



Projet d'Option

- Optimisation de trajectoire de drone à l'aide d'algorithmes génétiques -

Rapport Projet Final

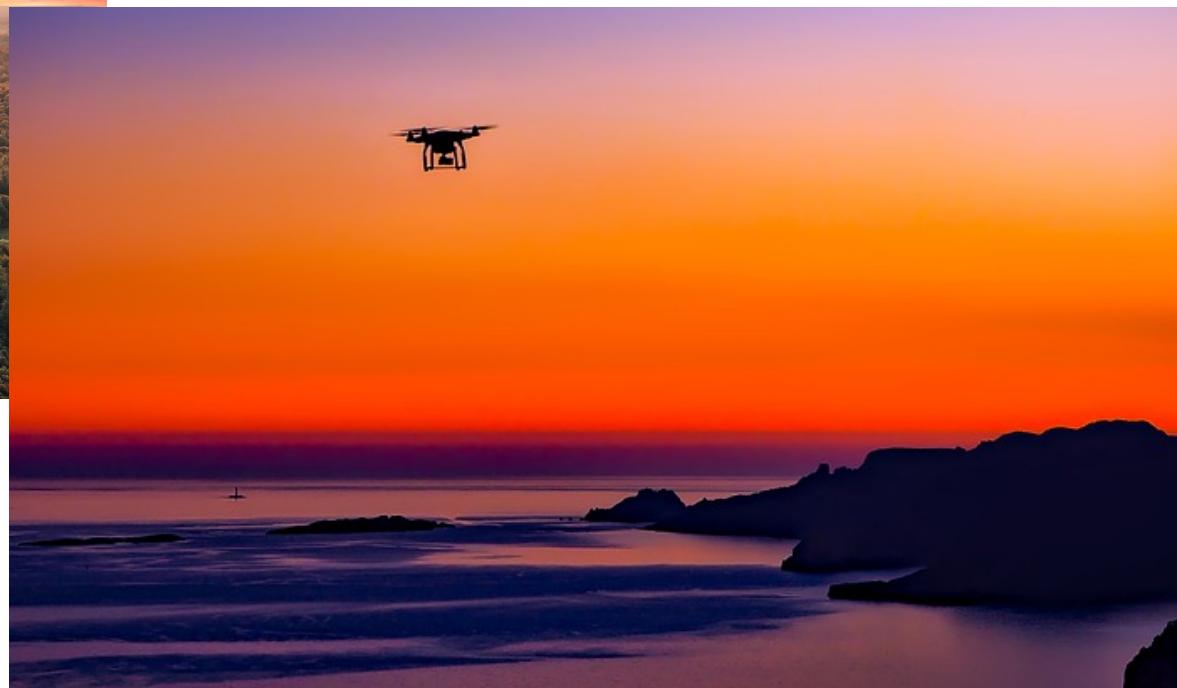
Massart - Tord

Option Aéronautique

Janvier 2021

SOMMAIRE

Introduction	2
1. Analyse Bibliographique	3
2. Optimisation de la trajectoire en 2D	6
3. Approfondissement : adaptation à tout relief	27
Conclusion	32
Bibliographie	33



Introduction

Il est parfois nécessaire de conférer un caractère furtif au vol d'un drone. Ce caractère furtif peut être, en partie, obtenu par une adaptation de la structure du drone. Cependant, il est souvent plus efficace d'utiliser le relief de la zone survolée pour « masquer » sa trajectoire. La difficulté de cette opération est alors de concilier les performances limitées du drone avec les caractéristiques topologiques du relief.

Un travail consistant à optimiser la trajectoire d'un drone modèle à l'aide d'« automates cellulaires » a été déjà accompli et a permis d'obtenir les premiers résultats probants ; cependant cet algorithme ne permet pas de calculer une trajectoire pour laquelle de nombreux paramètres interviendraient.

Le but de ce travail est donc, à partir de la mise en évidence des principaux paramètres caractéristiques du vol d'un drone et du travail déjà effectué, d'utiliser tous ces paramètres pour générer la trajectoire « idéale », à l'aide d'un algorithme génétique.



1. La notion d'algorithme génétique

Notre projet s'inscrit dans la continuité du projet réalisé par Nan ZHANG et Manon DREUX, eux-mêmes s'étant inspirés de celui de Matthieu PARIS et Eliot GIRAUD.

Les étudiants ont débuté la programmation en 2D de la trajectoire du drone afin qu'il soit le plus proche possible du relief. Ils ont en particulier permis de stocker le relief sous la forme d'une liste de points, et ont utilisé un algorithme récursif ainsi que des listes chaînées pour créer la trajectoire. De nombreux paramètres ont été pris en compte. Le dernier groupe a généralisé ces algorithmes au 3D, et les résultats sont plutôt probants.

Cependant, à la lecture de leur rapport, une des faiblesses de leur approche est l'imposition directe des contraintes par l'homme. C'est pourquoi l'utilisation de machine learning pourrait permettre d'obtenir des trajectoires encore plus pertinentes, étant donné que l'ordinateur, par l'auto-apprentissage de ses erreurs, développerait lui-même un panel de contraintes beaucoup plus poussées. Nous nous sommes tournés vers les algorithmes génétiques, qui nous semblent être les plus adaptés à notre problème.

Nous avons fait le choix de prendre le même drone que celui proposé par le groupe précédent: le *RQ7B–shadow 200*. Celui-ci possède les caractéristiques suivantes :



Vitesse de croisière	130 km/h
Vitesse minimale	75 km/h
Vitesse maximale	204 km/h
Angle de décrochage	20°
Rayon d'action	10m

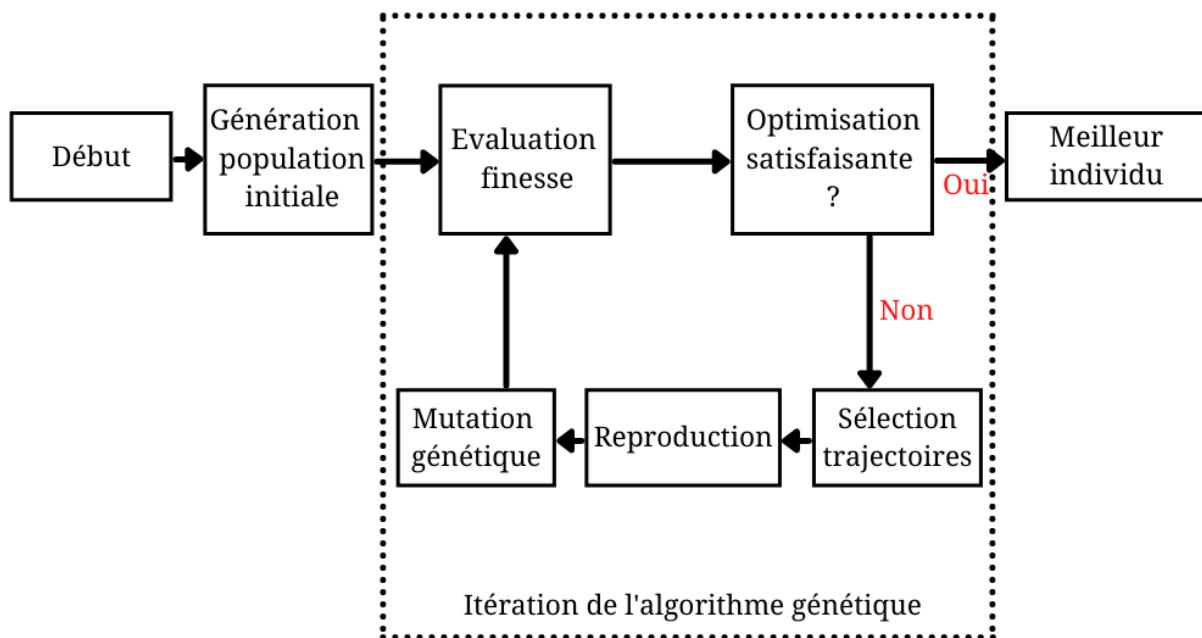
Pour comprendre notre projet, parlons tout d'abord des algorithmes génétiques, l'essence de notre projet.

La notion d'algorithme génétique traduit une méthode d'obtention d'une solution approchée à un problème d'optimisation, en s'inspirant de la sélection naturelle.

Initialement, il y a une population de solutions candidates appelées parfois individus, créatures, phénotypes qui va évoluer de génération en génération jusqu'à la génération qui contient les meilleures solutions. Ici nous prenons comme population les possibles trajectoires construites aléatoirement.

Chaque individu comprend des propriétés et il peut être sujet à des transformations génétiques (mutation, croisement par exemple).

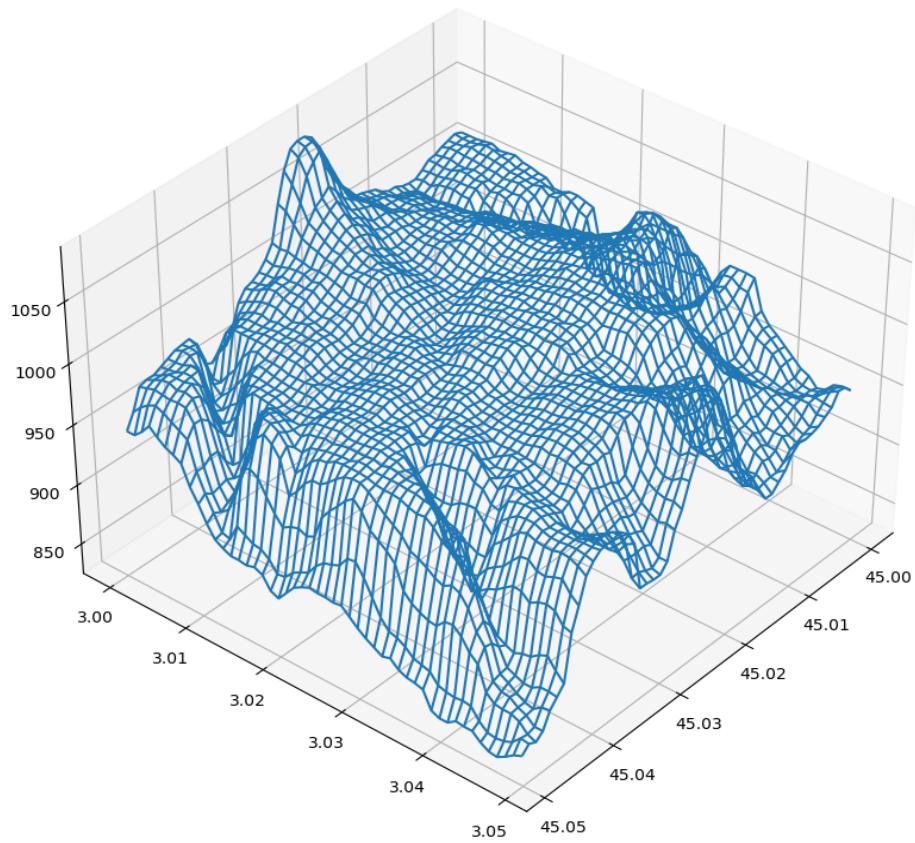
Chaque individu est évalué et cette valeur d'aptitude (appelée finesse) est un critère pour sa survie d'une génération à une autre. Dans notre cas, la finesse sera calculée avec la furtivité du drone, sa rapidité, sa précision, etc...



Représentation schématique de l'architecture d'un algorithme génétique

Nous commencerons par une approche 2D pour maîtriser cette méthode et commencer à penser son adaptation aux trajectoires. Ensuite l'objectif est de mettre en œuvre cette méthode en 3D.

