

UNIVERSITÉ DE MONTPELLIER
FACULTÉ DES SCIENCES
Informatique



FACULTÉ DES SCIENCES

RAPPORT DE PROJET

T.E.R. HLIN601

**Automatisation d'ensemencement de
graines avec un drone**

Proposé par :

SIMIONE Jeremy & DUCHON Damien & HENRIKSEN LAREZ Leif & GUILLEN Victor

Année : 2020

Encadrante :

Hinde Bouziane

Nous remercions notre encadrante Mme Hinde Bouziane pour son accompagnement, sa bienveillance et ses conseils tout au long de ce projet qui nous ont permis de le porter à son état actuel.

Table des matières

Introduction	2
1 Organisation du projet	5
1.1 Objectif et mise en place du cahier des charges du projet	5
1.2 Méthode et organisation du travail	6
1.3 Outils de travail collaboratif	6
1.4 Matériel utilisé	7
2 Conception et implémentation	9
2.1 Algorithmes et implémentation du parcours	9
2.1.1 Génération des points de passages/semaison	9
2.1.2 Arbre couvrant de poids minimum	11
2.1.3 Le calcul du parcours	12
2.1.4 Point final : le Voyageur de commerce	13
2.2 Conception de l'application Android	15
2.2.1 Activation de l'application et intégration du SDK	16
2.2.2 Affichage de la carte et interface utilisateur	18
2.2.3 Implémentation de la mission du drone	19
2.2.4 Localisation de l'appareil sur la carte	20
2.2.5 Création des points des passages et du parcours	20
2.2.6 Implémentation de la mission	21
2.2.7 Communication avec le Raspberry	22
2.3 Communication entre l'application et le moteur : relâchement des graines	22
2.3.1 Principe de base	22
2.3.2 Établir la connexion	22
2.3.3 Contrôle du moteur avec le Raspberry	23
3 Bilan et difficultés rencontrées	25
3.1 Avancement du projet	25
3.2 Difficultés rencontrées	25
3.3 Connaissances et apprentissage	26
Conclusion	27

Annexes	28
Table des figures	32
Bibliographie	33

Introduction

Présentation du Sujet

Dans le cadre du projet semestriel de L3 informatique à la faculté des sciences de Montpellier, nous nous sommes fixés pour objectif d'automatiser l'ensemencement de graines sur un terrain donné avec un drone et via une application smartphone.

Le drone doit pouvoir suivre un parcours via une application Android tout en revenant à sa position initiale. C'est lors de ce parcours que le drone ensemence une graine à une position donnée. Les graines doivent être reparties de manière régulière et séquentielle.

Des problèmes évidents nous sautent tout de suite aux yeux : faire attention aux éventuels obstacles, temps de vol du drone limité, optimisation de l'espace par exemple.

Nous avons besoin d'un Raspberry Pi, un smartphone Android. Nous détaillerons le matériel et les outils dans la prochaine section puisque cela constitue une grande partie de notre projet.

Éléments de motivation

Quelles ont été nos motivations pour avoir choisi ce sujet? Nous savions que l'opportunité de travailler sur un drone était unique et à saisir. L'application concrète des notions et



FIGURE 1 – Exemple d'un drone de relâchement de graines

concepts que nous apprenons depuis maintenant 3 ans sur un drone nous a tout de suite beaucoup plu. Nous n'avons pas l'habitude, dans notre cursus en tout cas, de penser et programmer sur un outils de ce type.

De plus, ayant déjà approfondis quelques connaissances dans le passé sur la réalisation d'application Android, nous voulions pousser encore un peu plus loin ce domaine de compétence, et ce sujet s'y apprête bien. Nous savions qu'en choisissant de réaliser une application nous devrions établir une communication entre ces deux outils : le drone et le smartphone. Nous trouvons alors ce sujet complet, c'est-à-dire que l'éventail des compétences requises balaie plusieurs notions essentiels à notre goût, ce qui renforce notre intérêt pour celui-ci. De fait l'utilisation et la connaissance des applications smartphone (ici Android donc) est plus que dans l'ère du temps.

Ensuite pourquoi l'ensemencement de graine ? Nous nous sommes dit qu'il fallait travailler sur une surface d'un terrain assez conséquent : par exemple un champ. Ensuite nous voulions apporté une solution à un problème, ensemencer des graines de manières automatique, rapide, propre et durable. Nous sommes sensibles à l'entretien de l'environnement et ce projet pourrait répondre à ce problème de manière sensé.

Différentes pistes, différentes solutions

Afin de réaliser ce projet au mieux, nous avons exploré plusieurs solutions, toutes avec leurs avantages mais surtout leurs inconvénients. Pour la semaison des graines le drone lui-même n'étant évidemment pas capable de réaliser cette tâche nous avons du réfléchir a ce que nous allions utiliser pour pouvoir semer des graines depuis l'appareil. La seule solution réalisable que nous avons trouvé a été de se servir d'un nano-ordinateur : le Raspberry Pi qui est une machine capable d'exécuter un système d'exploitation et des programmes comme un serveur ou encore contrôler des éléments.

— 1 ère solution :

Comme solution initiale nous avons eu comme idée de connecter un capteur GPS directement au Raspberry.

Ce qui nous permettrait de récupérer les coordonnées GPS en temps réel et d'envoyer des ordres directement au Raspberry sans passer par l'application Android.

La difficulté étant d'une part de réussir à connecter le capteur au Raspberry, et d'autre part de récupérer les informations fournies par le capteur et bien sûr de faire le lien avec l'application Android.

— 2ème solution :

Pour la deuxième solution après concertation, une approche du problème à l'aide d'un capteur de mouvement (gyroscope ou autre) nous semblait plausible. Le Raspberry serait en capacité de lâcher une graine quand le drone est à l'arrêt c'est à dire quand le gyroscope serait fixe (sans vitesse)

La difficulté étant ici d'arriver à comprendre les données récupérer par le gyroscope et connaître le temps de stabilisation de celui-ci afin de savoir si nous sommes à l'arrêt ou non.

— **3ème solution :**

Pour cette solution nous avons pensé à une indépendance de la communication entre le Raspberry et le drone, c'est à dire que le drone serait branché directement au Raspberry et qui lui serait en attente d'un ordre (ordre donné via l'application Android).

Le problème étant de savoir si oui ou non nous pouvions brancher le Raspberry en direct et communiquer des informations via USB, ce qui pourrait être possible si l'API DJI contenait des fonctions permettant la communication entre plusieurs appareils branchés au drone en ayant par exemple une fonction nous donnant la visibilité des ports de connections du drone.

— **Dernière solution :**

Nous avons pensé à une communication des différents éléments/systèmes via un client/serveur c'est à dire :

- L'application qui va récupérer les informations actuelles du drone pendant sa mission et qui va communiquer les informations.
- Le Raspberry Pi qui va attendre un message grâce à un serveur constamment en écoute, et qui, après réception, va activer un servomoteur qui, lui, va ouvrir la trappe permettant de lâcher les graines.

La difficulté étant surtout de pouvoir trouver un réseau wi-fi pour communiquer car communiquer par le réseau mobile rendrait cette solution beaucoup trop complexe vis à vis de nos connaissances en réseaux.

Chapitre 1

Organisation du projet

1.1 Objectif et mise en place du cahier des charges du projet.

Avant de nous lancer dans le développement et la conception, nous nous sommes réunis dans le but de se mettre d'accord sur les aspects essentiels du projet et ceux que nous voulions approfondir. Mme Bouziane nous laissant libre sur le niveau des ambitions pour réaliser ce projet.

L'objectif est donc de mettre en place une application Android permettant de générer un parcours sur un terrain donné. Il est nécessaire également d'établir une communication entre le drone et l'application Android. On utilisera aussi un nano-ordinateur qui contrôlera la semaison de graines en fonction de l'emplacement courante du drone.

Cette application sert donc d'interface utilisateur pour la gestion du drone et de son parcours d'ensemencement.

Nous avons donc défini un cahier des charges précis afin d'établir les fonctionnalités et les contraintes que notre application doit respecter.

Un parcours optimisé : L'objectif principal est d'automatiser un parcours par rapport à une surface donnée. Nous voulons évidemment que le chemin soit adapté et optimisé, c'est-à-dire qu'il prenne en compte d'éventuels obstacles, que le drone ne parcours pas de distances inutiles par exemple.

Application Android : Nous voulons développer une application Android pour générer ce parcours, commander le drone et surveiller en temps réel son déplacement à l'aide de sa caméra intégré. Cette application permet une meilleure ergonomie, une meilleure portabilité et nous semble être la solution la plus évidente.

Choix du terrain : Notre application contient une carte Google Maps permettant de placer des points. L'utilisateur sélectionne une surface en plaçant manuellement des points sur la carte ce qui génère les points de passage du terrain choisi.

Communication entre l'application et le Raspberry : L'application doit pouvoir communiquer avec le Raspberry quand la position d'un emplacement de semage est atteint.

Interface utilisateur : L'interface doit être la plus intuitive possible pour l'utilisateur avec des boutons simples permettant de réaliser les actions souhaitées et permettant de voir le suivi du drone.

Semaison des graines : Une trappe doit être actionnée (ouverte) afin de lâcher les graines sur le point de semage.

