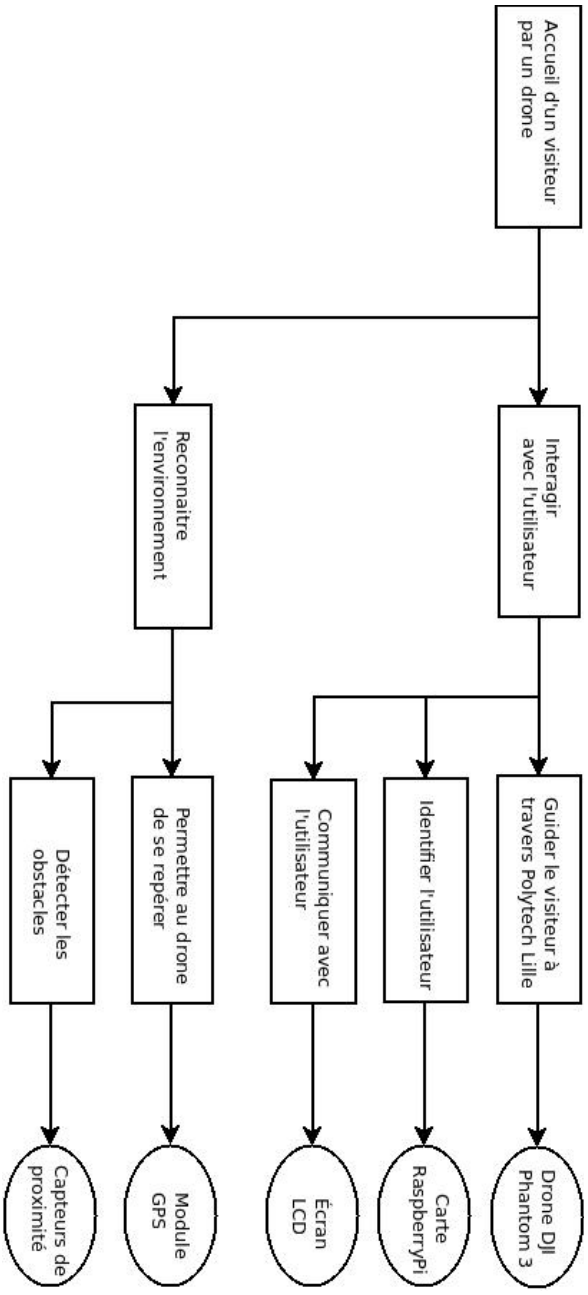
- Le drone se déplacera à une hauteur conséquente et sera capable d'éviter et de contourner les obstacles présents sur son chemin.

- Le drone devra pouvoir se repérer automatiquement à l'extérieur.

II) PREMIÈRES SOLUTIONS APPORTÉES

a. Choix techniques proposés

Nous utiliserons probablement la plateforme ROS (Robot Operating System), qui nous permet des algorithmes de SLAM (Simultaneous Localization And Mapping). Nous aurons aussi recours à des bases de données afin de recenser les utilisateurs, les salles… nécessitant alors une machine capable de les gérer via un logiciel de type MySQL.



Afin d'utiliser au mieux notre matériel, et afin d'optimiser le temps qu'il nous est alloué, nous définissons une liste de tâche à réaliser. Cette liste se divise en 3 grandes parties, la partie préliminaire où différentes études seront réalisée afin de mieux comprendre notre projet, la première partie qui consiste à réaliser les différentes fonctions de base de notre projet pour l'utilisation en extérieur et enfin la troisième partie dans laquelle nous travaillerons sur la partie en intérieur.

b. Liste des tâches à effectuer

Préliminaires :

* Étude du mode de communication entre le drone et l'application qui lui est dédiée.
* Étude de ROS et des algorithme de SLAM pour voir si ces composants logiciels peuvent aider au développement du projet.
* Installation de ROS sur RaspberryPi

Partie 1 :

* Installation du SDK de DJI.
* Essais des capteurs de proximités qui seront couplés avec la Raspberry.
* Gestion d'un itinéraire par le drone, plus précisément la gestion d'un vol automatique avec le phantom3.
* Communication drone/capteur par le biais de la carte Raspberry.
* Identification d'une personne grâce à une connexion distante.
* Mener uen voiture à sa place de parking.

Partie 3 :

* Se repérer dans Polytech grâce à un algorithme de SLAM intégré à la Raspberry.
* Se rendre dans une salle donnée en partant de l'entrée de Polytech (coté parking).

Le calendrier prévisionnel de ce projet a été réalisé grâce à un diagramme de Gantt et est disponible en Annexe [1].

III) RÉALISATIONS

Dans cette partie nous détaillerons le travail réalisé lors de nos séances de projet. Cette partie se divise en 2 sous-parties qui traiteront respectivement des réalisations matérielles et des réalisations logicielles.

a. Partie matérielle

La partie matérielle de notre projet tournait principalement autour de la création d'un système capable de détecter les obstacles autour du drone. Nous décidons donc de commander des capteurs de proximité. Nous avions le choix entre 2 familles de capteurs : les capteurs infrarouge et les capteurs à ultrasons.

Notre choix s'est porté vers les capteurs infrarouge car leur plage de détection est largement supérieur au ultrasons. Nous avons donc choisi le capteur sharp GP2Y0A02YK. Ce choix s'est fait en en prenant en compte 2 critères : le prix et la plage de détection et ce capteurs possède le meilleur rapport détection/ prix. En effet celui celui-ci détecte les obstacle jusque à 150cm ce qui est suffisant pour notre contexte.