Si le temps nous est suffisant, nous désirons aussi être en mesure de créer un relief à partir des coordonnées GPS de départ et d'arrivée du drone. Cela nous permettrait de tester nos trajectoires en conditions réelles.



Exemple de relief 3D à étudier

2. Optimisation de la trajectoire en 2D

Comme évoqué précédemment, l'objectif de cette partie est d'optimiser la furtivité de la trajectoire du drone en 2D.

Création du relief

Nous travaillons tout d'abord sur la gestion des reliefs afin de pouvoir les utiliser lors du développement de notre algorithme génétique.

Afin de pouvoir comparer nos résultats à ceux des années précédentes, nous reprenons le relief d'étude utilisé dont les coordonnées des points en surface sont stockées dans un fichier .txt comme ci-dessous :

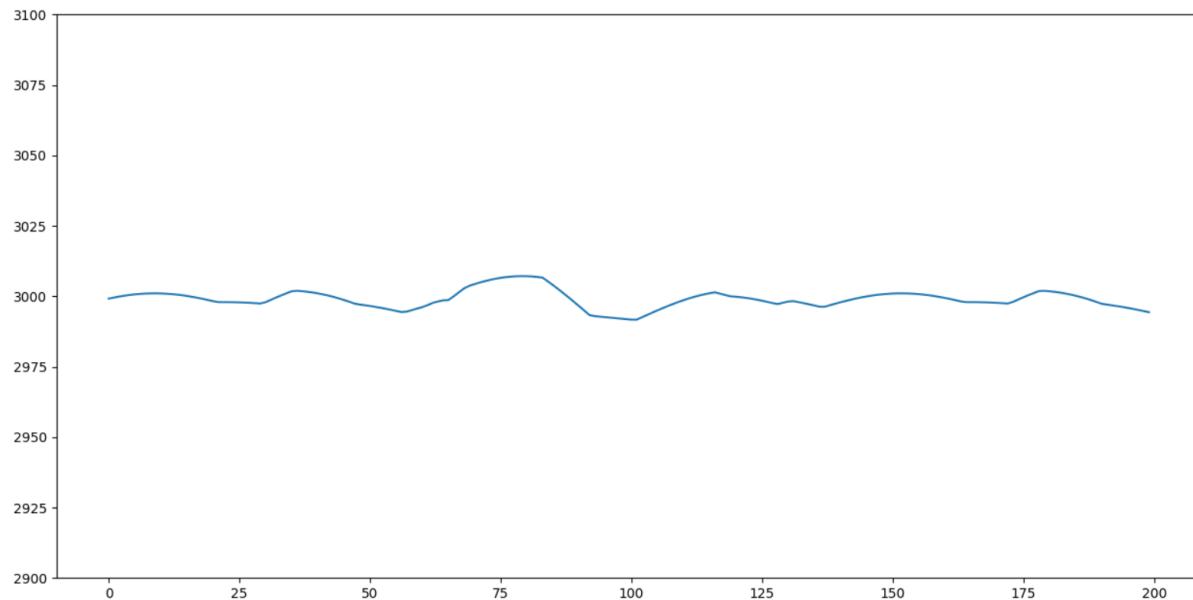
1	2999.55	11	3000.91
2	2999.9	12	3000.78
3	3000.2	13	3000.6
4	3000.46	14	3000.38
5	3000.68	15	3000.11
6	3000.84	16	2999.8
7	3000.96	17	2999.46
8	3001.02	18	2999.08
9	3001.04	19	2998.68
10	3001	20	2998.25

Nous créons une classe *gestionnaireRelief()* dans laquelle nous stockons le relief d'étude et définissons des méthodes utiles pour la suite de l'étude :

Nom	Type	Fonction
reliefChoisi	Liste des points	Conservation des coordonnées du relief
lireRelief(fichier)	Fonction	Lecture d'un fichier .txt et stockage dans reliefChoisi
longueurRelief()	Fonction	Retourne la longueur du relief
getAltitude(longitude)	Fonction	Retourne l'altitude d'un point connaissant sa longitude

Ainsi, ces outils seront par la suite utilisés dans les algorithmes génétiques. Comme ces fonctions sont simples et ont déjà été rédigées en C++ lors des années précédentes, nous ne rentrerons pas dans le détail de leur construction.

Voici le tracé du relief choisi pour l'élaboration de nos algorithmes :



Graphe du relief

A ce stade, l'algorithme que nous développons n'est capable de fonctionner que lorsque l'utilisateur a créé en amont un fichier .txt contenant le relief d'étude, ce qui reste assez contraignant. Nous verrons dans la partie 3 comment pallier ce problème et simplifier l'expérience utilisateur de notre algorithme final.

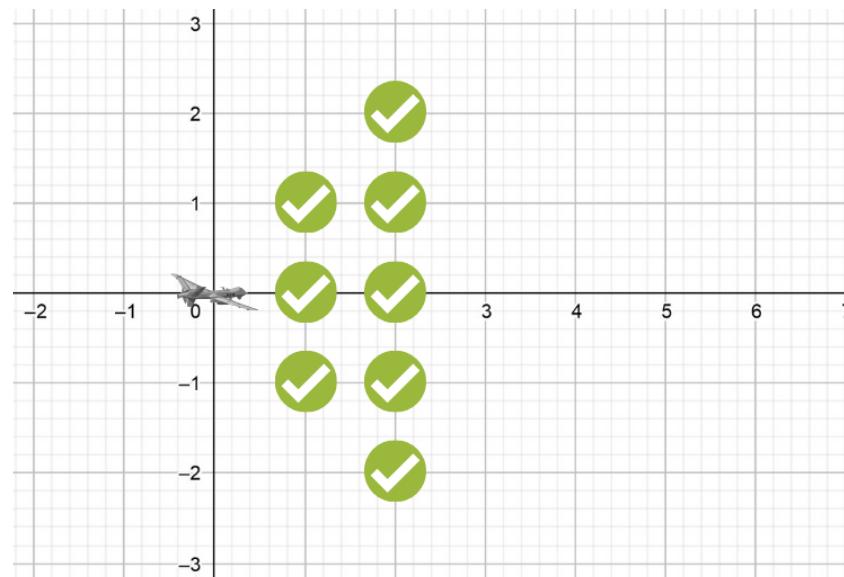
Discrétisation de l'étude

_____ Nous discrétisons le problème de la même manière que les projets précédents, en choisissant dans un premier temps un pas d'un mètre. Nous envisageons par la suite d'affiner cette discrétisation (cf. Partie 3).

Ainsi, les coordonnées des points du relief et de la trajectoire (cf. sous partie suivante) seront calculées tous les mètres. C'est d'ailleurs pour cette raison que le fichier .txt contient les altitudes des points à chaque mètre.

Trajectoire du drone

Ensute, intéressons-nous à la définition de la trajectoire du drone. Pour respecter les capacités du drone ainsi que la discréétisation du problème, voici l'ensemble des positions pouvant être prises par le drone (cf. calculs réalisés les années précédentes) :



Représentation graphique des positions possibles prises par le drone à chaque pas de temps

Sous python, voici la liste des déplacements possibles par le drone à chaque instant de la discréétisation :

```
listeDeplacementsPossibles = [[1,1], [1,0], [1,-1],  
[2,2],[2,1],[2,0], [2,-1], [2,-2]]
```

Numériquement, on crée une classe *trajectoire()* pour gérer les trajectoires du drone. Voici l'ensemble des composants de cette classe :

Attributs de la classe <i>trajectoire()</i>			
Nom	Type	Description	Initialisation
trajectoire	Liste	Liste des points de la trajectoire du drone	[None,None,...,None]
deplacements	Liste	Liste des déplacements choisis tout le long de la trajectoire	[]

GestionnaireRelief	gestionnaire Relief	Relief d'étude, longueur, etc.	Argument d'entrée
finesse	flottant	Critère de notation de la trajectoire (cf. sous partie finesse)	0
distanceParcourue	flottant	Distance totale parcourue par le drone lors de la trajectoire	0
vitesseMoyenne	flottant	Vitesse moyenne du drone lors du parcours de la trajectoire	0
tempsMission	flottant	Temps de parcours	0
pointInitial	doublet	Point de départ du drone	Argument d'entrée
pointFinal	doublet	Point d'arrivée du drone	Argument d'entrée
hauteurSecurite	flottant	Hauteur minimale entre le drone et le relief à respecter en chaque point	Argument d'entrée

Fonctions/Méthodes de la classe <i>trajectoire()</i>			
Nom	Description	Entrées	Sorties
genererDeplacements()	Créer la liste des déplacements choisis au hasard du point de départ au point d'arrivée	-	-
miseAJourTrajectoire()	Créer la trajectoire à partir d'une liste <i>deplacements</i> choisis par le drone	deplacements	-
genererIndividu()	Utilise genererDeplacements() et miseAJourTrajectoire() pour créer une nouvelle trajectoire	-	-
tailleTrajectoire()	Calcule la taille de la trajectoire (ex : si le drone se crash, la taille est inférieure à la distance départ-arrivée)	-	taille

Voici l'ensemble des outils créés par la classe *trajectoire()*. Nous ajouterons par la suite d'autres méthodes, notamment pour le calcul de la finesse. Revenons sur trois fonctions en particulier, qui sont plus complexes et demandent davantage de réflexion que les autres.

Tout d'abord, nous aurions pu travailler uniquement sur les trajectoires, mais nous nous sommes rendus compte que l'algorithme génétique ne pouvait s'appliquer qu'aux déplacements et non pas aux trajectoires. C'est pour cette raison que nous travaillons directement avec les déplacements choisis par le drone, et définissons la fonction *genererDeplacements()*. En voici son fonctionnement :

Fonction : genererDeplacements()	
<u>Objectif</u> : Créer la liste <i>déplacements</i> de la classe trajectoire de manière aléatoire	
<u>Entrées</u> : - self	<u>Sorties</u> : - self
<u>Algorithme</u> :	
si la hauteur initiale respecte la hauteur de sécurité faire : tant que l'indice positionX est inférieur à la longueur du relief faire : Le drone prend une trajectoire aléatoire dans la liste des déplacements possibles Ajout de ce déplacement dans la liste des déplacements de la trajectoire positionX += 1 fin tant que fin si	

Ensuite, il faut convertir ces déplacements en trajectoire du drone, c'est pourquoi nous définissons la fonction *miseAJourTrajectoire()* qui prend en argument la liste des déplacements générés au hasard par la fonction *genererDeplacements()*, et remplit la liste des points de la trajectoire du drone. Voici l'algorithme développé :

Fonction : miseAJourTrajectoire()

Objectif : Remplir la liste trajectoire de la classe trajectoire à partir des déplacements

Entrées :

- self
- déplacements

Sorties :

- self

Algorithme :

k=0

indicePositionX = 1

tant que l'indice k est inférieur à la longueur de la liste *déplacements faire* :

si le k-ième déplacement est d'1 mètre horizontalement **faire** :

 On remplace la indicePositionX-ème valeur de la trajectoire par le point précédent + le k-ième déplacement

si la coordonnée verticale du nouveau point de la trajectoire **ne respecte pas** la hauteur de sécurité **faire** :

 On provoque la fin de la boucle itérative puisque le drone s'est écrasé sur le relief

fin si

fin si

si le k-ième déplacement est de 2 mètres horizontalement **faire** :

 On remplace la indicePositionX-ème valeur de la trajectoire par le point précédent + la moitié du k-ième déplacement

 On remplace la (indicePositionX + 1)-ème valeur de la trajectoire par le point précédent + le k-ième déplacement

si la coordonnée verticale du nouveau point de la trajectoire **ne respecte pas** la hauteur de sécurité **faire** :

 On provoque la fin de la boucle itérative puisque le drone s'est écrasé sur le relief

fin si

fin si

fin tant que

Finalement, la fonction genererIndividu() combine les deux fonctions précédemment détaillées pour créer une trajectoire aléatoire.