Génération automatique des points de passage du drone : Les points de passages (semaison) sont générés automatiquement par l'application. L'utilisateur peut paramétrer la distance entre ces points et les limites du terrain choisi.

1.2 Méthode et organisation du travail

Nous nous sommes retrouvés quotidiennement pour rendre compte des avancées de chacun. L'avancement du projet s'est fait dans l'ordre de priorité défini au départ. Nous nous sommes également réunis au complet à plusieurs reprises pour effectuer un bilan de mi-parcours, vérifier si les deadlines étaient respectées et discuter de la suite du projet.

Nous avons travaillé soit chacun de notre côté sur des tâches simples, soit en groupe lorsque les tâches étaient un peu plus complexes. Nous nous retrouvions donc sur le campus de la faculté ou en appel vocal sur nos ordinateurs respectifs pour travailler ensemble.

Environ toutes les deux semaines, parfois plus, nous nous sommes réunis avec notre encadrante Mme. Bouziane afin de faire le point sur l'état d'avancement du projet. Ces réunions nous ont permis de discuter des difficultés rencontrées au fur et à mesure avec notre encadrant. Mme Bouziane nous a aussi donné des pistes de documentation et des conseils pour la réalisation du projet, par exemple pour établir une communication entre l'application et le drone.

1.3 Outils de travail collaboratif

Nous avons choisi d'utiliser **GitHub**. Il permet la gestion des versions du projet. GitHub est très pratique et facile d'utilisation. Il facilite le travail en collaboration sur un projet informatique tel que celui-ci.

Pour faciliter et centraliser la communication, les ressources, les fichiers entre tous les membres du groupe et tenir une chronologie dans la réalisation de ce projet, nous avons créé un serveur Discord. Nous pouvions ainsi garder une trace de l'avancée de chacun et communiquer en vocal ou par écrit plus aisément et spontanément.

A propos des langages que nous utilisons :

Nous sommes contraint d'utiliser le langage de programmation **Java**, qui est un langage de programmation orienté objet (nous avons tous utilisé naturellement l'IDE Eclipse), puisque le développement d'une application android nécessite l'utilisation d'Android Studio et que le langage requis est donc Java. Tout l'aspect algorithmique a donc aussi été développé avec ce langage.

Pour envoyer et recevoir des appels depuis le Raspberry Pi nous avons utilisé le langage **Python** qui est un langage de programmation interprété. Cette interface nous permet d'actionner un petit moteur déclenchant le semage de graines.

Énoncé rapidement plus haut mais néanmoins non négligeable, nous avons donc utilisé **Android Studio** afin de développer l'application. L'utilisation de cet environnement de développement est primordial et obligatoire pour une confection convenable d'application.



FIGURE 1.1 – Logos des outils de travail utilisés

1.4 Matériel utilisé

Naturellement pour ce projet nous utilisons un drone. La faculté des sciences de Montpellier nous a fournis un drone de la marque Dji, le modèle étant le Phantom 3 SE. Il pèse 1,2Kg, a une longueur (sur la diagonale) de 35cm et est équipé d'une caméra de 12M pixels. Il se contrôle à l'aide d'une Radiocommande, celle-ci nous sera très utile afin d'établir une communication entre les différents éléments : le drone, le smartphone et le Raspberry Pi.



FIGURE 1.2 – Dji Phantom 3 SE et sa radiocommande

Afin d'actionner mécaniquement la trappe de semaison, nous utilisons un SG90 Mini Servo Moteur. Un petit moteur simple et très léger.

Pour contrôler ce petit moteur, nous utilisons donc un Raspberry Pi, un nano-ordinateur monocarte conçu par des professeurs d'informatique de l'université de Cambridge. Cet ordinateur, de la taille d'une carte de crédit permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux notamment.

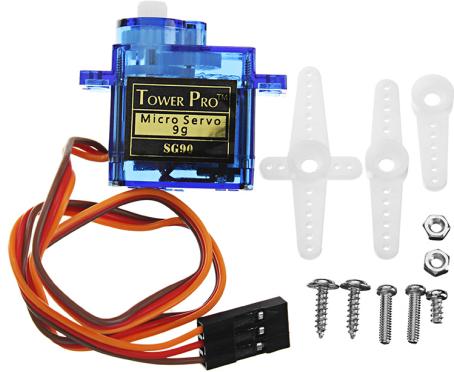


FIGURE 1.3 – SG90 Mini Servo Moteur

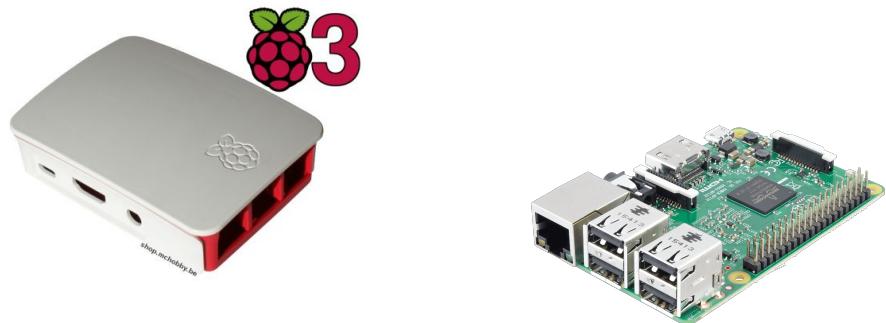


FIGURE 1.4 – Raspberry Pi 3 Model B et son boîtier

Enfin, nous utilisons notre smartphone personnel, sous Android. N'importe quelle smartphone Android aurait pu faire l'affaire. De notre côté il s'agit d'un Xiaomi Redmi Note 7.



FIGURE 1.5 – Xiaomi Redmi Note 7

Chapitre 2

Conception et implémentation

2.1 Algorithmes et implémentation du parcours

Nous avons choisi de créer un parcours dans l'esprit du célèbre "Voyageur de commerce". Celui-ci constraint d'avoir le même point de départ et d'arrivée et doit obligatoirement passer par tous les points de passages une et une seule fois. L'objectif est donc de faire gagner du temps à ce voyageur de commerce. Il faut lui organiser un parcours optimal (en temps minimum) de tel sorte qu'il ne passe pas deux fois au même endroit et en prenant en compte la distance parcourue. C'est dans cet esprit que nous voulons réaliser le parcours pour le drone.

Algorithm 1 Problème "Voyageur de commerce"

Entrée: Un ensemble de n points dans le plan : $1, \dots, n$.

Sortie: Une énumération des sommets (v_1, \dots, v_n) telle que : $d(v_1, v_2) + d(v_2, v_3) + \dots + d(v_{n-1}, v_n) + d(v_n, v_1)$ soit minimum.

L'implémentation de cet algorithme se fera avec le langage de programmation Java.

2.1.1 Génération des points de passages/semaison

Avant tout, nous devons générer un quadrillage de points sur le terrain donné. Pour cela on crée une grille de point sur laquelle nous superposons le terrain. Enfin nous gardons en retour uniquement le terrain et les points qui y sont internes. Le schéma [Figure 2.1](#) illustre le processus décrit.

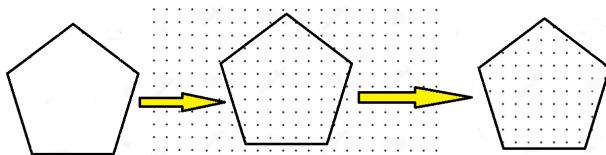


FIGURE 2.1 – Génération des points de passage

Le polygone est paramétrable sur le nombre et la longueur de ses côtés. Les points le sont

aussi au niveau de la distance entre eux, verticalement et horizontalement. Les points internes au polygone seront donc les points de passage du drone et d'ensemencement.

Algorithme Point dans polygon :

L'algorithme utilisée pour déterminer si un point est dans un polygone, est un algorithme de ray casting. Pour chaque point p_1 , l'algorithme génère un autre point p_2 à droite et très éloigné de p_1 , créant une droite de p_1 vers p_2 , après il compte le nombre d'intersections entre la droite p_1, p_2 et les côtés du polygone, si le nombre est pair alors le point p_1 est dehors, sinon il y est dedans.

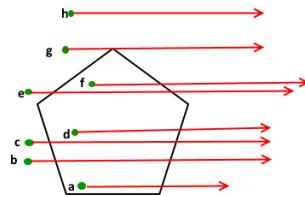


FIGURE 2.2 – Calcul des intersections

Algorithm 2 Problème "Point dans polygone"

Entrée: Ensemble des côtés d'un polygone : C, Ensemble des points à tester : P

Sortie: Ensemble des points dans le polygone

```

1: pointsInternes  $\leftarrow \emptyset$ 
2: pour  $p_1$  dans P faire
3:    $p_2 \leftarrow (+\infty, p_1.y)$ 
4:    $nbIntersections \leftarrow 0$ 
5:    $rayDroite \leftarrow (p_1, p_2)$ 
6:   pour c dans C faire
7:     si intersection(c, rayDroite) alors
8:        $nbIntersections++$ 
9:     fin si
10:    fin pour
11:    si  $nbIntersections \% 2 > 0$  alors
12:      pointsInternes  $\leftarrow i$  pointsInternes  $\cup p_1$ 
13:    fin si
14:  fin pour
15: renvoie pointsInternes

```

Implémentation en Java :

L'implémentation utilise trois classes : Polygon, Point et Line.