Figure 2 : capteur Sharp GP2Y0A02YK

Nous commandons donc 3 capteurs de cette sorte afin de couvrir une large zone à l'avant du capteur comme sur le schéma ci-dessous.

Figure 3 : Cône de détection voulu du drone avec les capteurs.

Nous commencons par relier notre Raspberry au réseau afin de pouvoir installer des paquets dessus. Pour cela, nous configurons le fichier < /etc/dhcpcd.conf> en rajoutant les lignes :

```
interface eth0

static
ip_adress=172.26.78.118/20
routers=172.26.79.254
static
domain_name_servers=193.48.57.34
```

Ayant une RaspberryPi3 à notre disposition, nous décidons de coupler cette dernière à notre jeu de capteurs. Pour cela nous nous inspirons du schéma de montage suivant :

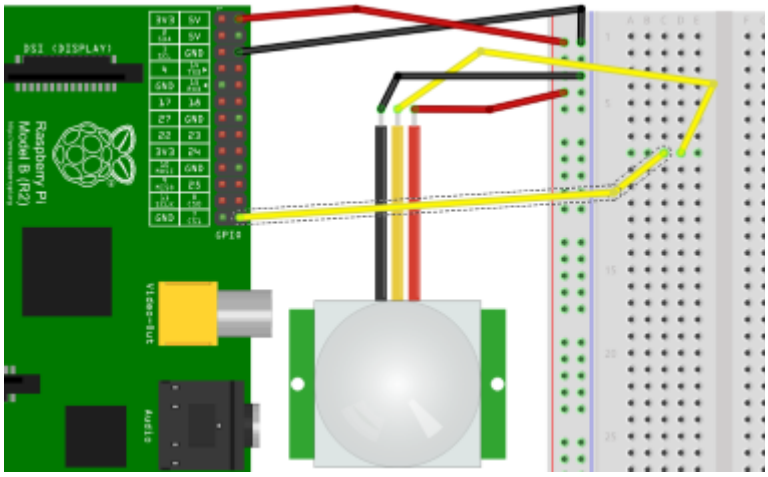
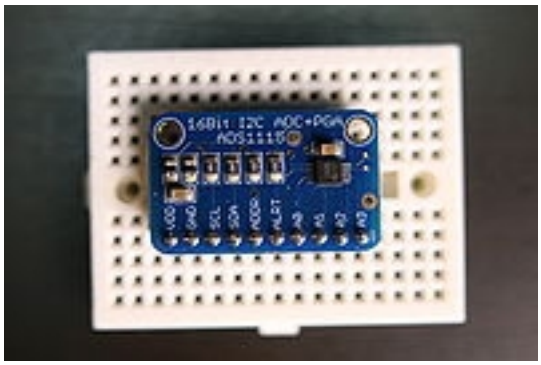


Figure 4 : Schéma de montage d'un capteur avec la Raspberry.

Mais un problème apparaît rapidement. La valeur de la broche étant sensée recevoir la donnée est toujours à 1 (valeur récupérée grâce à l'utilitaire 'wiringPi' installé sur la Raspberry) même lorsque la distance à l'obstacle varie. Nous formulons donc l'hypothèse que la Raspberry contient un système de seuil qui fait qu'au delà d'une certaine valeur, le signal détecté retourne la valeur logique 1. Pour remédier à cela, nous intégrons à notre système un convertisseur analogique digital.



Ce convertisseur sur 12 bits va nous permettre de convertir la valeur analogique de notre capteur pour l'envoyer à la Raspberry. De plus ce convertisseur possède 4 entrées analogique, ce qui est plus que suffisant pour accueillir nos 3 capteurs de proximité. Il communique avec la Raspberry via le protocole I2C.

Figure 5 : ADC.

Maintenant que nous sommes en mesure de relever la valeur de notre capteur, il ne nous reste plus qu'à trouver comment la récupérer et l'afficher sur le terminal. Nous installons donc des librairies dont python sur notre carte et nous entamons la création d'un programme qui lira la valeur retournée par notre capteur (branché sur A0 de l'ADC).

Nous nous documentons donc sur internet et trouvons rapidement quelques exemples de code en python. Nous décidons d'abord d'afficher en colonnes les valeurs relevées totues les 0,5 secondes afin d'éviter la surcharge d'un éventuel buffer ce qui ferait perdre en réactivité pour notre système.