A partir de cette étape, nous sommes capables de tracer une trajectoire et son relief associé. Plaçons nous par exemple dans ces conditions :

Nom du critère	Valeur
Point de départ	[longitude = 0 m, altitude = 3030 m]
Point d'arrivée souhaité	[longitude = 199 m, altitude = 3010 m]
Hauteur de sécurité	10 m

Traçons la trajectoire obtenue :

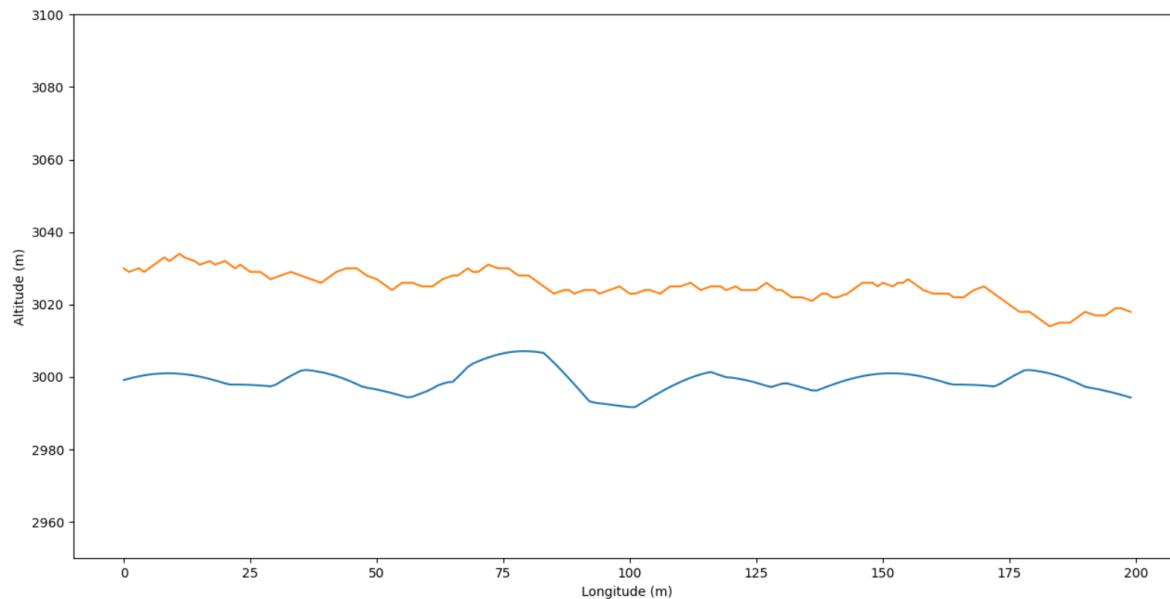


Fig. Tracée de la première trajectoire du drone

Dans ce cas, nous remarquons que le drone respecte en tout point la condition de hauteur de sécurité, il ne s'écrase donc pas au sol avant d'atteindre la distance désirée. Par contre, on remarque que la trajectoire n'arrive pas au point final souhaité. Il s'agit d'un problème que nous réglerons dans la suite de l'étude.

Finesse : premier calcul

_____ Comme expliqué précédemment, notre algorithme génétique consiste à faire évoluer une population de trajectoires (créées aléatoirement) sur plusieurs générations en les faisant se reproduire entre elles, dans le but de les optimiser pour converger vers la meilleure trajectoire possible, en s'inspirant du modèle de la sélection naturelle.

Pour cela, il faut avant tout être en mesure d'évaluer la qualité de chacune des trajectoires pour les comparer entre elles. On crée une fonction *finesse()* qui permet de calculer la finesse de

chaque trajectoire, et fonctionne comme un système de notation. Le but de notre algorithme génétique est donc d'obtenir la finesse la plus grande possible.

L'objectif de cette sous-partie de l'étude est donc de créer cette fonction *finesse()*. Notre problème est dit multi-critère, c'est-à-dire qu'il dépend de plusieurs paramètres. En effet, la qualité d'une trajectoire dépend de trois critères :

- Le respect de la hauteur de sécurité (i.e ne pas se crasher)
- La furtivité
- La rapidité

La difficulté réside ainsi dans la recherche d'une formule permettant d'obtenir un bon compromis entre ces 3 critères.

Pour simplifier le problème dans une première partie, nous avons décidé de ne pas prendre en compte le critère de rapidité, et de ne se concentrer que sur les deux autres critères, plus importants pour notre étude.

Tout d'abord, on rédige une fonction qui permet de calculer l'erreur d'altitude moyenne :

Fonction : erreurAltitude()	
<u>Objectif</u> : Calcul de l'erreur d'altitude de la trajectoire	
<u>Entrées</u> : - self	<u>Sorties</u> : - erreurAltitude
<u>Algorithme</u> (recherche de maximum classique):	
erreurAltitude = 0 Pour i allant de 0 à longueur trajectoire -1 faire : erreurAltitude += (différence de hauteur entre le drone et le relief au i-ème point) ² erreurAltitude = erreurAltitude/(longueur relief) Retourner(erreurAltitude)	

Ensuite, on peut calculer la finesse avec l'algorithme *finesse()* :

Fonction : finesse()

Objectif : Calcul de la finesse (fonction d'adaptation) de la trajectoire

Entrées :

- self

Sorties :

- finesse

Algorithme (recherche de maximum classique):

- Calcul de l'erreur d'altitude
- Mise à jour de l'erreur d'altitude dans la classe trajectoire

si le drone s'écrase au sol **alors** :

$$\text{finesse} = 0$$

Sinon faire :

$$\text{finesse} = (\text{hauteur de sécurité}) / (\text{erreur d'altitude})$$

Fin si

- Retourner(finesse)

Population

La deuxième étape de l'algorithme génétique consiste à créer une population d'individus :

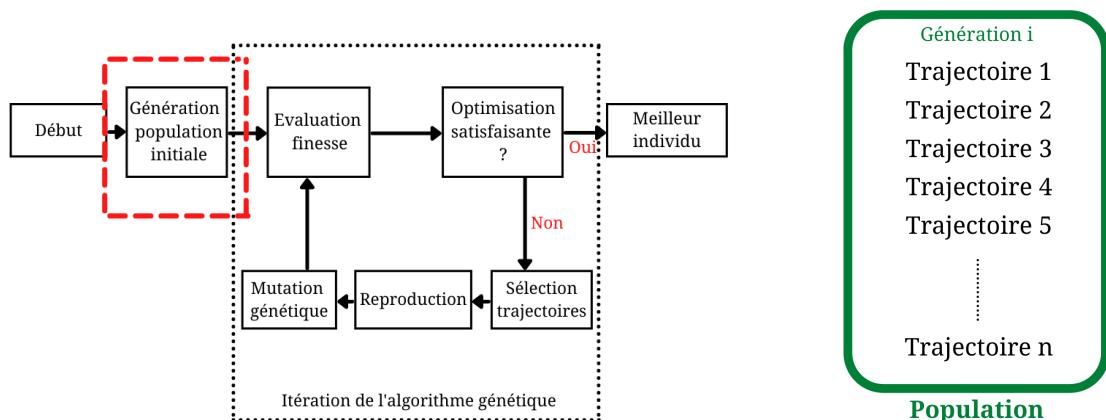


Schéma du principe de l'algorithme génétique

Représentation d'une population

Pour se faire, nous créons une classe *population()* dans le but de stocker ces populations et certains outils pour les manipuler. Voici ses composantes :

Arguments d'entrée de la classe <i>Population()</i>	
Nom de l'argument	Description
GestionnaireRelief	Ensemble des informations du relief d'étude
taillePopulation	Taille de la population à créer
init	Booléen

Attributs de la classe <i>Population()</i>			
Nom	Type	Description	Initialisation
trajectoires	Liste de trajectoires	Stocke les trajectoires de la population	- Si init == True : [None,..., None] - Sinon on initialise la population avec des trajectoires aléatoires

Fonctions/Méthodes de la classe <i>Population()</i>			
Nom	Description	Entrées	Sorties
taillePopulation()	Retourne la taille de la population de trajectoires	-	-
getTrajectoire()	Retourne la indice-ième trajectoire de la population	- Indice	trajectoire
sauvegarderTrajectoire()	Ajoute la trajectoire d'entrée à la indice-ième case de la population	- Trajectoire - Indice	-
getMeilleureTrajectoire()	Retourne la trajectoire avec la finesse la plus élevée de la population	-	trajectoire

La plupart de ces outils ne présentent pas de difficultés et ne nécessitent pas d'explication particulière. Revenons tout de même sur la fonction *getMeilleureTrajectoire()* en expliquant son fonctionnement :

Fonction : getMeilleureTrajectoire()

Objectif : Obtenir la trajectoire avec la meilleure finesse de la population

Entrées :

- self

Sorties :

- meilleureTrajectoire (trajectoire)

Algorithme (recherche de maximum classique):

La meilleure trajectoire initiale est la première trajectoire de la population

pour i allant de 1 à la taille de la population-1 :

si la finesse de la i-ème trajectoire est supérieure à la finesse de la meilleure trajectoire actuelle **alors** :

 La meilleure trajectoire actuelle devient la i-ème trajectoire

fin si

fin pour

On retourne la meilleure trajectoire à la fin de la boucle itérative.

Nous avons donc créé tous les outils nécessaires pour rédiger notre algorithme génétique en 2D. Rentrons désormais dans le vif du sujet.

Principe de l'algorithme génétique appliqué au problème

S'inspirant du principe de sélection naturelle, le processus de l'algorithme génétique passe par plusieurs étapes encadrées en rouge ci-dessous :

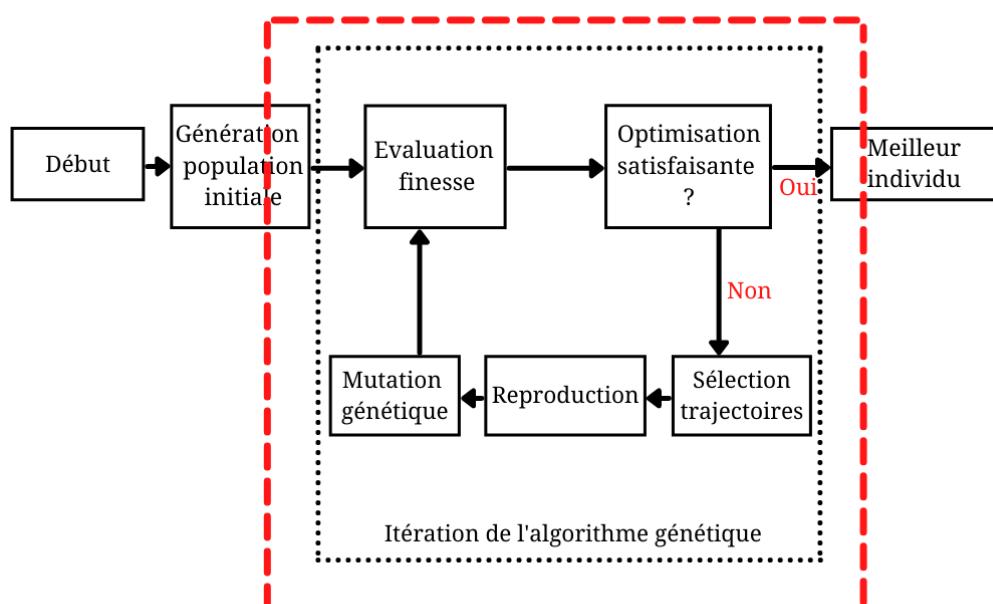


Schéma de fonctionnement d'un algorithme génétique

Une fois la population de trajectoires créée, il s'agit de lui appliquer de manière itérative notre algorithme génétique, jusqu'à convergence des finesse de la population, ce qui passe par :

- **La sélection** : les trajectoires de la génération précédente sont sélectionnées ou non pour créer la génération suivante, en fonction de leur finesse. Plusieurs méthodes de sélection existent, nous les étudierons plus en détail par la suite.
- **La reproduction (ou crossover)** : On choisit au hasard deux "parents" de ces trajectoires sélectionnées, on les "fusionne" pour ainsi créer la trajectoire "enfant", nouvel individu de la génération suivante. On répète ce processus jusqu'à la création complète de la nouvelle population.
- **La mutation génétique** : Pour assurer la diversité génétique des populations, les trajectoires de la nouvelle génération subissent ensuite des mutations génétiques avec une probabilité à choisir judicieusement (généralement de l'ordre de 0.01).