Polygon contient comme attributs, une liste des côtés du polygone : la liste des points internes, et la distance des points internes entre eux et les côtés du polygone; et comme méthodes : les getters, setters, et l'algorithme Point dans polygone.

Point est défini par un triplet de coordonnées (x,y,z) et la latitude et longitude, Point contient aussi les méthodes pour calculer la distance entre deux points, transformer latitudes et longitudes en coordonnes cartésiennes.

Line représente une droite d'un point a vers un point b, et contient les méthodes nécessaires pour déterminer s'il y a une intersection entre deux droites.

Pour appliquer l'algorithme Point dans polygone, une instance de la classe Polygone est créé a partir d'une liste des points avec des coordonnes GPS, après les coordonnes cartésiennes sont calcules, un quadrillage de points est généré et avec l'aide des classes Point et Line, l'algorithme pour déterminer si un point est dans le polygone est applique, le résultat du dernier est sauvegarde dans un attribut de Polygon appelé pointsInternes pour être utilisé dans les algorithmes suivants.

2.1.2 Arbre couvrant de poids minimum

L'ensemble de points ainsi générée, nous pouvons considérer un graphe. Ce graphe est alors constitué de l'ensemble de points interne au polygone.

Nous considérons ce graphe, pour commencer, comme "complet", c'est-à-dire que pour tout couple de points (x,y) il existe une arête, autrement dit, tout point est relié à tout les autres. Plus formellement, on construit K_n sur $1, \dots, n$ pondéré par $w(ij) = d(i,j)$.

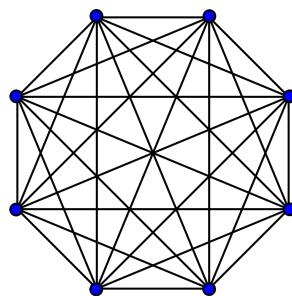


FIGURE 2.3 – Exemple de graphe complet (K_8)

Ensuite, l'objectif est de trouver un arbre couvrant de poids minimum (ACPM) pour ce graphe. Plusieurs algorithmes de construction d'ACPM existent sur lesquelles nous pouvons nous appuyer telles que l'algorithme de Boruvka ou l'algorithme de Prim par exemple. Tout ces algorithmes sont de type glouton. Arbitrairement nous avons choisis de nous inspirer de l'algorithme de Kruskal.

Algorithm 3 Kruskal

Entrée: Un graphe connexe $G = (V, E)$ donné par liste d'arêtes avec une fonction de poids w sur les arêtes de G .

Sortie: Un ensemble A d'arêtes de G formant un arbre couvrant de poids minimum (ACPM) de G .

Cet algorithme s'exécute en $O(n^2 + m \log n)$ dans le pire des cas.

Implémentation en Java :

Nous avons crée une liste de *Point* contenant tout les points du graphe. Nous avons également implémenté une classe *Edge* contenant deux points et la distance qui les séparent. Pour modéliser le graphe nous avons crée une liste contenant des éléments de type *Edge* représentant le graphe complet initiale. Puis nous trions la liste de *Point* (par pairs) par ordre croissant des distances.

Pour effectuer le calcul intermédiaire de composante connexe dont nous avons besoin pour la construction de Kruskal, nous avons utilisé la structure de donnée *HashMap*. Cette structure de donnée est une implémentation basée sur la table de hachage (*Hashtable*) de l'interface *Map*. Elle fournit toutes les opérations de mappage facultatives et autorise les valeurs *null* et la clé *null* (la classe *HashMap* est à peu près équivalente à *Hashtable*, sauf qu'elle autorise les valeurs *null*).

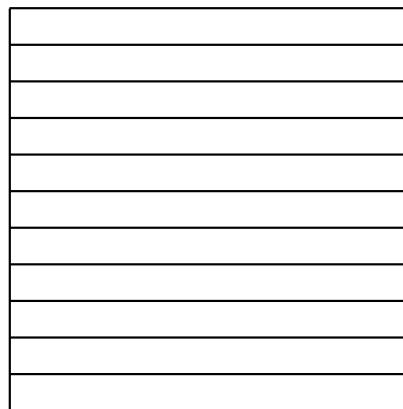


FIGURE 2.4 – Résultat obtenu après Kruskal

En annexe (Figure 3.1), l'extrait de code de notre algorithme.

2.1.3 Le calcul du parcours

Maintenant que nous avons notre Arbre, la prochaine étape est de réaliser un parcours type "Parcours en profondeur" sur celui-ci. Ce parcours doit optimiser le trajet et donc le temps de vol du drone.

Pour cet algorithme nous considérons évidemment le graphe de Kruskal obtenu. Il nous faudra implémenter le graphe comme une liste de voisins, ça sera la structure sur laquelle nous nous appuierons pour cet algorithme.

Algorithm 4 Parcours en Profondeur

Entrée: Un graphe $G = (V, E)$ donné par listes de voisins, gérées comme des piles, et r un sommet de G .

Sortie: Une fonction père : $V \rightarrow V$.

Cet algorithme s'exécute en $O(n + m)$ dans le pire des cas.

Implémentation en Java :

L'algorithme contient plusieurs structures de données. Une *HashMap* tel que à un point on associe sa liste de voisins *HashMap<Point, ArrayList<Point>> voisins*. Nous avons utilisé le même type de stockage pour les père : à un point on associe son père. Ensuite comme nouvelle structure de donnée nous avons utilisé la pile, ou la *Stack* en Java.

La classe *Stack* représente une pile d'objets dernier entré, premier sorti (ou en anglais "last-in-first-out" : LIFO). Elle étend la classe *Vector* avec cinq opérations qui permettent de traiter un vecteur comme une pile. Les opérations *push* et *pop* habituelles sont fournies, ainsi qu'une méthode pour jeter un œil à l'élément supérieur de la pile, une méthode pour tester si la pile est vide et une méthode pour rechercher la position d'un élément dans la pile. Cette pile est un élément essentiel au bon déroulement de l'algorithme.

Le principe du parcours est en fait assez simple, il explore intégralement les chemins les uns après les autres. Pour chaque sommet rencontré, il marque le sommet courant comme visité (de façon à ne pas l'explorer à nouveau), et enchaîne avec le premier sommet voisin. Il arrête ce mode de fonctionnement quand un sommet n'a plus de voisins, ou que tous ses voisins sont déjà marqués. Il revient (ou "backtrack") alors au premier sommet père ayant encore un voisin. L'exploration s'arrête quand tous les sommets ont été visités. Nous avons décidé de modifier un minimum cet algorithme afin de, au lieu de faire un backtrack, chercher le point le plus proche une fois qu'on a atteint le dernier point pour pouvoir parcourir une et une seule fois chaque point. Une fois tous les points parcourus nous avons décidés de relier manuellement le point final et le point de départ pour que le drone revienne à sa position initiale une fois le parcours terminé.

En annexe (Figure 3.2), l'extrait de code de notre algorithme.

2.1.4 Point final : le Voyageur de commerce

Enfin, il ne nous reste plus qu'à boucler le tout. Le principe du "Voyageur de Commerce" est qu'il ne doit y avoir qu'un seul passage par point, ce qui n'est plus une contrainte à résoudre

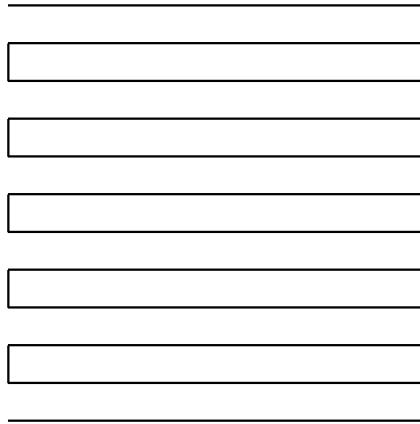


FIGURE 2.5 – Résultat obtenu après le parcours en profondeur, Le drone devra suivre ce parcours

à ce stade. Il faut également qu'il termine son parcours à l'endroit où il l'a commencé. Rien de plus simple, nous devons ajouter une dernière itinérance (donc une dernière arête au graphe) du dernier point de passage à la position initiale. De cette manière là, nous sommes certain que le "voyageur" ne passera qu'une seule fois par point tout en retrouvant son point de départ.

Implémentation en Java :

Le Voyageur de Commerce se construit en trois étapes :

- Étape 1 : Générer les points de passages, et donc de semaison, pour une surface donnée. Ces points sont ceux internes au polygone passé en paramètres.
- Étape 2 : Modéliser un ACPM à partir de l'algorithme de Kruskal sur le graphe complet comportant tous les points internes au polygone.
- Étape 3 : "Parcourir en Profondeur" cet arbre de Kruskal afin d'y retourner un chemin optimisé respectant l'ensemble des contraintes du Voyageur de Commerce.

Pour illustrer ces trois étapes concrètement en Java, ci-dessous notre algorithme interprétant le "Voyageur de Commerce" :

"Malgré la simplicité de son énoncé, il s'agit d'un problème d'optimisation pour lequel on ne connaît pas d'algorithme permettant de trouver **une solution exacte** rapidement dans tous les cas. Plus précisément, on ne connaît pas d'algorithme en temps polynomial, et sa version décisionnelle [...] est un problème NP-complet, ce qui est un indice de sa difficulté.