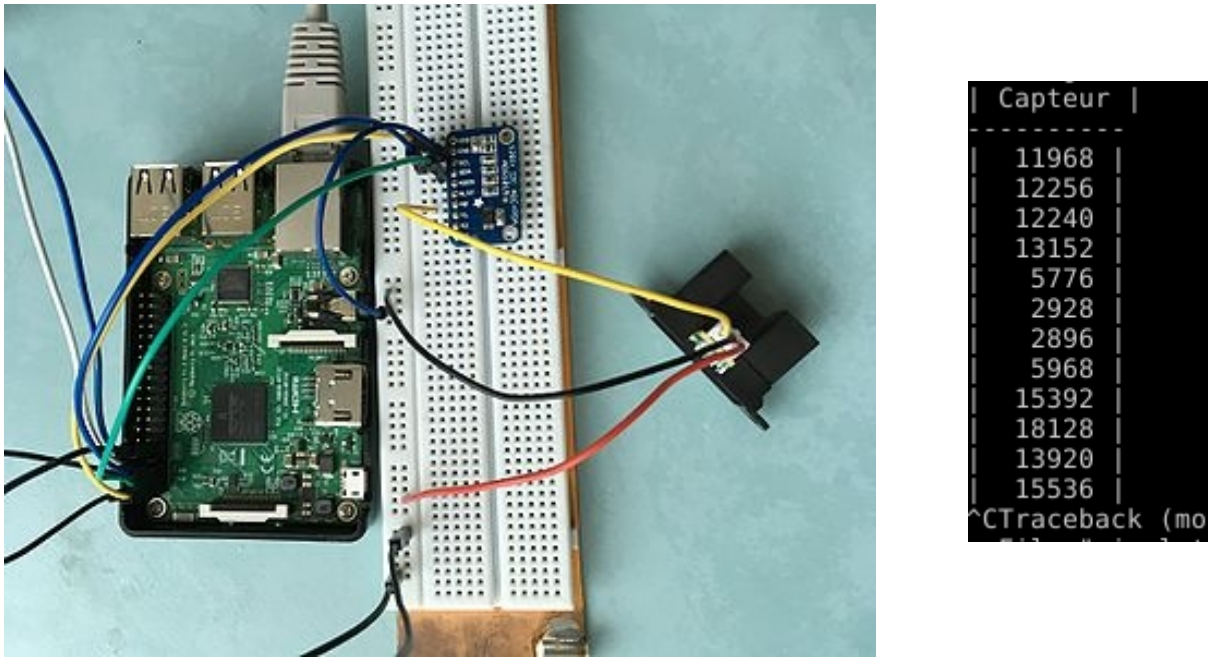


Figure 6 : Photographie du montage (gauche) et résultat sur le terminal (droite).

Maintenant que notre programme fonctionne pour un capteur, nous rajoutons les 2 autres capteurs sur A1 et A2 de l'ADC.

Afin de pouvoir détecter les obstacles, nous créons un support pour nos capteurs qui pourra également accueillir l'ADC ainsi que la Raspberry. Ce support devant être le plus léger possible, nous utilisons du balza, un bois léger utilisé pour le modélisme et qui est assez résistant pour supporter notre montage.

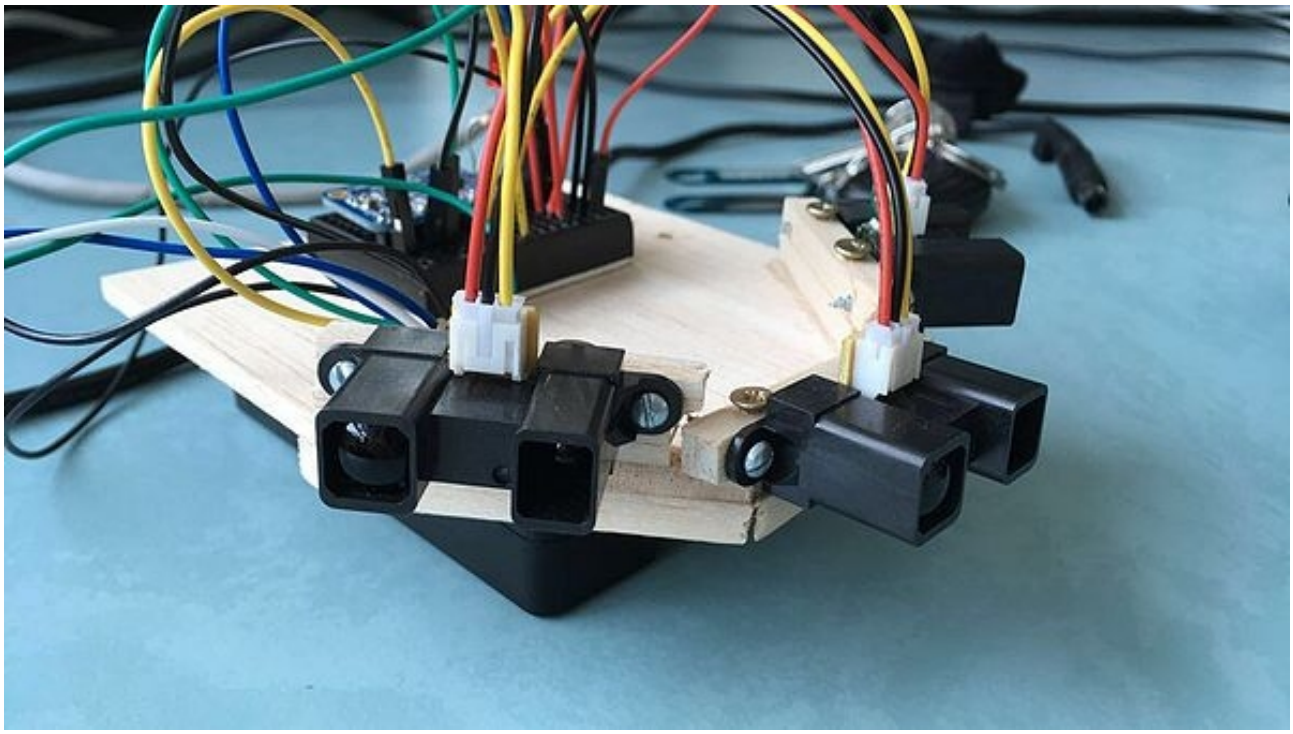


Figure 7 : Nacelle finale de notre projet avec la Raspberry, l'ADC et les 3 capteurs.