Afin de rédiger un algorithme génétique efficace pour cette optimisation, il doit être en mesure répondre aux mieux à certaines contraintes :

Cahier des charges/Exigences de l'algorithme génétique		
Nom de la contrainte	Description	Arguments/Fonctions concernés
Sélection valorisante	Les trajectoires doivent être choisies judicieusement pour améliorer la finesse	- Finesse - Sélection
Diversité génétique	Les générations doivent rester diversifiées pour pouvoir comparer de manière réaliste les trajectoires entre elles	- Crossover - Taille de la population - Nombre d'itérations - Mutation génétique
Convergence rapide	Le temps de convergence doit rester raisonnable et ne doit pas excéder 30 minutes	- Complexité des algorithmes - Taille de la population - Nombre d'itérations

La suite de l'étude consiste alors à adapter les étapes de sélection naturelle à l'optimisation de la furtivité de nos trajectoires, tout en respectant ce cahier des charges.

Sous Python, nous créons une classe *GA()* (i.e. Genetic Algorithm) qui permet de stocker l'ensemble de ces fonctions :

Arguments d'entrée de la classe GA()	
Nom de l'argument	Description
GestionnaireRelief	Relief d'étude
pop	Population à faire évoluer

Attributs de la classe GA()			
Nom	Type	Description	Valeur
GestionnaireRelief	gestionnaireRelief()	Stocke le relief sur lequel on travaille	Argument d'entrée
tauxMutation	Flottant	Probabilité qu'un individu subisse une mutation génétique	0.1
elitisme	Bolléen	- Si elitisme == True, on conserve la meilleure trajectoire de la génération précédente	True

Fonctions/Méthodes de la classe GA()			
Nom	Description	Entrées	Sorties
selectionParTournoi()	Sélection d'une trajectoire de la génération précédente par tournoi	Population	Trajectoire sélectionnée
selectionParRang()	Sélection d'une trajectoire de la génération précédente par rang	Population	Trajectoire sélectionnée
crossover()	Création d'un individu enfant à partir de deux parents	- Parent 1 - Parent 2	Enfant
mutation()	Mutation génétique du nouvel individu (trajectoire enfant)	Trajectoire enfant	Trajectoire enfant
evoluerPopulation()	Utilisation des fonctions ci-dessus pour faire évoluer la population	Population	Population

	d'une génération		
--	------------------	--	--

Sélection

Nous allons maintenant enrichir notre population en croisant des individus. Nous allons essayer de prendre des morceaux de solution de certains membres de la population et des morceaux d'autres individus pour créer des nouveaux membres qui, on l'espère, seront des solutions meilleures à notre problème.

Il est certes tout à fait possible de choisir des individus au hasard et de les mélanger aléatoirement pour créer de nouveaux individus, mais d'autres méthodes de sélection plus efficaces existent.

En effet, comme l'objectif est d'améliorer les trajectoires de génération en génération, il faut sélectionner les meilleures trajectoires "parents" de l'ancienne génération pour les croiser entre elles. Attention cependant à ne pas détériorer la diversité génétique. Ainsi, nous utilisons des algorithmes de sélection qui sont capables de conserver un bon compromis entre ces deux critères.

Voici deux méthodes que nous utiliserons :

- La sélection par roulette : Pour chaque individu, la probabilité d'être sélectionné est proportionnelle à son adaptation au problème. Afin de sélectionner un individu, on utilise le principe de la roue de la fortune biaisée. Cette roue est une roue de la fortune classique sur laquelle chaque individu est représenté par une portion proportionnelle à sa qualité de finesse. On effectue ensuite un tirage au sort homogène sur cette roue.

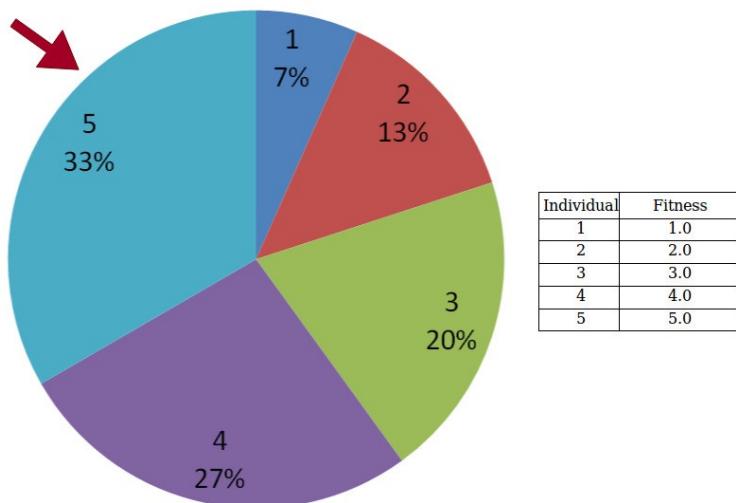


Schéma représentatif de sélection par roulette

- Sélection par tournoi : Cette technique utilise la sélection proportionnelle sur des paires d'individus, puis choisit parmi ces paires l'individu qui a la meilleure finesse. Dans notre cas, sélectionner une trajectoire par tournoi consiste à choisir au hasard 5 trajectoires dans la génération actuelle, puis à choisir celle de finesse la plus haute.

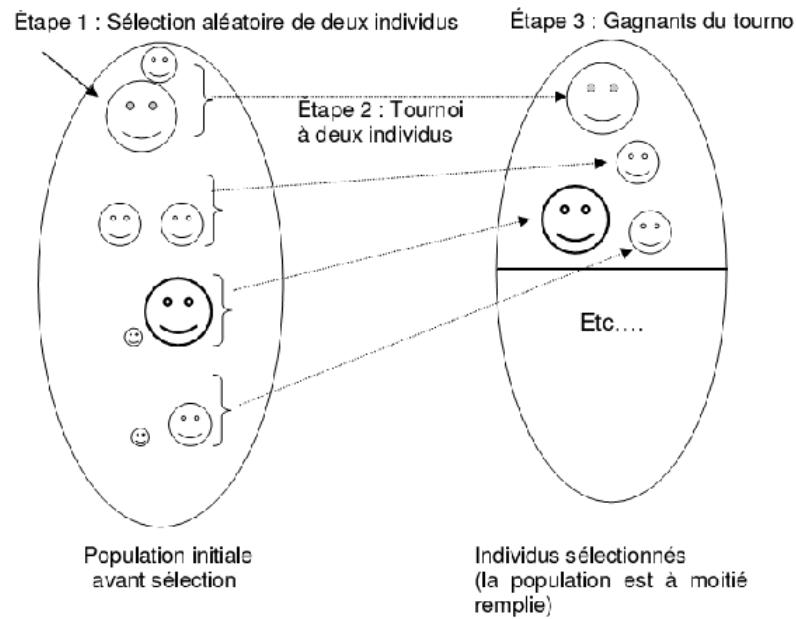


Schéma représentatif de sélection par tournoi

Pour davantage de biodiversité au sein des populations, nous choisissons de sélectionner le premier parent par roulette, et le second par tournoi.

Crossover : Reproduction

Supposons que nous avons sélectionné deux individus parents de la génération précédente. Cette étape de reproduction consiste alors à générer une nouvelle trajectoire "enfant" en fusionnant les deux parents. Ainsi, itérer plusieurs fois ce processus permettra par la suite de faire évoluer la population d'une génération.

Une première question se pose : comment peut-on fusionner deux trajectoires ?

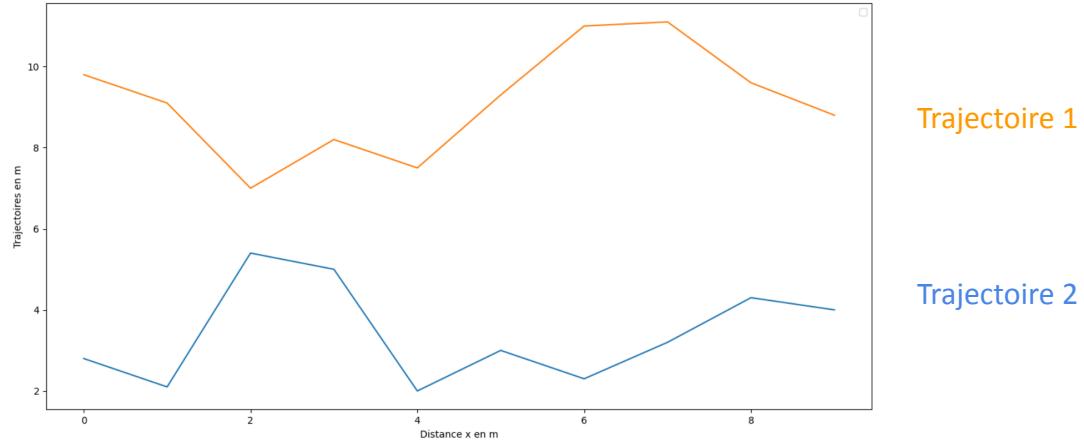
La première méthode évidente consiste à scinder la liste des points des deux trajectoires "parent" au même point, pour recoller les deux parties sectionnées. Par exemple, fusionnons ces deux trajectoires sur 10 mètres à l'aide de cette méthode :

```

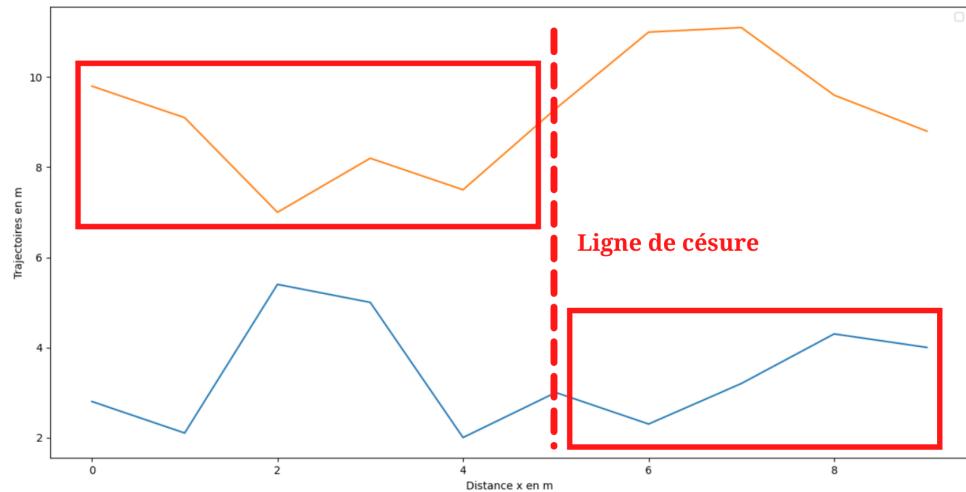
Traj1 = [[0, 2.8], [1, 2.1], [2, 5.4], [3, 5], [4, 2],[5, 3], [6, 2.3], [7, 3.2], [8, 4.3], [9, 6.6]]
Traj2 = [[0, 9.8], [1, 9], [2, 7], [3, 8], [4, 7.5],[5, 9.3], [6, 11.0], [7, 11.1], [8, 9], [9, 8.8]]