C'est un problème algorithmique célèbre, qui a généré beaucoup de recherches et qui est souvent utilisé comme introduction à l'algorithmique ou à la théorie de la complexité. Il présente de nombreuses applications que ce soit en planification et en logistique, ou bien dans des domaines plus éloignés comme la génétique [...] " - wikipedia.org

```

339 public ArrayList<Point> voyageurDeCommerce(Polygon poly) {
340
341     // *** ETAPE 1 *** : établir les points (et leurs coordonnées) internes au polygon :
342     // la forme, la taille et donc la surface que nous devons traiter (celui passé en paramètres)
343     setListePoint(poly.getPointsInternes());
344
345     // Noter le nombre de points, donc le nombre de point de passage
346     // Ainsi que le nombre d'arêtes du graphe complet obtenu avec cet ensemble de points
347     int nbPoints = this.listePoint.size();
348     int nbAretes = (nbPoints*(nbPoints-1))/2;
349
350     // *** ETAPE 2 *** : On initialise et stock toutes les arêtes du graphe et leurs distance
351     // puis les arêtes sont triées par ordre croissant des distances
352     initDistances(nbPoints);
353
354     // On trouve un arbre couvrant de poids minimum pour ce graphe.
355     // --> l'algorithme de Kruskal, que l'on stock dans la liste de point "krus"
356     this.krus = kruskal(nbPoints, nbAretes);
357
358     // *** ETAPE 3 *** : Réaliser un parcours en profondeur
359     // de l'arbre couvrant de poids minimum (ACPM) modélisé précédemment
360     ArrayList<Point> parcours = parcours(krus);
361
362     return parcours;
363 }

```

FIGURE 2.6 – Voyageur de Commerce (Java)

En effet, le Voyageur de Commerce est un problème NP-difficile. L’itinéraire construit est au pire 2 fois plus long que le meilleur itinéraire possible. Dans son cas il est possible d’approcher la solution optimale à un facteur 2 en temps polynomial : on parle d’une **2-approximation**.

Figure 2.7, affichage du résultat obtenu de notre version du "Voyageur de Commerce", sur une surface rectangle dont les points sont à une distance de 30. Cette illustration est exactement le chemin que devra emprunter le drone ainsi que tous les points de passages obligatoire de saison. C'est un chemin optimal (mais pas l'unique) pour minimiser le temps de vol, ce qui était notre objectif premier sur cette partie du projet.

2.2 Conception de l’application Android

Pour la réalisation de l’application Android nous nous sommes orientés vers le site DJI developer qui est un site conçu pour les développeurs d’applications de produits DJI. Après des recherches nous avons donc utilisé le SDK (Software Development Kit) de DJI qui a été le point d’entrée obligatoire pour pouvoir communiquer avec le drone et connaître les outils pour le contrôler. Nous avons donc suivi plusieurs tutoriaux pour comprendre les outils utilisés et utiliser ce kit de développement. La réalisation de l’application a donc été découpée en plusieurs parties :

- L’activation de l’application pour pouvoir utiliser le SDK
- L’utilisation d’une carte GoogleMaps
- Le contrôle du drone
- Mise en place d’un client UDP -> communication raspberry

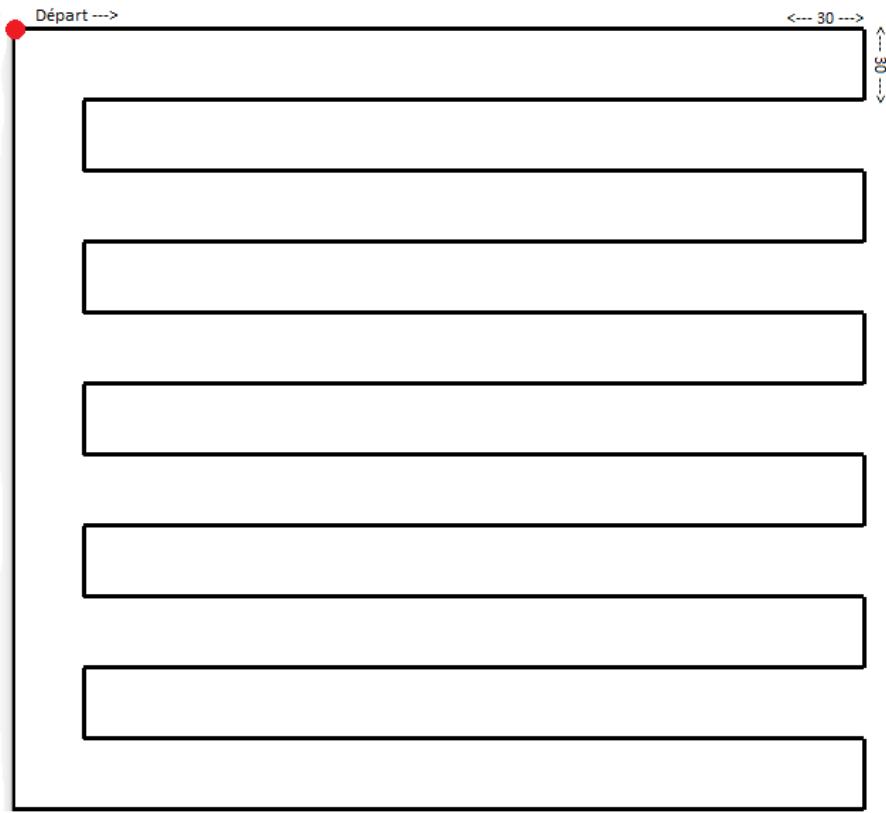


FIGURE 2.7 – Résultat du voyageur de commerce

Implémentation en Java :

Nous avons utilisé le langage Java pour réaliser notre application car nous connaissons bien ce langage mais surtout parce que c'est le plus utilisé dans le développement d'applications Android.

2.2.1 Activation de l'application et intégration du SDK

L'intégration du SDK à l'application est réalisée dans le fichier build.gradle de l'application qui correspond au fichier qui gère les dépendances, bibliothèques de l'application. Le fichier build.gradle est fourni, les seules différences avec un fichier build.gradle classique sont l'utilisation de certaines bibliothèques et dépendances indispensables à l'utilisation du SDK DJI.

Après cette étape nous pouvons vérifier la bonne importation du SDK dans la structure du projet sur l'IDE.

L'activation du SDK nécessite de créer un compte sur le site DJI developer pour avoir accès au Developer Center afin d'enregistrer une application car chaque application a besoin d'une clé unique (API) pour fonctionner avec le SDK DJI. Cette clé est obtenue en utilisant le même nom de package que l'application enregistrée sur le site et en plaçant la clé fournie dans le fichier AndroidManifest de l'application.

Ce fichier contient toutes les permissions/instructions nécessaires au bon fonctionnement du

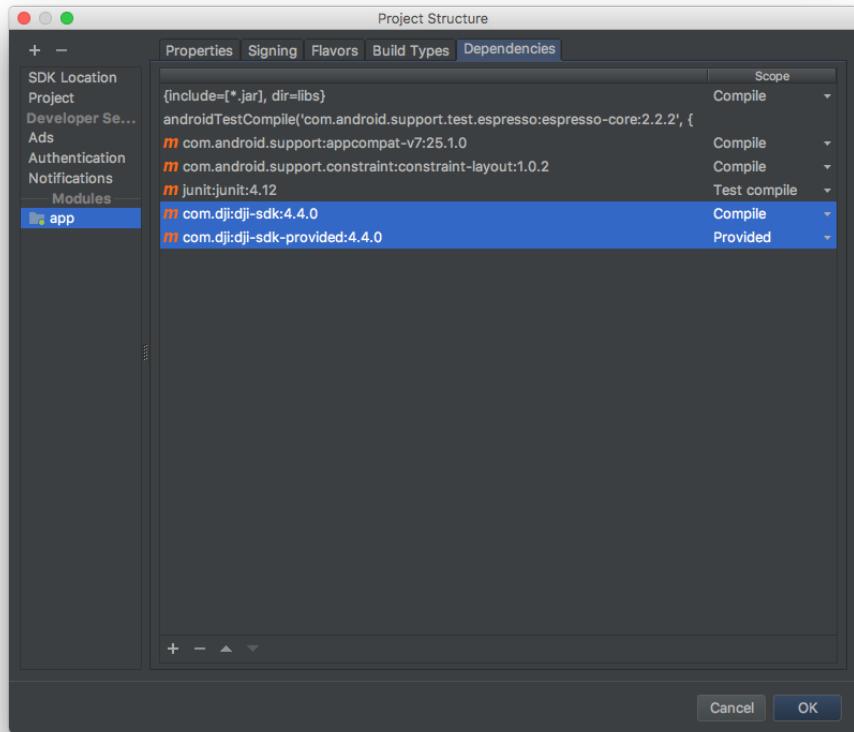


FIGURE 2.8 – Capture d'écran de la structure du projet

SDK. Le fichier `AndroidManifest` est indispensable car il gère les différentes activités (les vues) de l'application, deux dans notre cas et spécifie les options du projet, contient différentes informations sur lui, ses permissions et encore d'autres éléments indispensables pour l'application. Ce fichier est toujours généré par défaut par Android Studio et est peut être complété en cas d'ajout d'activités par exemple.