Pour ajouter nos capteurs et pouvoir gérer l'allumage d'une LED s'allumant lorsque un obstacle est trop proche, nous importons la librairie RPi.GPIO qui permet de gérer les I/O de la Raspberry. A partir de là le programme est assez simple :

```
# impression des colonnes de valeurs
print('| capteur Droit | Capteur Avant | Capteur Gauche |'.format(*range(4)))
print('-' * 50)

# Main
while True:
    # Read all the ADC channel values in a list.
    values = [0]*4
    for i in range(4):
        # Lecture des valeurs du ADC
        values[i] = adc.read_adc(i, gain=GAIN)

    # Affichage des valeurs recuperee
    print('|    {0:>6}     |    {1:>6}     |    {2:>6}     |'.format(*values))
```

Figure 8 : Morceau de programme montrant la récupération des valeurs.

L'ensemble du programme est consultable en Annexe [2].

Nous utilisons une boucle for afin de stocker la valeur de chaque capteur dans un tableau. Puis nous les affichons une par une ; C'est ces valeurs que nous testerons afin de voir si un obstacle est trop proche ou non grâce à une simple instruction if.

Le résultat retourné sur le terminal est le suivant :

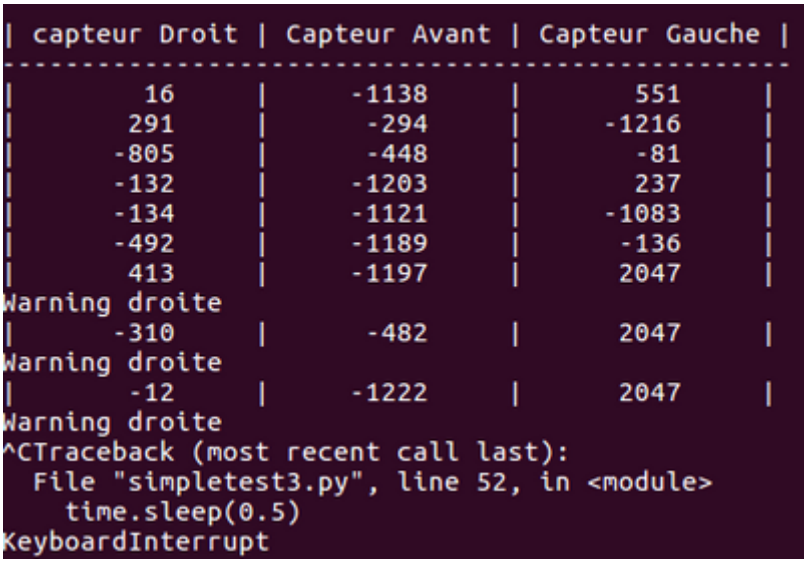


Figure 9 : Résultat final affiché sur le terminal.

En conclusion de la partie matérielle, nous disposons d'une nacelle de 300 grammes qui pourra être embarquée par le drone. Celle-ci détecte un obstacle jusque à 1 mètre apparaissant dans un angle de 130° à l'avant du drone. Néanmoins l'utilisation d'une batterie externe est nécessaire est nécessaire afin d'éviter une trop grosse perte d'autonomie du drone si cette dernière était alimentée par le drone (via un câble double micro-usb). Nous restons donc dans la gamme des 500g de notre cahier des charges.

b. Partie logicielle

En terme de pilotage, une simple télécommande fournie par DJI pour le Phantom 3 suffit. Cependant, dans le cadre de notre projet, nous avons pour objectif de réaliser un système de pilotage automatique du drone.

Nous avons envisagé différentes solutions pour arriver à cette fin :

- Réussir à récupérer directement via un port du drone ou de la télécommande les informations de vol pour pouvoir les analyser en temps réel puis les utiliser afin contrôler le drone.
- Recréer le circuit de la télécommande afin de lui ajouter des modifications permettant de répondre à notre projet.
- Utiliser une application du type de celle fournie par DJI (Application officielle DJI GO - For Phantom 3).

Les deux premières solutions ont toutes les deux été retirées de nos plans. En effet, tandis que la seconde aurait été difficilement réalisable étant donné l'absence totale de plans du circuit de la télécommande que nous aurions pu réutiliser, l'absence pure et simple de moyen nous permettant de récupérer les informations de vol en cours de vol nous a forcé à écarter la première.

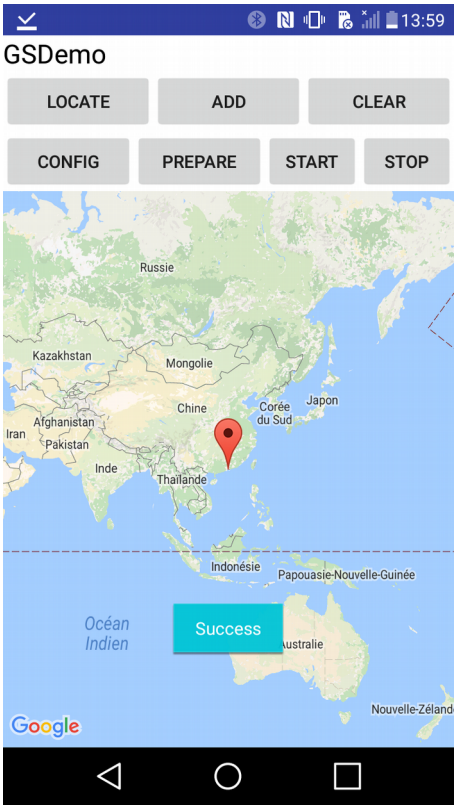
Ne restait donc que la troisième solution.