```

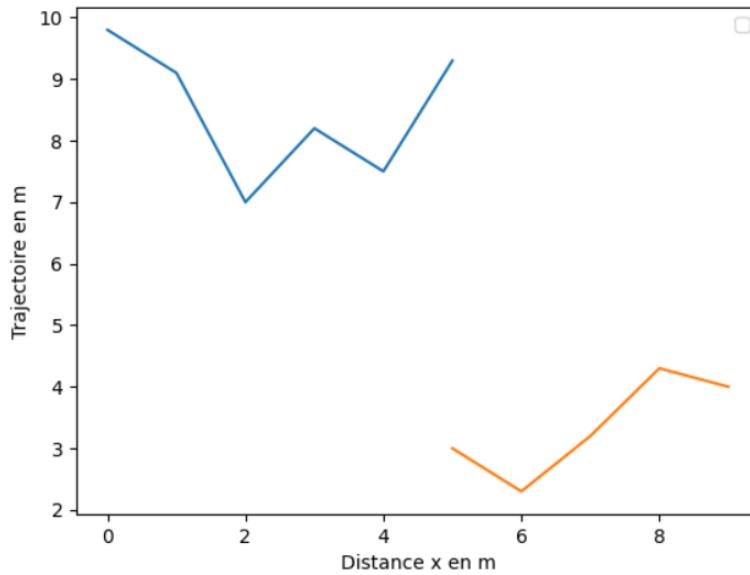
Traçons ces trajectoires afin que ce soit plus représentatif :



On coupe par exemple les deux trajectoires en $x=5m$:



On recolle ensuite les deux demi-trajectoires, en prenant par exemple la première section de la trajectoire 2, et la seconde section de la trajectoire 1 :



Graphe de la trajectoire après fusion

Ainsi, on remarque que fusionner deux trajectoires par leurs coordonnées n'a pas de sens, puisque la trajectoire obtenue est discontinue, ce qui est impossible en pratique. Il faut donc trouver une autre méthode de Crossover.

Après réflexion, nous avons décidé de fusionner les deux trajectoires "parent" non plus par leurs points mais par les déplacements successifs pris par le drone à chaque pas de temps. C'est pour cette raison que la classe *trajectoire()* définie précédemment est composée d'une liste *deplacements*.

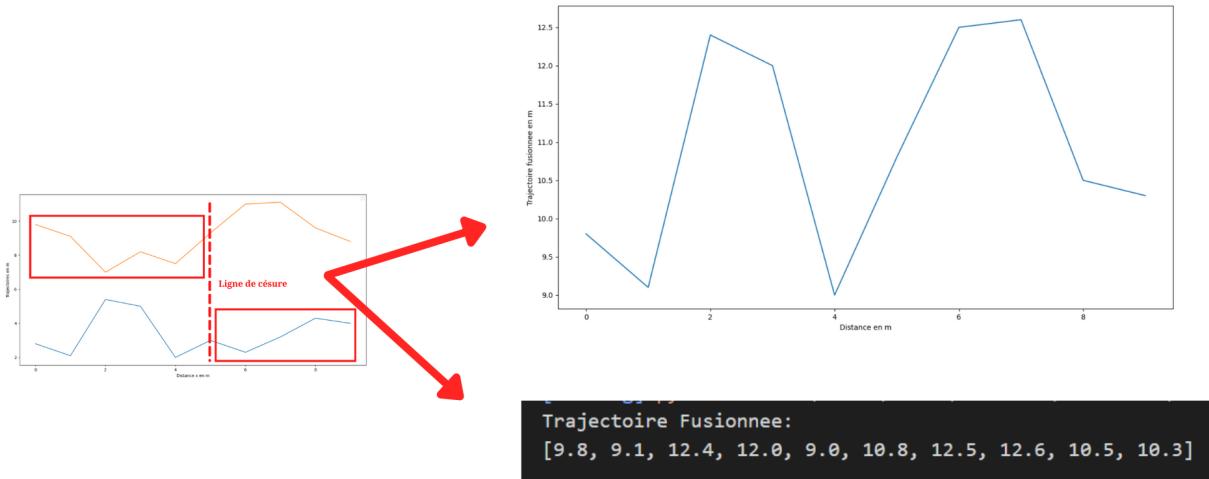
Reprenons notre exemple précédent pour illustrer cette notion :

```
Traj1 = [[0, 2.8], [1, 2.1], [2, 5.4], [3, 5], [4, 2],[5, 3], [6, 2.3], [7, 3.2], [8, 4.3], [9, 6.6]]
Traj2 = [[0, 9.8], [1, 9], [2, 7], [3, 8], [4, 7.5],[5, 9.3], [6, 11.0], [7, 11.1], [8, 9], [9, 8.8]]
```

Dans ce cas, voici les listes des déplacements respectifs :

```
Deplacements 1 :
[[1, -0.7], [1, 3.3], [1, -0.4], [1, -3], [1, 1], [1, -0.7], [1, 0.9], [1, 1.1], [1, 2.3]]
Deplacements 2 :
[[1, -0.8], [1, -2], [1, 1], [1, -0.5], [1, 1.8], [1, 1.7], [1, 0.1], [1, -2.1], [1, -0.2]]
```

Ainsi, fusionner les deux sections de trajectoires ne pose plus de problème, et voici le résultat dans le cas de notre exemple :



Représentation schématique du crossover de deux parents

Voici finalement notre algorithme de reproduction en suivant cette méthode :

Fonction : crossover()	
<u>Objectif</u> : Créer une trajectoire enfant à partir de deux parents trajectoires	
<u>Entrées</u> : <ul style="list-style-type: none"> - self - parent1 - parent2 	<u>Sorties</u> : <ul style="list-style-type: none"> - trajectoireEnfant
<u>Algorithme</u> : <ul style="list-style-type: none"> - Initialisation de l'enfant comme trajectoire - Choix aléatoire de la position de découpage des deux parents trajectoires - On ajoute la première section des déplacements de la trajectoire parent 1 à la trajectoire enfant - On ajoute la seconde section des déplacements de la trajectoire parent 2 à la trajectoire enfant - On met à jour la liste des points de trajectoire de l'enfant - On retourne la trajectoire enfant 	

Mutation génétique

Une fois la trajectoire “enfant” créée, celle-ci subit une mutation juste avant de l’ajouter à notre nouvelle population. Ceci permet d’améliorer une nouvelle fois la diversité génétique. Dans notre problème, la mutation consiste à modifier, avec une probabilité de *tauxMutation = 0.015*, chacun des déplacements de la nouvelle trajectoire. Ceci permet d’éviter que l’algorithme génétique converge trop rapidement vers une trajectoire non optimale.

Voici notre algorithme de mutation génétique :

Fonction : muter()	
Objectif : La trajectoire subit une mutation pour augmenter la diversité génétique	
<u>Entrées :</u> - trajectoireEnfant	<u>Sorties :</u> - trajectoireEnfant
<u>Algorithme :</u> On parcourt la liste <i>déplacements</i> de la trajectoire enfant : Avec une probabilité de <i>tauxMutation=0.015</i> , on modifie chaque déplacement de la liste par un déplacement pris au hasard dans la liste des déplacements possibles (cf. Trajectoire du drone) - On retourne la trajectoire enfant	

Evolution de la population

Ensuite, on utilise successivement les fonctions créées précédemment pour faire évoluer la population d’une génération, grâce à la fonction *evoluerPopulation()* :

Fonction : evoluerPopulation()	
Objectif : Faire évoluer la population d’une génération	
<u>Entrées :</u> - self - population	<u>Sorties :</u> - nouvellePopulation
<u>Algorithme :</u> Initialisation de la nouvelle population si le booléen elitisme (stocké comme argument de la classe <i>GA()</i>) vaut <i>True faire</i> :	

On conserve la meilleure trajectoire de la génération précédente comme première trajectoire de la nouvelle population

Sinon faire :

Pour i allant de 0 à la taille de la population **faire** :

Obtention du premier parent trajectoire par sélection par rang

Obtention du second parent trajectoire par sélection par tournoi

Génération de la trajectoire enfant par crossover des deux parents

Mutation génétique (probable) de l'enfant

Ajout de l'enfant dans la nouvelle population

Fin pour

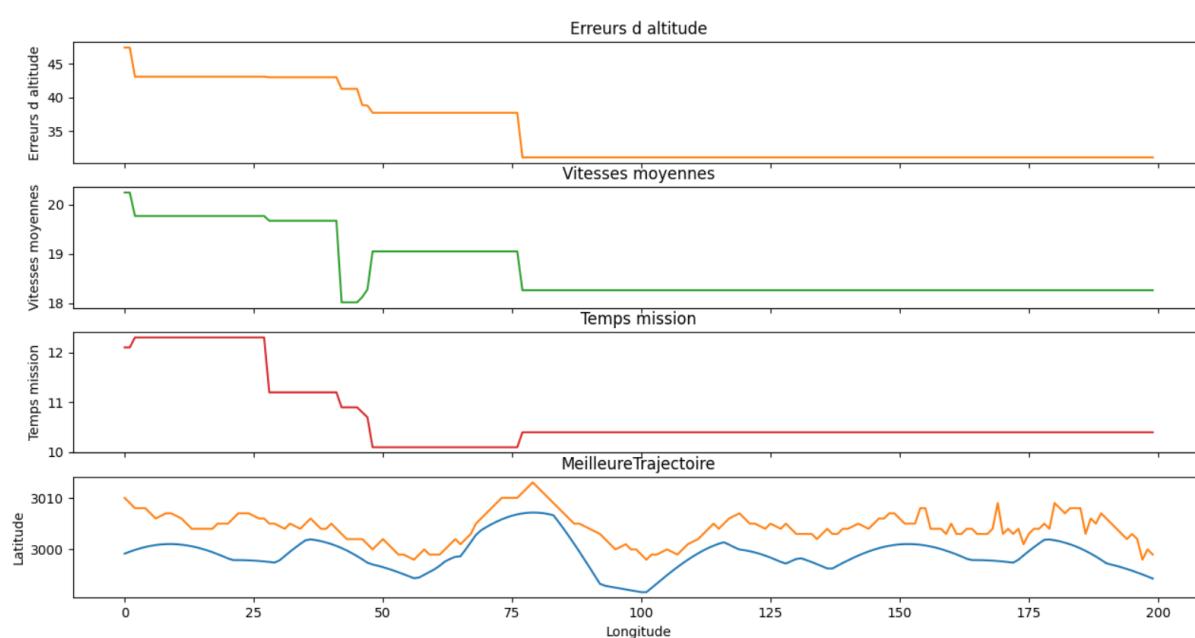
fin si

On retourne la nouvelle population

Premiers résultats

Pour tester notre algorithme, nous faisons évoluer une population de 100 trajectoires sur 200 générations.