Finalement côté classes, la définition d'une méthode dans la classe "MApplication" est indispensable car certaines classes du SDK ont besoin d'être chargées avant d'être utilisées, le processus de chargement est effectué par la méthode `Helper.install()`, cette classe sert à éviter des plantages de l'application.

La classe `ConnectionActivity` contient plusieurs variables, qui sont des `TextView` afin de visualiser l'interface d'enregistrement du produit et d'un bouton permettant d'ouvrir l'activité principale si un produit est connecté et qui est par défaut bloqué, on ne peut appuyer dessus. La méthode `onCreate()` qui est utilisée pour lancer l'activité appellera la méthode "`checkAndRequestPermissions()`" pour vérifier et demander des autorisations d'exécution. La méthode "`checkAndRequestPermissions()`", elle, aidera à appeler la méthode "`startSDKRegistration()`" qui est une méthode de callback pour enregistrer le SDK de l'application. Enfin la méthode "`onRequestPermissionsResult()`" aidera à vérifier si l'application dispose de suffisamment d'autorisations, si c'est le cas, la méthode "`startSDKRegistration()`" est donc appelée pour enregistrer l'application.

Cette classe contient aussi une méthode qui va rafraîchir l'interface si l'application a bien été enregistrée et le produit connecté en déverrouillant le bouton "OPEN" et en affichant les

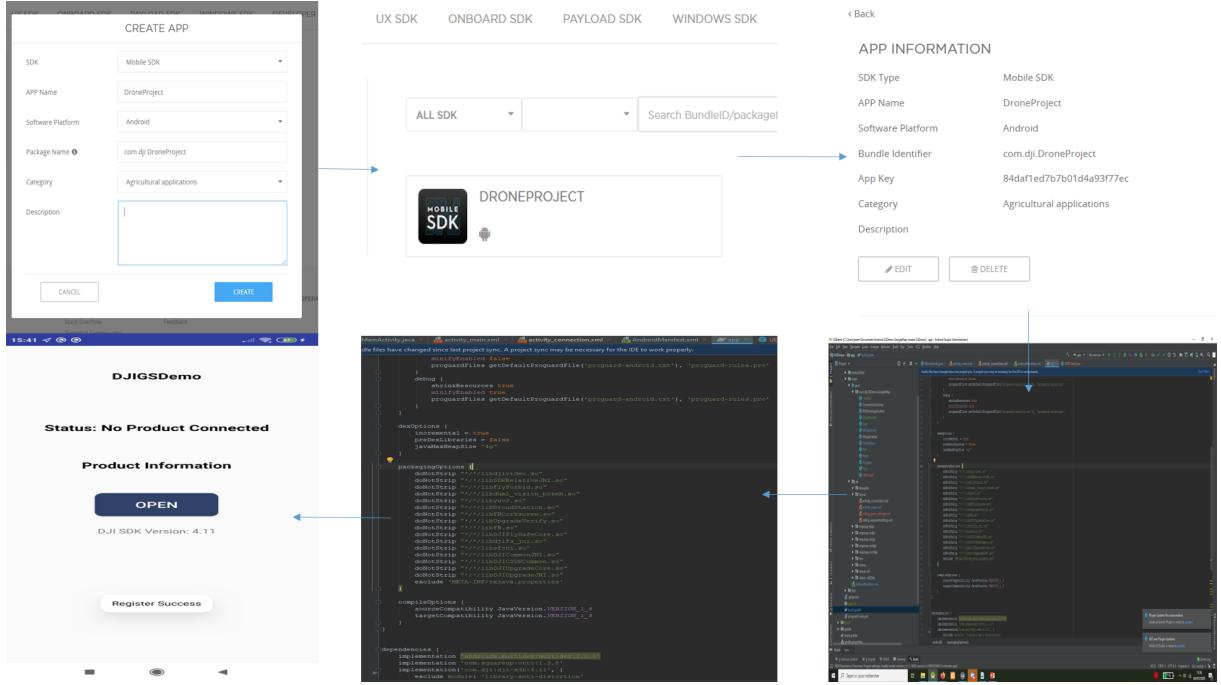


FIGURE 2.9 – Schéma de l'activation de l'application

éléments d’informations du produit connecté à l’appareil Android. Etat de connexion qui est défini dans la classe DJIDemoActivity grâce à une variable mProduct qui sert à connaître l’état de connexion à l’appareil. Elle contient une méthode de callback pour vérifier l’état d’enregistrement du SDK et le statut de connexion du produit sur l’appareil et sert à compléter la méthode startSDKRegistration() en enregistrant grâce un callback l’état de connexion du produit et l’état d’enregistrement. Tout ceci est représenté visuellement par le fichier XML connection_activity.xml correspondant à cette activité. Cette activité est affichée tant que l’utilisateur n’a pas connecté son appareil à l’application. Un message indiquant la réussite ou non de l’enregistrement s’affiche sur l’appareil Android.

2.2.2 Affichage de la carte et interface utilisateur

Nous avons choisi d’utiliser pour cette partie l’API GoogleMaps car c’est une API très bien documentée et facile d’utilisation, de plus, le site DJI developer contenait un tutoriel pour utiliser cette API et réaliser des points de passage à suivre par le drone ce qui nous a été bien utile comme modèle pour réaliser un parcours généré de façon automatique.

Pour utiliser la carte nous avons suivi une procédure assez simple qui consistait à activer des services pour notre projet et ajouter la clé API de GoogleMaps dans le fichier AndroidManifest.xml et qui ressemblait à l’intégration du SDK DJI sans classes nécessaires.

L’interface utilisateur possède huit boutons qui sont définis dans le fichier activity-main.xml qui contient des balises correspondant aux différents boutons et aussi un bouton pour la localisation de l’appareil android et un fragment qui représente une partie de notre activité qui contiendra la carte. Ce fichier est l’activité principale et donc l’interface utilisateur de l’application (Figure 2.10).

Pour fonctionner cette interface et cette carte ont besoin d’être implémentées par des



FIGURE 2.10 – Une capture d'écran de l'interface utilisateur

variables et méthodes dans la classe MainActivity. La mMinActivity contient donc maintenant neuf variables correspondant aux différents boutons, une variable pour notre carte, plusieurs méthodes dont une qui va se charger d'initialiser l'interface grâce à la méthode "initUI()" et une qui va charger la carte avec la méthode "onMapReady()". Dans notre méthode onCreate() nous allons appeler la méthode initUi() puis créer une variable "SupportMapFragment" pour appeler la méthode "onMapReady()" de manière asynchrone car c'est la procédure pour créer une carte.

Ensuite des méthodes permettront l'affichage de boîtes de dialogue créées dans un fichier XML pour pouvoir configurer la mission du drone et le parcours en appuyant sur des bouton grâce à l'implémentation de la méthode "OnClick()" et des méthodes "showSettingsDialog()" et "showSpaceSettingsDialog()" mais aussi des deux fichiers XML permettant de le visualiser sur l'application.

2.2.3 Implémentation de la mission du drone

Pour réaliser la mission à effectuer par le drone nous avons donc suivi le tutoriel qui nous a permis d'obtenir les méthodes de bases pour le contrôler. Cette partie se découpe en plusieurs sous parties :

- Localisation de l'appareil sur la carte
- Création des points des passages et du parcours
- Implémentation de la mission
- Charger, démarrer et stopper la mission
- Crédit d'un client UDP -> communication raspberry

2.2.4 Localisation de l'appareil sur la carte

Pour obtenir la localisation du drone un BroadcastReceiver doit être mis en place dans la méthode `onCreate()` de la `MainActivity` avec l'implémentation de la méthode `onReceive()`, cette méthode sera invoquée quand le statut de connexion du drone changera. On met alors en place des variables qui vont nous servir à stocker la position du drone, le marqueur et une instance de `flightController` qui nous servira à communiquer avec le drone et obtenir des informations sur les contrôleurs de vols.

Ensuite quand le statut de connexion du drone change c'est à dire de non connecté à connecté la méthode "`initFlightController()`" est appelée et va initialiser les éléments nécessaires comme les contrôleurs de vols (moteurs, capteurs, altitude..) du drone et obtenir la position initiale qui sera renseignée par la méthode `updateDroneLocation()` qui met à jour la position du drone.

Cette méthode est ensuite assigné à un bouton pour que l'utilisateur puisse dès qu'il le souhaite localiser l'appareil. Bouton qui est implémenté dans la méthode surchargée `onClick()` qui implémente les appels de méthodes voulu en fonction du bouton.

2.2.5 Crédit des points des passages et du parcours

Les points de passages ou Waypoint sont implementés par les méthodes `onMapClick()` et `markWaypoint()` par défaut mais pour notre projet ce n'est pas ce que nous voulions. Nous nous sommes donc servi de la méthode `markwayPoint()` pour placer sur la carte des marqueurs correspondant à la surface à traiter et qui sont insérés dans une liste de points (`ArrayList<Point>`) qui nous servira par la suite.