La société DJI met à disposition des possesseurs de drones un SDK, complété par une dizaine de codes exemple en open source que les apprentis-développeurs peuvent utiliser pour apprendre à créer leur propre application dédiée au drone.

Dans le cadre de notre projet nous utilisons le code source disponible à l'adresse suivante :

https://developer.dji.com/mobile-sdk/documentation/android-tutorials/GSDemo-Google-Map.html

Le fonctionnement de ce code est le suivant :



Lors du lancement de l'application, une carte du monde s'ouvre avec un marqueur affiché sous la position du drone. Plusieurs bouton permettent plusieurs actions.

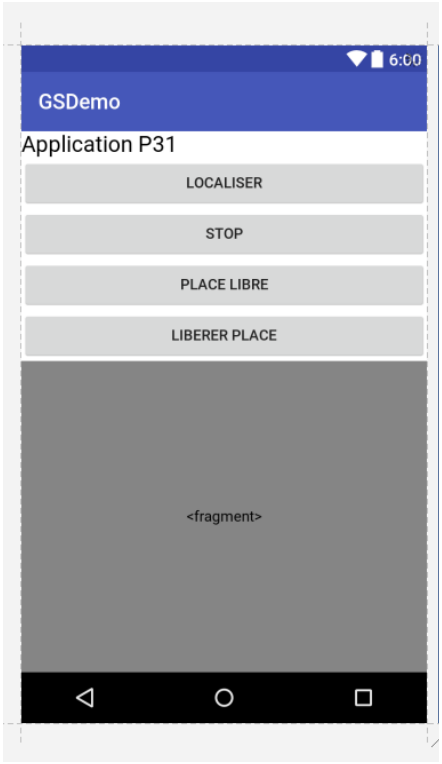
Un bouton "Add" permet d'activer la fonction d'ajout de Waypoints lors d'un appui sur l'écran. Un waypoint est simplement un point par lequel le drone passera forcement lors du son vol. On peut ajouter plusieurs waypoints pour définir un trajet précis du drone. Arrivé au dernier waypoint, le drone retourne à sa position initiale puis se pose. Un bouton "clear" permet d'effacer tous les waypoints déjà ajoutés. Un bouton "Config" permet d'ouvrir une interface de configuration des paramètres de vol : vitesse, altitude, type de guidage (waypoint, circle, etc). Le bouton "Prepare" enregistre tous les waypoints dans le plan de vol. Enfin, le bouton "Start" fais décoller le drone si rien ne s'y oppose.

L'application utilise évidemment l'API Google Maps. Cette interface de programmation, créé par Google contient un ensemble de fonctions, classes et méthodes pour utiliser le système de cartes de Google.

Le code de l'application contient une vingtaine de méthodes diverses et variées :

- Une fonction permettant d'ajouter un marqueur sur la carte du monde à l'endroit où l'utilisateur clique,
- Une fonction permettant d'ouvrir un sous-menu pour fixer les caractéristiques du vol,
- Une fonction permettant de valider les waypoints (WP) indiqués par l'utilisateur et de préparer le plan de vol,
- Et plus d'une dizaine d'autres fonctions que nous ne détaillerons pas ici car leur utilité dans le cadre de ce projet est moindre.

En premier lieu, afin de donner corps à notre application, il convient de créer l'interface principale, celle que l'utilisateur rencontrera lors de l'utilisation. Comme on peut le constater, plusieurs boutons ont été retirés pour laisser leur place à d'autres. Dorénavant, le bouton "Place libre" réalise la totalité du travail, de l'application des waypoints au démarrage du drone. Le bouton "Stop" a été laissé pour un soucis de sécurité lors des essais. Enfin, le bouton "Libérer Place" demande à l'utilisateur le numéro de la place qu'il occupait précédemment afin d'actualiser son statut (détails plus bas).




```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#FFFFFF"
    tools:context="com.dji.GSDemo.GoogleMap.MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <TextView
            android:id="@+id/ConnectStatusTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Application P31"
            android:gravity="center"
            android:textColor="#000000"
            android:textSize="21sp"
            />
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:id="@+id/locate"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Localiser"
            android:layout_weight="1"/>
    </LinearLayout>
```

Détails à noter, nous avons choisi d'afficher la carte en vue satellitaire pour faciliter l'utilisation.