On trace ci-dessous différentes valeurs en fonction du nombre d'itérations réalisées (numéro de la génération).



Graphes des valeurs caractéristiques de notre algorithme génétique

A la vue de ces résultats, nous observons que nos algorithmes convergent efficacement dès la 75ème génération.

Cependant, nous remarquons que le temps de mission ainsi que la vitesse moyenne remontent vers la 80ème génération. Cela est dû au fait qu'ici nous privilégiions de limiter l'erreur d'altitude. Et comme nous l'avons vu précédemment, la finesse est une histoire de compromis. Ici, l'erreur d'altitude est le critère avec la plus grande pondération.

Etant donnée la présence de fluctuations importantes de notre tracé, qui impliquerait sûrement des accélérations et des angles d'incidence importants, nous prévoyons de lisser la courbe à l'aide d'interpolations et/ou fonctions sinusoïdales.

3. Amélioration du modèle

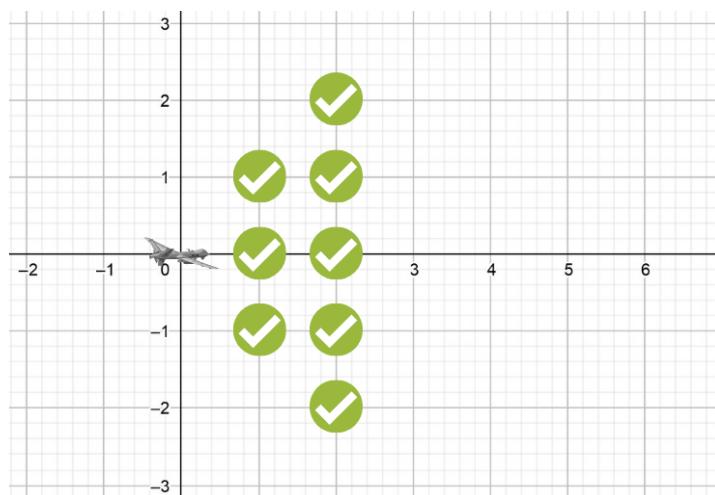
L'objectif de cette partie de l'étude est d'améliorer nos résultats en optimisant différents aspects de notre algorithme.

Amélioration de la discréétisation par 2

Pour rappel, l'algorithme que nous avions rédigé précédemment ne permettait de choisir des déplacements horizontaux et verticaux multiples du mètre à chaque pas de temps :

```
listeDeplacementsPossibles = [[1,1], [1,0], [1,-1],  
[2,2],[2,1],[2,0], [2,-1], [2,-2]]
```

Liste des déplacements possibles pris par le drone pour une discréétisation spatiale d'1 mètre



Représentation graphique des positions possibles prises par le drone pour une discréétisation d'1 mètre

Nous souhaitons maintenant affiner la précision de notre algorithme en choisissant de travailler sous une discréétisation spatiale de 0.5 m.

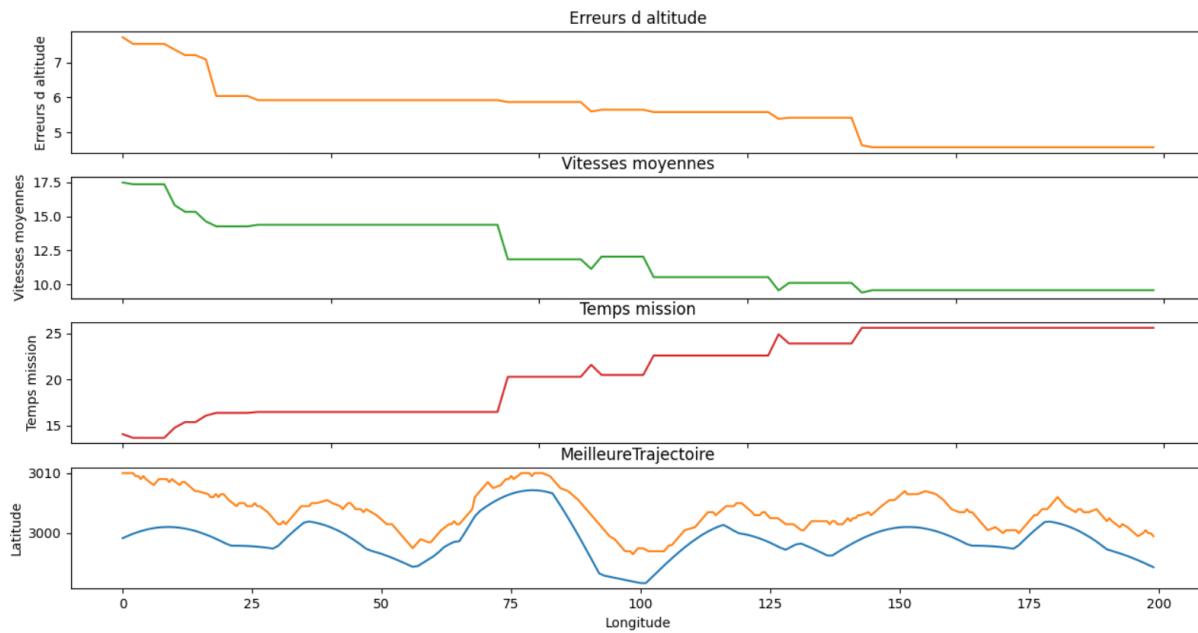
Voici la nouvelle liste des positions possibles pour le drone :

```
[[0.5, 0], [0.5, 0.5], [0.5, -0.5], [1.0, 0], [1.0, 0.5], [1.0, -0.5], [1.0, 1.0], [1.0, -1.0], [1.5, 0], [1.5, 0.5], [1.5, -0.5], [1.5, 1.0], [1.5, -1.0], [1.5, 1.5], [1.5, -1.5], [2.0, 0], [2.0, 0.5], [2.0, -0.5], [2.0, 1.0], [2.0, -1.0], [2.0, 1.5], [2.0, -1.5], [2.0, 2.0], [2.0, -2.0]]
```

Liste des déplacements possibles pris par le drone pour une discréétisation spatiale de 0.5 m

Ensute, nous adaptions la plupart des classes de notre algorithme afin de les rendre compatibles avec ce nouveau pas spatial.

Voici un exemple des nouveaux résultats que nous obtenons pour une discréétisation de 0.5 m, une population de 100 individus que nous faisons évoluer sur 100 générations :



Trajectoire obtenue après 100 générations d'une population de 100 trajectoires et critères associés

Nous remarquons que notre algorithme fonctionne toujours et la trajectoire reste furtive.

Comparons ces nouveaux résultats obtenus avec notre modèle précédent (précision spatiale d'1 m) lorsque l'on fait évoluer une population de 100 individus sur 100 générations :

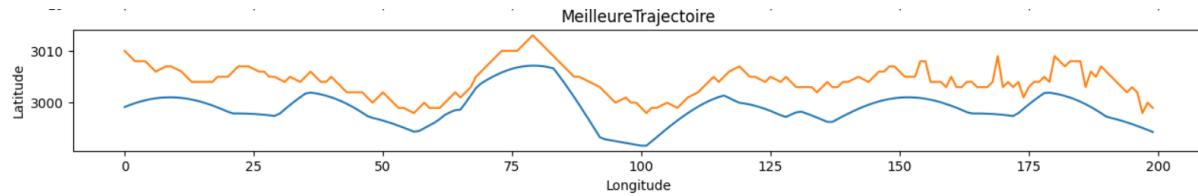
Précision spatiale	1 m	0.5 m
Erreur moyenne sur l'altitude (m)	6.8	4.6
Vitesse moyenne (m/s)	150	9.4
Temps de mission (sec)	11.5	25.5
Temps de convergence de l'algorithme (min)	11.25	24

On observe que l'erreur moyenne d'altitude, qui pour rappel correspond à l'écart moyen entre le relief et le drone lors de sa trajectoire, diminue. Ceci est très encourageant puisque plus cette erreur est faible, plus le drone est furtif. Néanmoins, on remarque que la vitesse moyenne diminue et le temps de mission augmente, et donc le drone sera donc moins rapide dans la réalisation de sa mission pour une précision spatiale de 0.5 m. Ceci n'est pas étonnant puisque la fonction finesse de notre algorithme ne prend en compte que l'erreur d'altitude.

Finalement, le temps de calcul de l'algorithme est nettement plus élevé pour la nouvelle discréétisation spatiale (2 fois plus), ce qui est encore une fois attendu compte tenu de la nouvelle discréétisation spatiale.

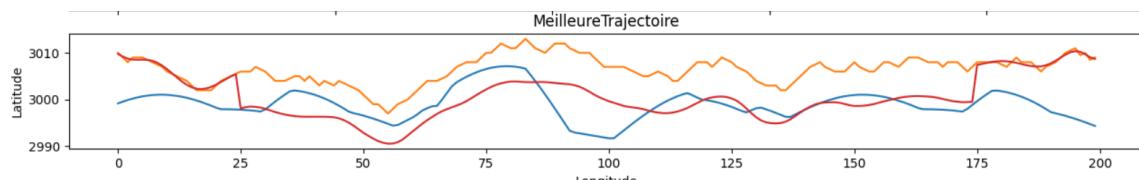
Lissage de la trajectoire

Comme nous pouvons le voir sur le graphe ci-dessous, les trajectoires obtenues numériquement sont très discontinues (présence systématique de "pics"), ce qui ne serait pas le cas dans la réalité :

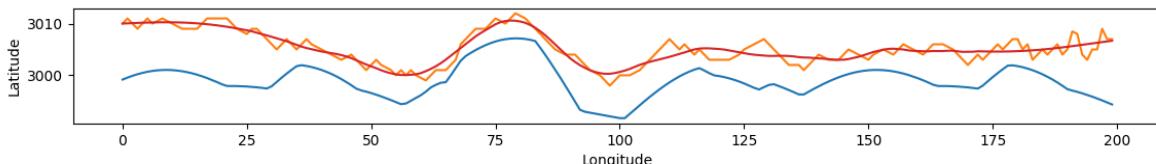


Exemple de trajectoire sans lissage

Ainsi, nous décidons de lisser les courbes des trajectoires par une approximation polynomiale grâce au module `scipy.signal` sous `Python`. Celle-ci prend en argument la trajectoire et le degré du polynôme pour l'approximation. Reste alors à choisir ce degré :



Tracé des trajectoires sans lissage (orange) et avec lissage (rouge) pour un degré 10



Tracé des trajectoires sans lissage (orange) et avec lissage (rouge) pour un degré 5

On remarque que l'approximation par un polynôme de degré 10 n'est pas du tout cohérent avec notre problème. En outre, le polynôme de degré 5 approche très précisément la trajectoire, et après avoir testé d'autres degrés moins pertinents, nous décidons de conserver ce degré 5 pour la suite.

Ainsi, les courbes sont désormais lissées et le drone pourra suivre ces trajectoires lissées qui s'avèrent plus réalistes.