L'ajout de marqueurs est possible pour l'utilisateur en appuyant sur le bouton "ADD" plus haut en appuyant sur l'écran pour placer un marqueur.

Une fois le nombre de points souhaités l'utilisateur peut alors appuyer sur un bouton "GENP" qui va faire appel à la méthode `showSpaceSettingsDialog()` pour afficher une boîte de dialogue afin de renseigner l'espace qu'il veut entre les points et entre les points et le bords, une fois entrées ces paramètres sont envoyés à la méthode `genererParcours()`, méthode dans laquelle nous allons créer une instance de `Polygon` avec les points ainsi placés et la distance souhaitée pour générer les points sur la carte.

Ensuite dans cette méthode le parcours est réalisé en faisant appel aux différentes méthodes nécessaires au parcours expliquées dans la partie "Parcours" puis représenté visuellement sur la carte par des traits entre les marqueurs et des couleurs différentes pour trois marqueurs :

- vert pour le point de départ
- magenta pour le deuxième point
- jaune pour le point final

afin de vérifier manuellement le résultat, ils contiennent aussi une fenêtre permettant de savoir leur position dans le parcours.

Les marqueurs sont ajoutées sur la carte dans une boucle via la méthode `addMarker()` de

L'API GoogleMaps et reliés entre eux à la sortie de la boucle pour bien visualiser le parcours par une autre méthode de cette API.

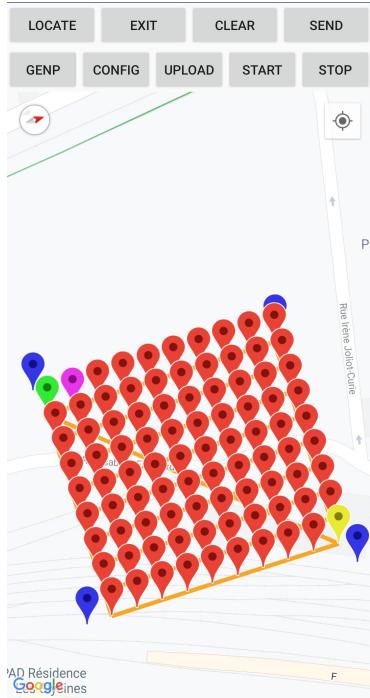


FIGURE 2.11 – Une capture d'écran du parcours

2.2.6 Implémentation de la mission

La mission est implémentée en prenant en paramètre le résultat de "genererParcours()" qui aura rempli une liste de points de passage nommée waypointList qui est une List de Waypoint. Pour construire le parcours une instance de waypointMission.Builder est créée et tiendra compte des ajout des points dans la liste de Waypoint et tout les paramètres à configurer pour la mission.

Ensuite l'utilisateur devra configurer la mission qu'il veut en appuyant sur le bouton "CONFIG" qui actionnera la méthode showSettingsDialog() qui affiche la boîte de dialogue permettant de paramétrier la mission.

Cette méthode sert d'entrée pour la méthode configWaypointMission() pour paramétrier la mission c'est à dire renseigner l'altitude désirée, la vitesse de l'appareil, l'action à réaliser par le drone après avoir fini la mission et choisir d'utiliser notre WayPoint List ou un autre parcours.

Après configuration la mission est "chargée" sur l'appareil grâce à une méthode qui déclera du succès ou non du chargement de la mission sur l'appareil.

La mission peut se déclencher grâce à deux boutons actionnant les méthodes startWaypointMission() et stopWaypointMission() qui sont les méthodes qui lancent et stoppent la mission.

Pendant la mission, l'application envoie des messages UDP au serveur pour pouvoir semer les graines.

2.2.7 Communication avec le Raspberry

Pour communiquer avec le Raspberry Pi via le Wi-Fi, la création d'un client UDP dans l'application Android a été nécessaire, nous avons donc réalisée une classe UDPClient qui correspond à un client UDP classique et qui contient une méthode sendInstruction() envoyant un caractère choisi auparavant pour déclencher une réaction après réception côté serveur.

Cette méthode nous sera utile pour envoyer une instruction au serveur dès qu'un point de passage sera atteint par le drone. On peut vérifier le bon fonctionnement du client UDP en appuyant sur le bouton "SEND" de notre interface ce qui actionnera le moteur si l'envoi de message à fonctionner.

2.3 Communication entre l'application et le moteur : relâchement des graines

2.3.1 Principe de base

Un serveur UDP est hébergé sur le raspberry (sur lequel est branché un servo moteur). Le serveur reçoit des messages du client (l'application mobile) et agit en conséquence sur le servo moteur afin de relâcher les graines.

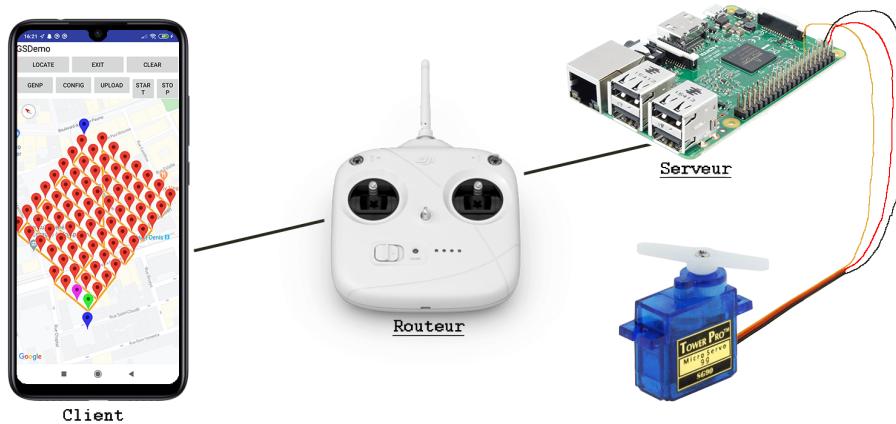


FIGURE 2.12 – Schéma du réseau utilisé

2.3.2 Établir la connexion

Afin de permettre une communication simple, il fallait mettre en place un serveur qui serait en écoute constante. Nous avons choisi d'utiliser le langage C pour créer ce serveur. De cette façon nous avons pu mettre en application directe les notions que nous avons apprise dans l'UE Réseaux HLIN611. Qui dit serveur dit réseau. Il fallait trouver un réseau sur lequel se brancher, mais pas un réseau basé sur un routeur classique comme une freebox, livebox ou autre, car quand on veut parsemer des graines dans un champ, on est rarement dans le rayon

de couverture d'une box internet. Par chance le drone fonctionne nativement avec une manette qui émet un signal wifi pour y connecter un smartphone. Nous avons donc trouvé notre réseau.

Ensuite, nous avions le choix entre deux protocoles de communications : TCP ou UDP ? Dans un premier temps nous avons mis en place un serveur TCP. Le protocole TCP est fiable car fonctionne avec connexion (accept) client/serveur. C'est justement cela qui engendre la problématique suivante : Comment agir si le drone s'éloigne trop et qu'il y a une perte de connexion ? La manière la plus simple de répondre à ça est de revenir sur le choix de protocole et de partir sur une nouvelle base en UDP.

Le protocole UDP ne nécessite aucune connexion, donc si le drone s'éloigne trop et qu'il y a perte de connexion au réseau, le serveur ne recevra plus les instructions momentanément mais dès que le drone reviendra assez proche, tout continuera normalement sans devoir refaire de connexion. De plus le message que nous avons choisi d'envoyer par l'application pour déclencher le moteur est l'entier 1. Ce message étant très petit, il y a peu de risque de perte de paquets, ainsi UDP est donc la solution qui semble la plus adaptée à notre projet.

Cette solution suffira dans le cadre de notre projet, mais présente tout de même des limites si la zone que nous voulons couvrir devient trop importante et va au delà de la zone de couverture wifi. Pour palier à cela, il existe des solutions de communications avec le raspberry utilisant le réseau téléphonique GSM ou LTE-M.

2.3.3 Contrôle du moteur avec le Raspberry

Lors du démarrage du serveur, la taille ainsi que le temps d'ouverture de la trappe, doivent être paramétrées. C'est le servo moteur, qui va s'occuper d'actionner la trappe. (cf [vidéo](#)) Pour manipuler ce servo, nous avons choisi d'utiliser le langage python, car c'est l'un des langages les plus utilisés et adaptés dans la programmation de raspberry/arduino. Le servo se connecte au raspberry grâce à 3 fils :

- Alimentation (5V) ici rouge
- Terre ici marron
- Modulation (gère l'info) ici orange

Nous allons gérer ce servo grâce à la technique PWM (Pulse Width Modulation) ou MLI en français (Modulation de Largeur d'Impulsions).

Comme dans la majorité des cas d'utilisation de servo, nous allons utiliser un signal de 50Hz. Cela correspond à 50 impulsions par seconde, soit une impulsion par 20ms. Les impulsions sont donc de 20ms chacune.

L'angle de l'hélice du servo est contrôlé en fonction du pourcentage du cycle pendant lequel le signal est ON. Si le signal est ON, alors le courant (5V) passe, et inversement si le signal est OFF.

D'après la documentation :

- signal ON pendant 0,5ms = angle 0°

— signal ON pendant 2,5ms = angle 180°

Ici, en python, la durée du cycle est calculée en pourcentage du cycle.