Le code en lui même de l'application se résume en une seule grosse fonction :

```
private void PLACECODE(){ /// COEUR DU PROJET
    if (isAdd==false){
        runOnUiThread(() -> { gMap.clear(); });
        if (mWaypointMission != null){
            mWaypointMission.removeAllWaypoints(); // Remove all the waypoints added to the task
        }
        isAdd = true;
        add.setText("Calcul du trajet");
        int i=0;
        while (places_libres_disp[i]!=1){
            i++;                                        // Recherche d'une place libre
        }
        places_libres_disp[i]=0;                        // Actualisation du statut de la place
        int j=0;
        for (j=0; j<places_libres[i].length;j++){
            LatLng PosFinale = new LatLng(places_libres[i][j][0],places_libres[i][j][1]);
            MarkerOptions markerOptions = new MarkerOptions();                          // Mise en place d'un marker pour
            markerOptions.position(PosFinale);                                          // chaque point du trajet
            markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_BLUE));
            Marker marker = gMap.addMarker(markerOptions);
            mMarkers.put(mMarkers.size(), marker);
            DJIWaypoint mWaypoint = new DJIWaypoint(places_libres[i][j][0], places_libres[i][j][1], altitude);
            if (mWaypointMission != null) {                                             // Ajout de chaque point du trajet
                mWaypointMission.addWaypoint(mWaypoint);                                // au plan de vol
            }
        }

        isAdd = false;
        add.setText("Place libre");                                                 // Affichage du numéro de la place
        setResultToToast("Numéro de place :" + i);
        mHeadingMode = DJIWaypointMission.DJIWaypointMissionHeadingMode.UsingWaypointHeading;
        mFinishedAction = DJIWaypointMission.DJIWaypointMissionFinishedAction.GoFirstWaypoint;
        if (mWaypointMission != null){
            mWaypointMission.finishedAction = mFinishedAction;                      // Configuration
            mWaypointMission.headingMode = mHeadingMode;
            mWaypointMission.autoFlightSpeed = mSpeed;
```

```
    if (mWaypointMission.waypointsList.size() > 0){
        for (i=0; i< mWaypointMission.waypointsList.size(); i++){
            mWaypointMission.getWaypointAtIndex(i).altitude = altitude;
        }

        setResultToToast("Set Waypoint attitude successfully");

    }
}
if (mMissionManager != null && mWaypointMission != null) {                      // Préparation du plan de vol

    DJIMission.DJIMissionProgressHandler progressHandler = new DJIMission.DJIMissionProgressHandler() {
        @Override
        public void onProgress(DJIMission.DJIProgressType type, float progress) {
        }
    };

    mMissionManager.prepareMission(mWaypointMission, progressHandler, new DJICommonCallbacks.DJICompletionCallback() {
        @Override
        public void onResult(DJIError error) {
            setResultToToast(error == null ? "Plan de vol prêt" : error.getDescription());
        }
    });
    mMissionManager.startMissionExecution(new DJICommonCallbacks.DJICompletionCallback() {
        @Override
        public void onResult(DJIError error) {
            setResultToToast("Mission Start: " + (error == null ? "Départ du drone" : error.getDescription()));
        }
    });
}
```

Cette fonction se sépare en plusieurs parties :

1- Les waypoints précédents sont effacés de l'écran. Il est important de noter que seule la partie graphique du waypoint disparaît, les coordonnées des points utilisés pour définir le précédent plan de vol sont toujours utilisables tant qu'elles n'ont pas été "libérées" par le départ d'un utilisateur.

2- Au début du programme, on a déclaré plusieurs tableaux :

```
double[] places_libres_disp = {0,1,1,1,1,1,1};
double[] places_handicap_disp = {1};
double[][][] places_libres = {{{50.607168,3.138409},{50.607193,3.138360},{50.607643,3.138624}},{{50.607168,3.138406},{50.607260,3.138045},{50.
double[][][] places_handicap = {{{50.607178,3.138409},{50.607193,3.138360},{50.607454,3.138902},{50.607676,3.137604},{50.607712,3.137621}}};
private GoogleMap gMap;
```

Le premier tableau, places_libres correspond à une liste de chemins, chaque chemin étant simplement une suite de coordonnées, représentées elle même par un tableau de 2 flottants. Le tableau places_handicap est défini de la même manière.

Le second tableau, places_libres_disp à le même nombre d'éléments que places_libres. Chacun de ses éléments correspond à la variable de disponibilité de chaque place de même indice dans places_libres (1 pour libre, 0 pour occupée).

Lorsqu'un utilisateur va indiquer par un appui sur le bouton "place libre" qu'il souhaite une place de parking, le programme va automatiquement aller chercher une place libre dans le tableau.

Dans le cadre de ce projet, nous avons uniquement sélectionné un échantillon d'une dizaine de places du parking de Polytech Lille, afin d'avoir des chemins variés.

3- Une fois l'indice de la place récupéré, on crée pour chaque point du chemin indiqué dans le tableau un set d'options de marker. on attribue à ce set d'options les coordonnées du point. On crée ensuite un marker basé sur ce set d'options puis on l'affiche sur la carte à l'aide de **gMap**.addMarker(markerOptions). Ensuite on crée le DJIWaypoint, un type d'objet qui peut définir un plan de vol du drone avec les coordonnées du point en question. Enfin, on ajoute le nouveau point au plan de vol à l'aide de la commande mWaypointMission.addWaypoint.

4- Vient ensuite la mise en place des caractéristiques du vol. Pour faciliter le suivi du drone par le conducteur, on place la vitesse de vol sur "Low" et à une altitude relativement basse. De plus le drone est programmé pour revenir à sa position initiale une fois le dernier waypoint atteint.