4. Approfondissement : adaptation à tout type de relief

Evolution vers un modèle 3D

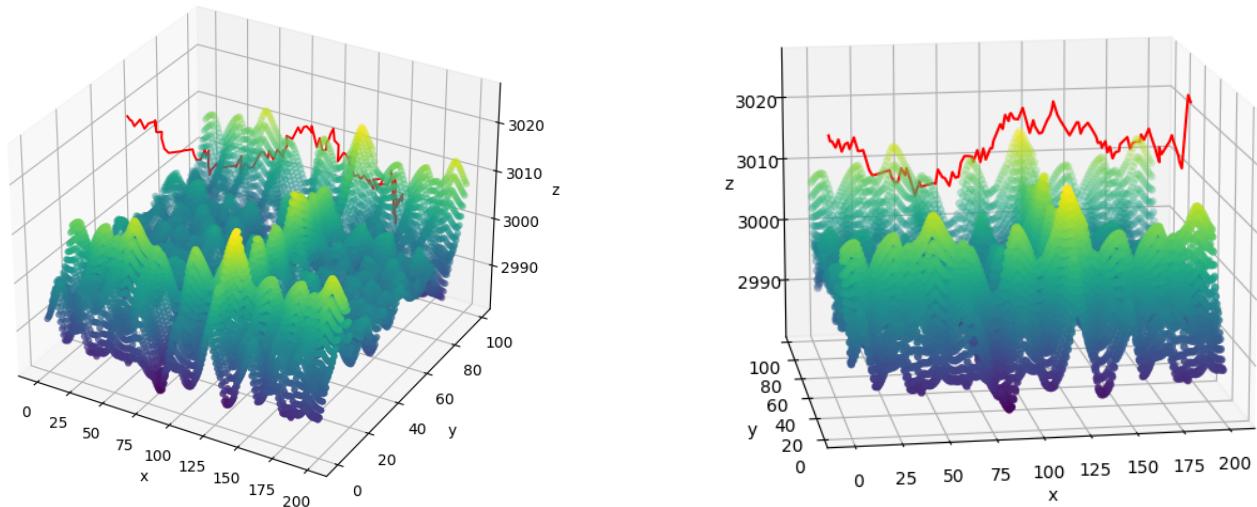
Dans une optique de généralisation de notre travail au 3D, nous sommes partis de nos travaux en 2D. A cela nous avons apporté quelques modifications.

Dans un premier temps, il fallait évidemment revoir les listes de coordonnées comme nous ajoutons une dimension. Ainsi les listes de relief, de déplacements et de trajectoire deviennent des listes de points (x, y, z). Cela a des conséquences sur les crossover, où l'on recolle désormais les déplacements suivant les 3 directions.

La finesse a aussi été impactée par cette modification. En effet, l'erreur d'altitude, précédemment calculée comme une différence de hauteur, devient une différence de rayon autour de la trajectoire. Cette évolution permet d'être plus proche du sol dans les vallées et d'éviter les crash non-verticaux.

Et enfin, au niveau du tracé nous utilisons à présent des fonctions python propre au tracé 3D.

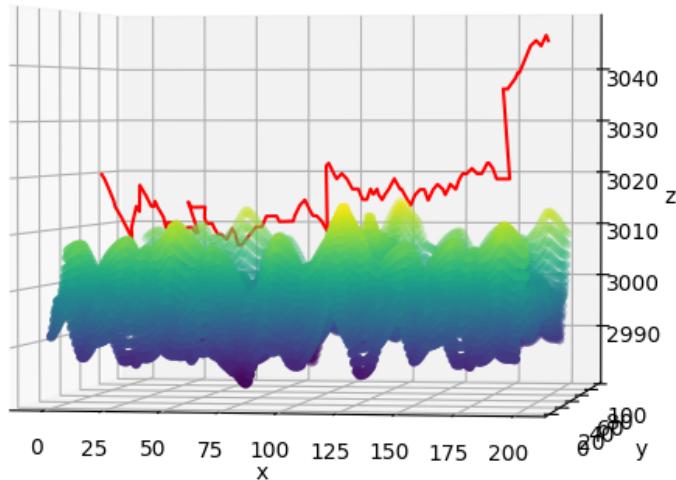
L'ensemble de ces évolutions nous permet d'obtenir les résultats suivants :



Résultats de l'optimisation en 3D pour 10 trajectoires sur 20 générations

On remarque déjà que l'algorithme converge. On a une trajectoire finale possible, qui suit le tracé du relief.

Cependant, on peut observer que cette trajectoire n'est pas encore suffisamment optimisée. Alors nous avons augmenté le nombre de générations, et comme on peut l'observer ci-dessous, les résultats sont nettement plus probants.



Résultats de l'optimisation en 3D pour 10 trajectoires sur 20 générations

Cependant ces résultats semblent quelque peu chaotiques, et c'est sans doute dû à la grande quantité de données à prendre en compte, comme les nouvelles possibilités de déplacements ou les crossovers en 3D.

Il ne reste plus qu'à lisser la courbe. Pour cela il faudrait sans doute penser à combiner 2 interpolations polynomiales : Une suivant chaque dimension.

Création d'un relief à partir de données GPS

Comme précisé précédemment dans notre cheminement jusqu'à la création de trajectoires optimisées dans un relief 3D, nous avons eu l'idée de mettre au point un algorithme de création de relief à partir de coordonnées GPS existantes. L'objectif est de passer d'un relief existant à une liste lisible par nos algorithmes génétiques d'optimisation 3D. Et ainsi nous pourrons optimiser la trajectoire de notre drone dans un environnement existant.

Dans un premier temps nous sommes partis sur l'utilisation d'un API google maps sur Python. Un API ou Interface de Programmation est un chemin que peut utiliser un algorithme

pour utiliser un logiciel. L'objectif est de pouvoir utiliser une fonctionnalité d'un logiciel sans se poser la question de comment cette tâche est mise en œuvre.

Les développeurs créent des interfaces de programmation pour les autres programmeurs, pour l'industrie informatique, mais aussi parfois pour leurs propres besoins. C'est par exemple Uber qui va utiliser la carte de Google Maps dans son application, ce qui évite à l'entreprise de reconstruire une carte du monde pour son application.

Dans notre cas, nous voulions utiliser l'*Elevation API* fournie par Google. Celui-ci permet d'obtenir l'altitude d'un point en entrant ses coordonnées GPS, de cette manière :

Sample request and response

You access the Elevation API through an HTTP interface, with requests constructed as a URL string, using latitude/longitude coordinates to identify the locations or path vertices. Requests must include your API key.

The following example requests the elevation for Denver, Colorado, the "Mile High City", in JSON format:

```
https://maps.googleapis.com/maps/api/elevation/json?locations=39.7391536,-104.9847034&key=[YOUR_API_KEY]
```

Cependant, comme on peut le voir sur la capture d'écran ci-dessus, pour pouvoir utiliser cet API, il est nécessaire d'avoir une clef. Et pour avoir cette clef, il faut payer un abonnement à Google pour l'utilisation de cet API. Nous avons donc abandonné cette méthode et cherché un moyen de créer notre relief autrement.

Ayant des notions dans l'utilisation de la bibliothèque *Selenium* de Python, qui permet d'aller sur internet et d'interagir avec les pages web, nous avons cherché un site internet qui donne l'altitude. Nous avions aussi comme critère de recherche que l'information ne soit pas trop difficile à extraire de la page web. En effet, avec Selenium, la recherche d'éléments se fait par le code source de la page. C'est aussi sur celui-ci que nous travaillons pour extraire l'altitude, il était donc nécessaire de trouver le site avec la construction la plus simple.

Après une analyse, nous nous sommes tournés vers "Elevation Finder". Ce site possède l'avantage d'avoir une barre de recherche, que l'on peut atteindre dans le code source, pour taper les coordonnées. Nous obtenons l'altitude sans avoir à passer par la carte.



Popular Map Tools

Radius Around a Point on a Map
How Far Is It Between
Area Calculator
Measure Distance on a Map
Find ZIP Codes Inside a Radius
Distance Between UK Postcodes
Elevation Finder
UK Postcode Map
Radius From UK Postcode

Map Resources

Download UK Postcodes
Full List of Map Tools

Blog

News

Contact

FAQs

About

About User Menu

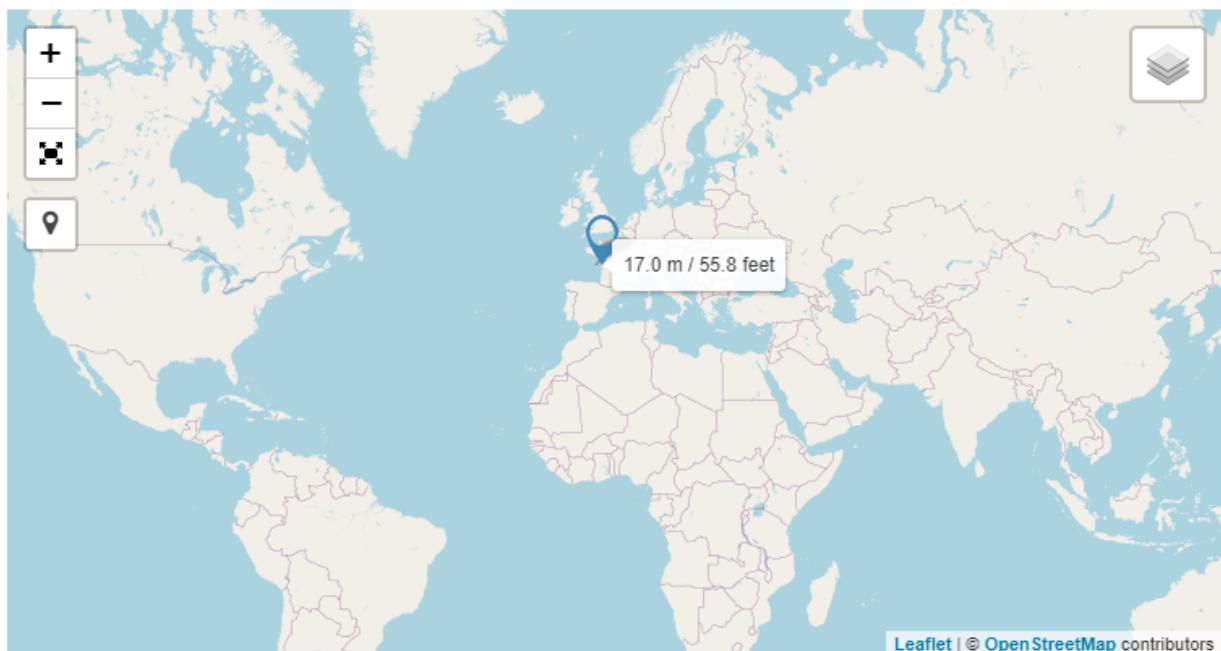
Site News

Find Place With Your Name

The Find Place With Your Name page has been updated. Better

in the text box.

Find Elevation Map



17.0 m or 55.8 feet

Location :46.18302,-1.04564

Click on the map or input a location below to find the elevation.

Latitude, Longitude

▼

Find a Location

Full Screen

Clear Map

Capture d'écran du site Elevation Map

Le premier problème que nous avons rencontré est l'apparition d'une fenêtre pop-up concernant les cookies à l'ouverture de la page. Nous avons vérifié qu'elle s'ouvrait à chaque fois, et nous avons pu coder une réponse qui accepte les cookies, ensuite nous pouvions commencer à travailler sur la page.