Exemple :

$$0^\circ = 0,5\text{ms} / 20\text{ms} * 100 = 2,5\%$$

$$180^\circ = 2,5\text{ms} / 20\text{ms} * 100 = 12,5\%$$

À partir de ces deux points, nous pouvons déterminer la fonction affine suivante, qui convertit les degrés en pourcentage de cycle :

$$f(x) = 0,55x + 10$$

Ainsi, nous pouvons moduler notre serveur, afin de relâcher plus ou moins de graines selon chaque convenance.

Chapitre 3

Bilan et difficultés rencontrées

3.1 Avancement du projet

Dans le chapitre précédent, nous avons vu la conception et l'implémentation du projet. Cette prochaine partie sera consacrée à l'avancement du projet. Nous savions que la charge de recherche et de documentation serait conséquente, ce que nous avons anticipé. En toute conscience, nous savions aussi que le travail à fournir en est de même, nous nous sommes donc organisés en conséquence.

Dès lors que certains outils furent installés correctement, que le cahier des charges fût validé et que les premiers fragments de codes furent fonctionnels, le projet a évolué rapidement. Nous avons essayé de répartir de la meilleure manière possible les différentes tâches à effectuer. En effet, nous avons réparti certaines tâches en fonction des compétences de chacun. Il est non négligeable que fournir un travail constant et régulier a permis la bonne et constante évolution du projet. Nous sommes content de l'avancement du projet aujourd'hui.

3.2 Difficultés rencontrées

Nous avons rencontré plusieurs difficultés durant le développement de ce projet que nous avons su surmonter. Dans un premier temps nous avons dû apprendre à maîtriser le drone, c'est-à-dire comprendre son mode de communication via la radiocommande et via le smartphone.

Ensuite, après avoir développé un prototype d'application, nous nous sommes confrontés aux problèmes algorithmiques du parcours. Plusieurs phases de débogage plus ou moins longues ont été nécessaires. Une fois cet aspect du projet terminé, nous avons établi une connexion entre l'application et le Raspberry Pi afin d'actionner le servomoteur quand celui-ci reçoit un message.

Mais notre principale difficulté a finalement été de réaliser ce projet avec la crise sanitaire qui a commencé en mars 2020. Cela nous a empêché d'avoir accès au drone qui appartient malheureusement à la faculté. C'est donc toute la phase de tests avec le drone qui n'a pas pu

être effectué. Nous avons alors ajouté quelques lignes de codes apportant un minimum de certitudes quant aux résultats souhaités.

3.3 Connaissances et apprentissage

Ce projet nous a permis d'approfondir les notions dans des domaines très variés. Nous avons développé une application Android. Les compétences requises ont été l'utilisation Android studio, qui est un nouvel IDE à prendre en main, ainsi qu'une nouvelle syntaxe et un nouveau "style" de programmation. Nous avons dû comprendre le fonctionnement d'une connexion d'un appareil indépendant à un autre. Nos compétences en graphes, réseaux et Java nous ont permis de comprendre et réaliser ce projet à bien.

Le travail sur le parcours nous ont fait replonger dans nos connaissances en algorithmique, mais aussi et surtout en théorie des graphes. Nous avons aimé travailler sur un problème parfaitement modélisé par des graphes. Nous avons choisis d'implémenter et résoudre ce problème et cela nous a beaucoup appris. Nous avons aussi appris à nous servir d'un kit de développement (SDK) afin de communiquer avec un objet connecté, le drone, ce qui était complètement nouveau pour nous et nous a permis de comprendre comment fonctionnent les communications avec un drone via une application Android.

Conclusion

L'objectif de ce projet était d'automatiser l'ensemencement de graines sur un terrain donné avec un drone et via une application smartphone.

Au terme de ce projet, deux applications sont réalisées, la première une application Android qui génère le plan de vol, envoi des messages à la deuxième pour lâcher des graines, communique avec le drone et le contrôle, la deuxième un serveur UDP qu'en réception d'un message, active un servomoteur et lâche des graines.

Les deux applications achèvent leurs objectifs, mais elles sont un peu rigides, nous aurions aimé améliorer leurs interfaces graphiques pour faciliter leur usage aux non informatiens et aussi faciliter les tests avec des différents paramètres. Malheureusement nous n'avons pas eu l'occasion de tester le plan de vol à cause de la crise sanitaire du COVID-19.

Le confinement nous a obligé à travailler d'une façon différente, nous étions habitués à travailler ensemble dans une même salle avec une communication directe, mais cela est devenu impossible avec la crise sanitaire, alors nous avons vécu les différents problèmes de communication et de gestion de version que se présentent dans les projets à distance. Nous avons donc approfondi nos connaissances sur des outils de gestion de version comme Git, et nous avons mieux compris l'importance des différents concepts informatique comme la documentation et les abstractions du code. Ces concepts sont souvent ignorés dans des petits projets avec un seul développeur, mais devient indispensable dès que le nombre de développeurs augmente et quand ce n'est pas possible d'avoir une communication directe.

Ce projet montre comment les différentes parties de l'informatique sont liées, d'une part le côté théorique utilisé pour créer les différents algorithmes, et d'une autre part le côté technique qui permet d'implémenter ces algorithmes avec les différents technologies disponibles. La dernière partie est la partie humaine, les informaticiens ne travaillent pas seuls, donc une compréhension des humaines est nécessaire pour avoir une bonne gestion du projet et pouvoir le conclure.

Annexes

Liens vers le code source

- GitLab UM : <https://gitlab.info-ufr.univ-montp2.fr/e20170006328/TER-L3-Drone/tree/master>
- GitHub : <https://github.com/LeifHenriksen/TER-L3-Drone>

Lien vers la vidéo de démonstration du servomoteur

Lien Youtube : <https://youtu.be/9ukoLNAadPg>

```
95     //Kruskal -> Arbre couvrant de poids minimum
96     public ArrayList<Point> kruskal(int n, int m) {
97         System.out.println("-----Debut kruskal-----");
98         //Arbre couvrant de poids minimum (acpm) -> résultat en sortie
99         ArrayList<Point> acpm = new ArrayList<>();
100
101        //Le tableau "comp" vérifiant comp(x) = comp(y) si et seulement si il
102        //existe un chemin de x à y (ou autrement dit si et seulement si x et y
103        //appartiennent à la même composante connexe de G).
104        HashMap <Point, Integer> comp = new HashMap<>();
105        for(int i=0;i<listePoint.size();i++ ) {
106            comp.put(listePoint.get(i),i);
107        }
108
109        Integer aux=0;
110        for(int i = 0; i < edge.size(); i++) {
111            if(!(comp.get(edge.get(i).p1).equals(comp.get((edge.get(i).p2))))) {
112                aux = comp.get(edge.get(i).p1);           //comp(x)
113                acpm.add(edge.get(i).p1);                  //A u {xy}
114                acpm.add(edge.get(i).p2);
115            }
116            for(int k= 0 ; k< listePoint.size();k++) {
117                if(comp.get(listePoint.get(k)).equals(aux)) {
118                    comp.put(listePoint.get(k),comp.get(edge.get(i).p2));
119                }
120            }
121        }
122        System.out.println("-----Fin kruskal-----");
123        return acpm;
124    }
```

FIGURE 3.1 – Extrait de code : l'algorithme de Kruskal

```

126● public ArrayList<Point> parcours(ArrayList<Point> kruskal) {
127     System.out.println("-----Debut Parcours-----");
128     ArrayList<Point> parcours = new ArrayList<Point>(); //Resultat retourné
129     HashMap<Point,ArrayList<Point>> voisins = initVoisins(kruskal); //HashMap : à chaque point sa liste de voisins
130     HashMap<Point,Integer> dv = new HashMap<>(); //Tampon pour savoir quels points ont été traités
131     for(int i = 0; i < listePoint.size(); i++) { //Sommets déjà vu : dv initialisé à 0
132         dv.put(listePoint.get(i),0);
133     }
134     HashMap <Point,Point> pere = new HashMap<>(); //Stockage par paire : un point et son père
135     Point racine = listePoint.get(0); //premier point = la racine
136     dv.put(racine,1); //On initialise la racine
137     pere.put(racine,racine); //Le père de la racine est elle même
138     Stack<Point> AT = new Stack<>(); //Pile AT (à traiter)
139     AT.push(racine);
140     while(!AT.isEmpty()) { //tant que AT n'est pas vide
141         Point x = AT.peek();
142         if(voisins.get(x).isEmpty()) {
143             AT.pop();
144         } else {
145             Point y = voisins.get(x).get(voisins.get(x).size()-1); //On traite le sommet en haut de voisins et on le dépile
146             if(dv.get(y) == 0) { //Déjà vu de y noté à 1
147                 dv.put(y,1); //On traite y pour la première fois
148                 AT.push(y);
149                 pere.put(y,x);
150                 parcours.add(x); //Ajout de l'arête au parcours
151                 parcours.add(y);
152             }
153             if(voisins.get(x).isEmpty()) {
154                 Point leplusproche = new Point(9999999,9999999,0,0,0);
155                 for (int i = 0; i < listePoint.size(); i++) {
156                     if( (!x.equals(listePoint.get(i))) && dv.get(listePoint.get(i))==0 ) {
157                         if(Point.distance(x, listePoint.get(i))<Point.distance(x, leplusproche)) {
158                             leplusproche= listePoint.get(i);
159                         }
160                     }
161                 }
162                 if(!leplusproche.equals(new Point(9999999,9999999,0,0,0))) {
163                     voisins.get(x).add(leplusproche);
164                 }
165             }
166         }
167     }
168 }
169 System.out.println("-----Fin Parcours-----");
170 return parcours;
171 }