5- Enfin, la dernière partie du code permet simplement de communiquer au drone les instructions de vol, puis de le faire décoller.

```
private void showRetourplaceDialog(){
    LinearLayout wayPointSettings = (LinearLayout)getLayoutInflater().inflate(R.layout.dialog_libererplace, null);

    final TextView wpplaceliberee_TV = (TextView) wayPointSettings.findViewById(R.id.placeliberee);

    new AlertDialog.Builder(this)
            .setTitle("")
            .setView(wayPointSettings)
            .setPositiveButton("Valider",(dialog, id) → {

                    String altitudeString = wpplaceliberee_TV.getText().toString();
                    placealiberer = Integer.parseInt(nulltoIntegerDefalt(altitudeString));
                    Log.e(TAG,"placealiberer "+placealiberer);
                    if (places_libres_disp[placealiberer]==1){
                        setResultToToast("Erreur");
                    }
                    else{
                        places_libres_disp[placealiberer]=1;
                    }
            })
            .setNegativeButton("Annuler", (dialog, id) → {
                    dialog.cancel();
            })
            .create()
            .show();
}
```

Enfin, la dernière fonction présentée ci dessus permet simplement d'ouvrir la fenêtre déclarée dans dialog_libererplace.xml.

IV) PROBLÈMES RENCONTRÉS

Comme lors de tout projet, notre groupe s'est confronté à de nombreux problèmes. Cela a donné lieu à des blocages, des impossibilités et nous a parfois demandé d'abandonner certains points du cahier des charges. Dans cette partie , nous parlerons des plus gros problèmes que nous avons rencontré et comment nous les avons résolu, dans le cas où ils étaient solvable.

Le premier problème que nous avons rencontré fût de ne pas pouvoir nous connecter à notre Raspberry. En effet, nous nous servions d'une RaspberryPi3, sur laquelle la liaison série n'était pas activée de base. Il a fallu aller dans le boot de la Raspberry, et modifier le fichier config.txt en y ajoutant les lignes :

enable_uart = 1
core_freq = 250

Ainsi, nous pouvions lancer dès le démarrage de la Raspberry la liaison série qui n'est plus lancée par défaut sur les Raspberry Pi 3. Néanmoins, au bout de la première connexion, ce fichier config.txt s'est complètement effacé sans que nous nous en rendions compte ce qui nous a retardé dans notre projet.

Autre problème auquel nous nous sommes heurtés, l'impossibilité de faire démarrer les moteurs du drone. Lorsque nous lancions notre drone et que nous le connections avec l'API fournie par DJI, cette dernière nous empêchait de décoller ou même de démarrer les moteurs tant que le calibrage de l'IMU n'était pas fait. Nous sommes donc allé dans les jardins et avons tenté de calibrer ce dernier par une procédure fournie. Cependant la calibration n'était pas prise en compte et nous gardions le même message d'erreur « Error calibrate IMU ». Nous avons donc mis à jour le software de la télécommande ainsi que celui du drone mais rien n'as changé. Le problème s'est avéré venir de l'API de DJI qui contenait certainement un bug dans sa version pour Iphone. Nous avons donc calibré le drone en prenant le téléphone d'un camarade sous Android, les téléphones fournis par l'école n'étant pas assez récents pour installer l'application.

Pour la partie logicielle, la plus grosse difficulté a été de comprendre les méthodes fournies par l'API Google Maps ainsi que la grosse majorité des fonctions fournies dans le code exemple pour pouvoir l'adapter et le réutiliser de manière à obtenir un résultat convenable.

CONCLUSION

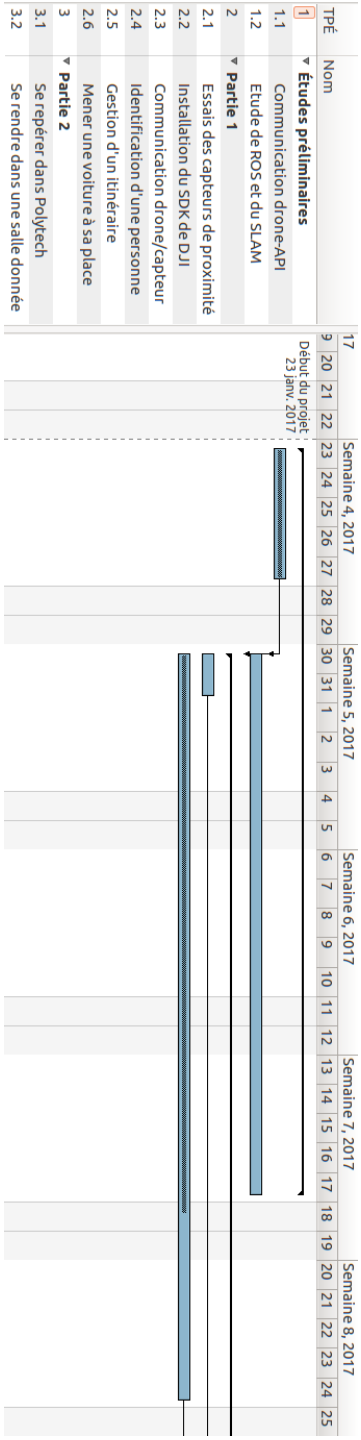
Afin de clôturer ce rapport, nous pouvons dire que ce module de projet a été très instructif pour nous. En effet, nous avons pu nous former de façon concrète sur de nombreux points tels que le langage Java présent dans Android Studio ou encore le Python utilisé avec la Raspberry. De plus, nous avons réussi à fournir un travail qui fonctionne et qui peut être utilisé malgré les nombreuses difficultés rencontrées. Ce projet nous a donc permis de nous confronter à ces problèmes, les étudier et les solutionner comme pour un véritable travail en entreprise. Malheureusement nous n'avons pas pu mener notre projet à terme et de nombreuse fonctions telles que l'identification du visiteur pourraient être ajoutées. Cela nous a donc permis de travailler sur un sujet qui, nous le pensons amènera à se développer avec l'expansion des drones, qu'ils soient volant, terrestres ou maritimes.