Maps you can make

Site Search

ENHANCED BY Goo



FreeMapTools

Popular Map Tools

Radius Around a Point on a Map
How Far Is It Between
Area Calculator
Measure Distance on a Map
Find ZIP Codes Inside a Radius
Distance Between UK Postcodes
Elevation Finder
UK Postcode Map
Radius From UK Postcode

Map Resources

Download UK Postcodes

Full List of Map Tools

Blog

News

Welcome to Free Map Tools

Free Map Tools asks for your consent to use your personal data to:



Personalised ads and content, ad and content measurement, audience insights and product development



Store and/or access information on a device



Learn more

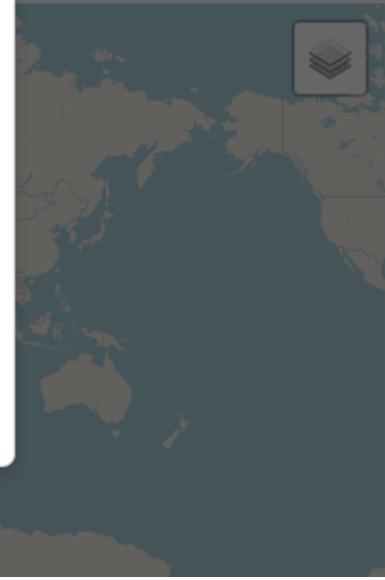
Your personal data will be processed and information from your device (cookies, unique identifiers, and other device data) may be stored by, accessed by and shared with [third party vendors](#), or used specifically by this site or app.

Some vendors may process your personal data on the basis of legitimate interest, which you can object to by managing your options below. You can withdraw your consent at any time by visiting our [privacy policy page](#).

[Manage options](#)

[Agree](#)

click/tap the map or type the address



Capture d'écran des cookies du site Web Elevation Map

Ensuite l'algorithme atteint la barre de recherche, en passant par le code source de la page, et y écrit des coordonnées. Après avoir validé la recherche, le site retourne l'altitude de ce point, que l'algorithme extrait.

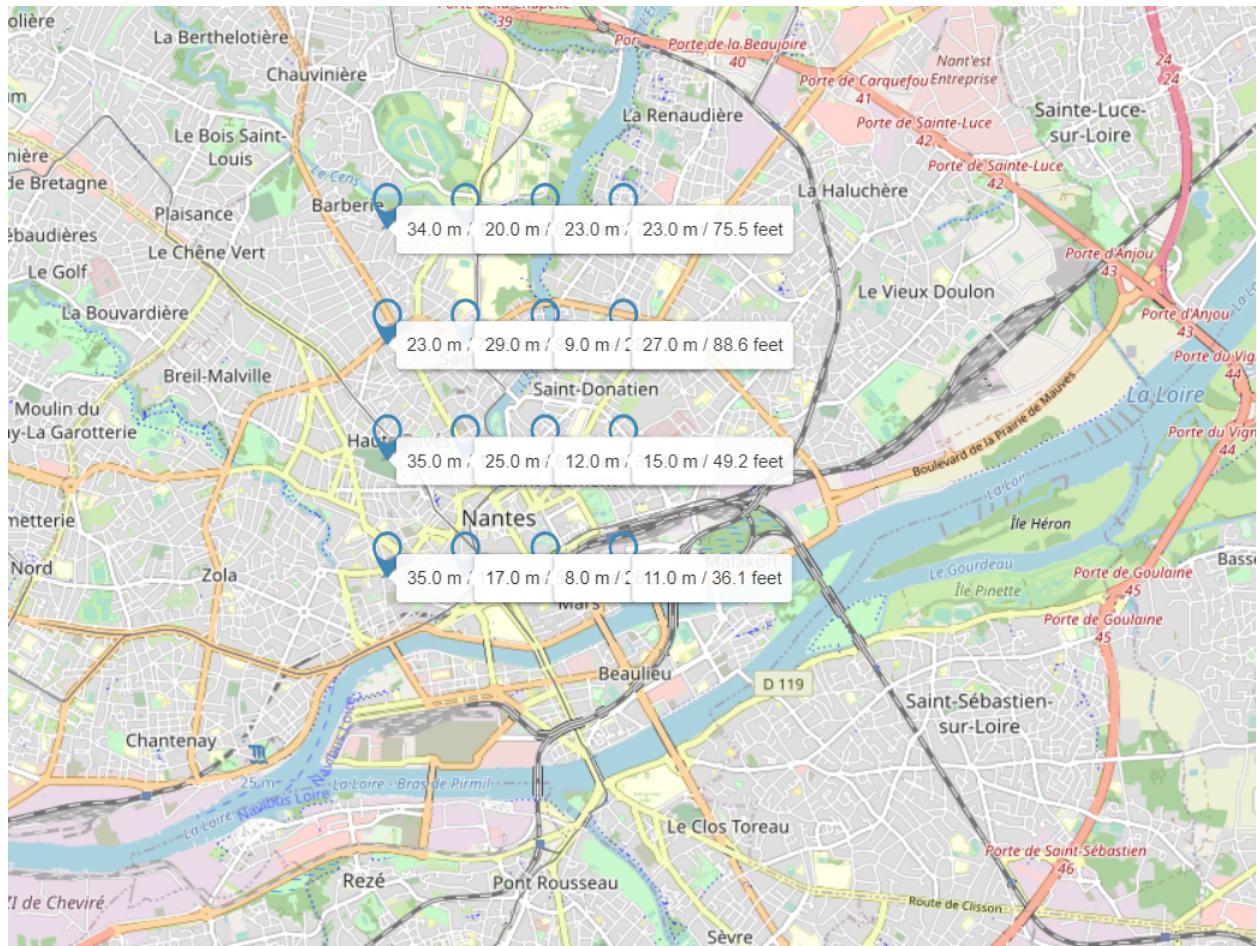
En itérant ce processus N fois, nous sommes en capacité de créer un relief.

Pour créer notre relief, nous prenons les coordonnées du point de départ et celui d'arrivée, puis à l'aide d'une double boucle, nous construisons un carré de points avec un pas d'un mètre, car c'est le pas choisi pour nos algorithmes génétiques 3D.

Ce pas de 1m nous a poussé à revoir nos algorithmes. En effet il faut rentrer des coordonnées en latitude, longitude sur le site, mais pour avoir le pas souhaité nous devons prendre comme paramètres de boucle les distances relatives. Les itérations sont donc faites sur

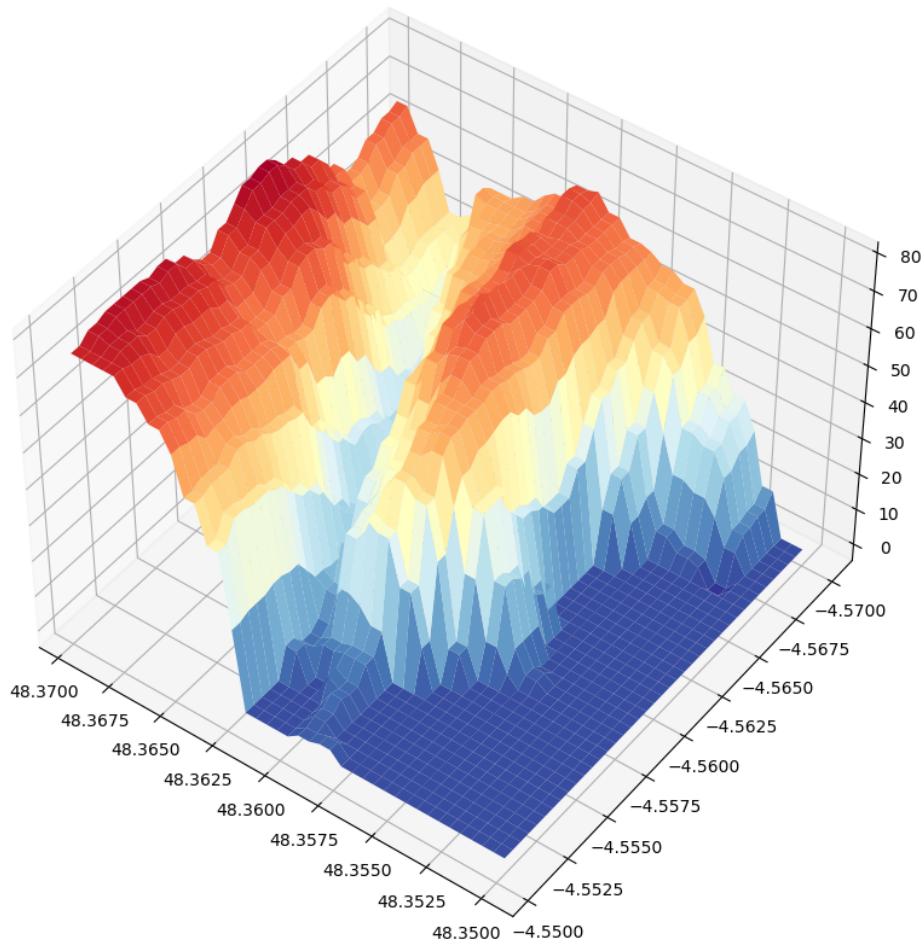
ces distances et dans chacune d'entre elles nous transcrivons ces distances en coordonnées, pour les coller dans la barre de recherche du site.

De cette manière, nous obtenons ce style de sélection de points (ici, pour plus de visibilité, nous avons pris un pas d'un kilomètre) :



Capture d'écran des points tracés sur le site Web

A chaque point, nous enregistrons l'altitude, ce qui nous permet de tracer un relief comme celui ci-dessous :



Ici le tracé nous permet de visualiser le succès de notre algorithme, cependant le plus important est qu'il retourne une liste contenant des triplets définissant chaque point de la forme :

$[(latitude_1, longitude_1, altitude_1), (latitude_2, longitude_2, altitude_2), \dots, latitude_n, longitude_n, altitude_n]$

C'est cette liste que nous prendrons en argument dans les algorithmes génétiques 3D.

Conclusion et suite du projet

Pour conclure, nous avons réussi à mettre en place un algorithme d'optimisation de trajectoire de drone à partir d'algorithme génétique. Notre modèle fonctionne plutôt bien en 2 dimensions. En effet, dans un premier temps les résultats donnés par l'algorithme ont été très encourageants sur le plan de la finesse, ie. de la furtivité. Cependant il y avait quelques soucis d'accélérations et d'angles impossibles pour notre drone.

Ces soucis se sont vus totalement corrigés avec l'amélioration de la discréétisation et le lissage de la courbe. L'harmonie entre une courbe viable pour le drone et la furtivité du tracé justifie une telle utilisation du machine learning.

Pour la suite, il serait sans doute intéressant de repenser la formule de la finesse pour y inclure de nouveaux paramètres, comme la vitesse, le temps de mission ou la consommation ... On pourrait aussi faire varier certains paramètres, comme le taux de mutation, pour trouver une valeur optimale de ceux-ci.

Ensuite, les résultats sur l'adaptation du modèle en 3 dimensions sont très encourageants. On obtient un tracé furtif sous condition de choisir un nombre de générations important.

De surcroît, l'algorithme de génération de relief à partir de coordonnées GPS fonctionne bien et donne en sortie des listes que l'on peut choisir en argument des algorithmes génétiques. On est donc en mesure de calculer une trajectoire sur un relief 3D existant.

Néanmoins, il y a encore du travail sur cette partie. On peut notamment continuer de travailler sur les algorithmes pour fournir des tracés encore plus optimisés. Et il serait intéressant de lisser la courbe, comme précédemment pour le 2D.

Bibliographie

https://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique

Rapport final Zang - Dreux

<https://ledataScientist.com/algorithme-genetique/>

<https://developers.google.com/maps/documentation/elevation/start>