```

FIGURE 3.2 – Extrait de code : l'algorithme parcours (parcours en profondeur modifié)

```

public class UDPClient {
    //On envoie une instruction au raspberry afin de lacher des graines
    //On utilise un protocole UDP
    public void sendInstruction() {
        new Thread(){
            @Override
            public void run(){
                try {
                    DatagramSocket UdpSocket = new DatagramSocket( port: 23600);
                    InetAddress serverAddr = InetAddress.getByName("61.169.243.191");
                    byte[] buf = new byte[1];
                    buf[0] = 1;
                    DatagramPacket packet = new DatagramPacket(buf, buf.length, serverAddr, port: 23600);
                    UdpSocket.send(packet);
                } catch(SocketException e) {
                    e.printStackTrace();
                } catch (IOException e){
                    e.printStackTrace();
                }
            }
        }.start();
    }
}

```

FIGURE 3.3 – Le client UDP

```

private void genererParcours(GoogleMap gMap, int s, int z) {
    if (gMap == null) {
        gMap = googleMap;
        setUpMap();
    }

    com.dji.GSDemo.GoogleMap.Polygon p = new com.dji.GSDemo.GoogleMap(pointsFinal,s,z);
    Chemin chemin = new Chemin();
    ArrayList<Point> parcours = chemin.voyageurDeCommerce(p);

    parcours.add(parcours.get(parcours.size()-1));
    parcours.add(parcours.get(0));
    for(int i = 0; i < parcours.size(); i += 2)
    {
        parcoursFinal.add(parcours.get(i));
        PolylineOptions polylines = new PolylineOptions();

        LatLng from = new LatLng(parcours.get(i).getLat(), parcours.get(i).getLng());
        LatLng to = new LatLng(parcours.get(i+1).getLat(), parcours.get(i+1).getLng());
        polylines.add(from, to).color(0xffff9a025).width(12);

        gMap.addPolyline(polylines);
    }

    for (int i = 0; i < parcoursFinal.size(); i++) {
//System.out.println(p.getPointsInternes().get(i));
        if(i==0) {
            gMap.addMarker(new MarkerOptions()
                .position(new LatLng(parcoursFinal.get(i).getLat(), parcoursFinal.get(i).getLng()))
                .title("Point n° " + i)
                .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN)));
        }
        else if(i==1) {
            gMap.addMarker(new MarkerOptions()
                .position(new LatLng(parcoursFinal.get(i).getLat(), parcoursFinal.get(i).getLng()))
                .title("Point n° " + i)
                .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_MAGENTA)));
        }
        else if(i==parcoursFinal.size()-1) {
            gMap.addMarker(new MarkerOptions()
                .position(new LatLng(parcoursFinal.get(i).getLat(), parcoursFinal.get(i).getLng())));
        }
    }
}

```

FIGURE 3.4 – Extrait de la méthode générerParcous

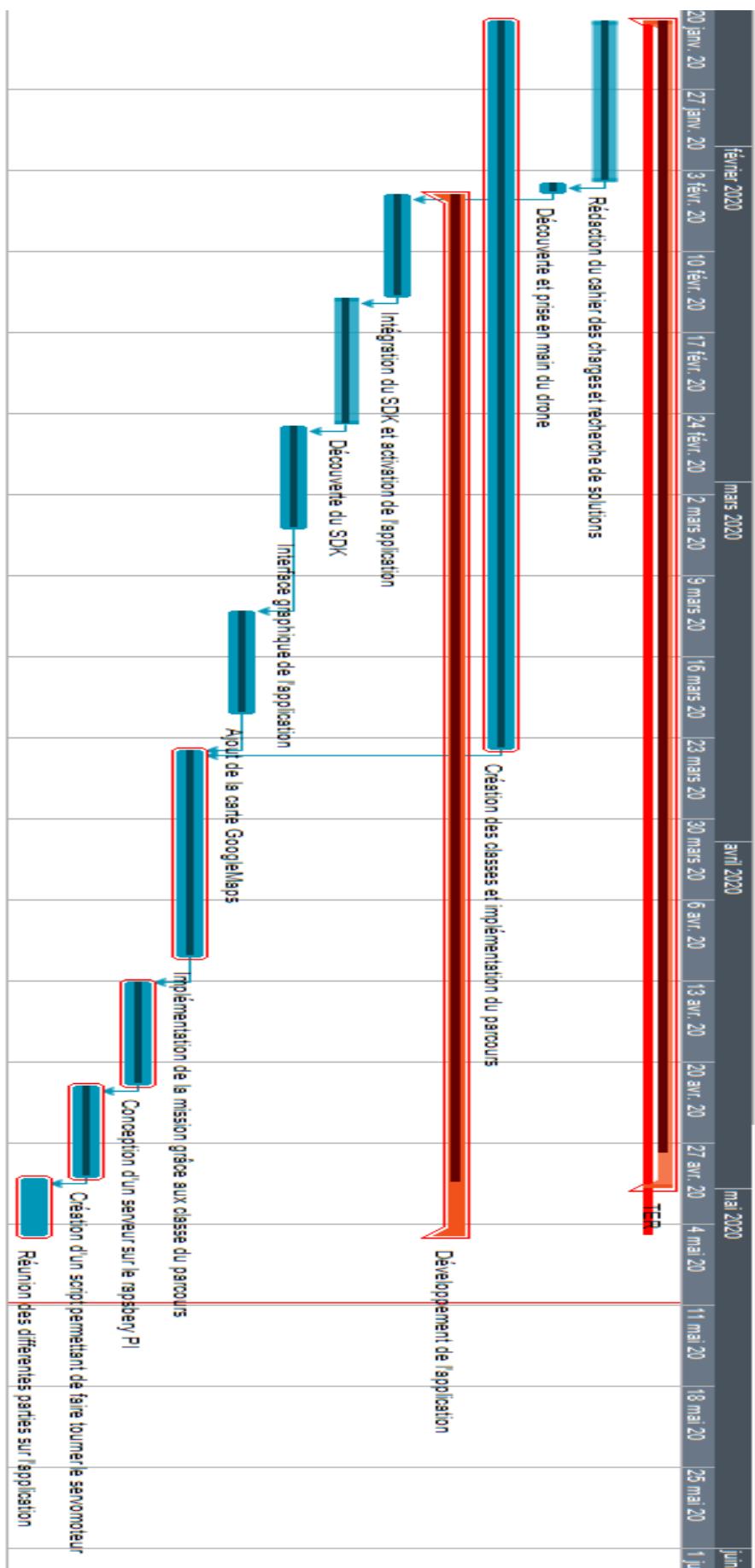


FIGURE 3.5 – Diagramme de Gantt

Table des figures

1	Exemple d'un drone de relâchement de graines	2
1.1	Logos des outils de travail utilisés	7
1.2	Dji Phantom 3 SE et sa radiocommande	7
1.3	SG90 Mini Servo Moteur	8
1.4	Raspberry Pi 3 Model B et son boitier	8
1.5	Xiaomi Redmi Note 7	8
2.1	Génération des points de passage	9
2.2	Calcul des intersections	10
2.3	Exemple de graphe complet (K8)	11
2.4	Résultat obtenu après Kruskal	12
2.5	Résultat obtenu après le parcours en profondeur, Le drone devra suivre ce parcours	14
2.6	Voyageur de Commerce (Java)	15
2.7	Résultat du voyageur de commerce	16
2.8	Capture d'écran de la structure du projet	17
2.9	Schéma de l'activation de l'application	18
2.10	Une capture d'écran de l'interface utilisateur	19
2.11	Une capture d'écran du parcours	21
2.12	Schéma du réseau utilisé	22
3.1	Extrait de code : l'algorithme de Kruskal	28
3.2	Extrait de code : l'algorithme parcours (parcours en profondeur modifié)	29
3.3	Le client UDP	29
3.4	Extrait de la méthode générerParcous	30
3.5	Diagramme de Gantt	31

Bibliographie

- Algorithme Voyageur de commerce : *Cours HLIN501*
- Client-Serveur UDP : *Cours HLIN611*
- Algorithme Point dans polygone : <http://alienryderflex.com/polygon/>
- Formules GPS : <https://wiki.openstreetmap.org/wiki/Mercator>
- Module GPIO : <https://raspi.tv/2013/rpi-gpio-basics-4-setting-up-rpi-gpio>
- Gestion Servo : <http://electroniqueamateur.blogspot.com/2015/11/controler-un.html>
- Documentation DJI : <https://developer.dji.com/mobile-sdk/documentation/introduction/index.html>
- Documentation GoogleMaps : <https://developers.google.com/maps/documentation/android-sdk/intro>
- Documentation Android : <https://developer.android.com/guide>