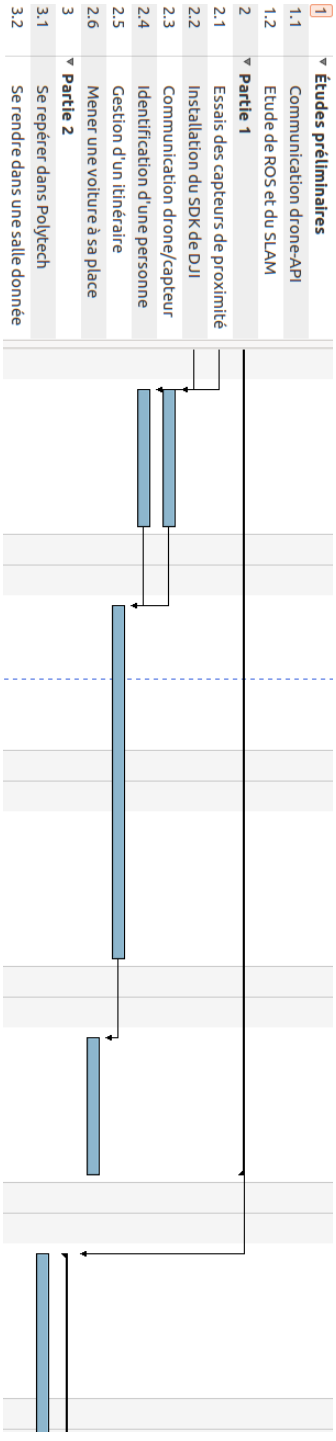
ANNEXES

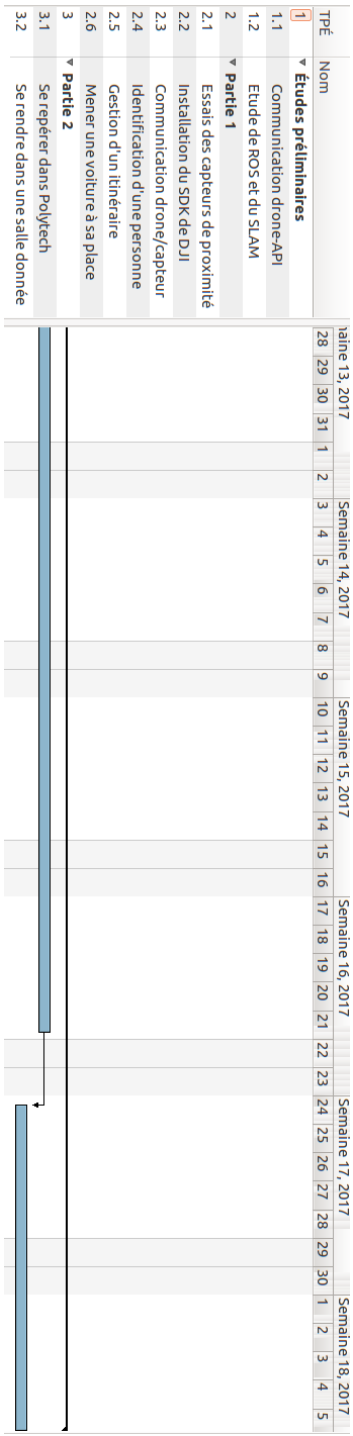
1/ Diagramme de Gantt du projet

2/ Code complet de la nacelle.

1/ Diagramme de Gantt du projet







2/ Code complet de la nacelle.

```
# Liste des importations
import time
import RPi.GPIO as GPIO

# Importation du module ADS1x15
import Adafruit_ADS1x15

# Configuration PIN de sortie
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7, GPIO.OUT)

# create an ADS1015 ADC (12-bit) instance.
adc = Adafruit_ADS1x15.ADS1015()

GAIN = 8
print("")
# impression des colonnes de valeurs
print('| capteur Droit | Capteur Avant | Capteur Gauche |'.format(*range(4)))
print('-' * 50)

# Main
while True:
    # Read all the ADC channel values in a list.
    values = [0]*4
    for i in range(4):
        # Lecture des valeurs du ADC
        values[i] = adc.read_adc(i, gain=GAIN)

    # Affichage des valeurs recuperee
    print('|    {0:>6}     |    {1:>6}     |    {2:>6}     |'.format(*values))


    if values[0]>=1000:               # Test capteur gauche
        print("Warning gauche")
        GPIO.output(7, 1)             # Clignotement LED

    if values[1]>=1000:               # Test capteur avant
        print("Warning avant")
        GPIO.output(7, GPIO.HIGH)     # Clignotement LED

    if values[2]>=1000:               # Test capteur droit
        print("Warning droite")
        GPIO.output(7, GPIO.HIGH)     # Clignotement LED

    # pause de 0.5 sec pour ne pas surcharger le buffer
    time.sleep(0.5)
    GPIO.output(7, GPIO.LOW)
```