

Rapport de suivi de projet

Amaury Louarn

23 juillet 2015

Table des matières

I	État de l'art	2
1	Inspirations pour le projet	3
2	Solutions envisagées mais non retenues	4
2.1	Mission Planner	4
2.2	U[g]CS	4
2.3	QGroundControl	4
2.4	droneAPI - python	5
2.5	droneAPI - Java	5
II	L'application qDroneShow	6
3	Travail effectué	7
3.1	Implémentation C++ du protocole MAVLink	7
3.2	Application qDroneShow	7
3.2.1	Pré-requis	7
3.2.2	Fonctionnalités	9
4	Futurs ajouts	11
4.1	Affichage en temps réel des positions des drônes	11
4.1.1	Affichage en 2D	11
4.1.2	Affichage en 3D	12
4.2	Contrôle d'effets montés sur les drônes	13
4.2.1	Montage des effets à bord des drônes	13
4.2.2	Coordination avec la station de contrôle	14
4.3	Planification et déroulement automatique d'un script de spectacle	14
4.3.1	Création des scénarios de spectacles	14
4.3.2	Déroulement d'un scénario	15

Première partie

État de l'art

Chapitre 1

Inspirations pour le projet

Le projet se base sur plusieurs vidéos trouvées sur internet. Tout d'abord, la vidéo promotionnelle du spectacle *Air 2015* (<http://air2015.nl>), qui montre des drônes volant dans un stade, et chaque drône porte des effets de sons et lumières. Ensuite, une vidéo d'Ars Electronica (<https://www.youtube.com/watch?v=ShG15rQK3ew>), montrant 49 quadrotors prendre des formations dans le ciel autrichien.

L'idée a donc été d'intégrer des drônes dans le cadre d'un spectacle ouvert au public, où les drônes suivent une « chorégraphie », mais aussi que les drônes puissent réagir à différentes impulsions générées par le public (par exemple, en se déplaçant vers le public après un stimulus, etc.)

Chapitre 2

Solutions envisagées mais non retenues

2.1 Mission Planner

Mission Planner était la solution que nous envisagions au début du projet. Cependant, Mission Planner propose uniquement de contrôler un unique drône. Cela ne nous permettait pas de mettre en place une chorégraphie aérienne composée de plusieurs drônes.

De plus, Mission planner ne propose pas de donner des ordres précis aux drônes en temps réel. Au contraire, Mission Planner se concentre majoritairement sur la plannification de mission, c'est à dire donner des *waypoints* au drôle à l'avance, qui seront ensuite stockés par le drôle, pour être exécutés à postériori.

2.2 U[g]CS

U[g]CS, comme Mission Planner, se concentre sur la plannification de mission, et donc des ordres en différé, et non en direct. De plus, U[g]CS est un logiciel propriétaire, donc difficilement adaptable à nos besoins.

2.3 QGroundControl

De même que U[g]CS, et que Mission Planner, QGroundControl s'oriente vers la planification de missions, et non l'envoi de plan de vol en temps réel. De plus, comme Mission Planner, il ne permet que de contrôler un seul drôle.

Cependant, QGroundControl étant écrit en C++, et étant sous licence LGPL, il nous sera possible d'adapter du code de QGroundControl pour notre application.

2.4 droneAPI - python

DroneAPI, dans sa version python, est la solution technique la plus proche de ce que l'on recherche. En effet, cette API permet de générer en temps réel les messages MAVLink à envoyer aux drônes, ce qui permet notamment de pouvoir envoyer les *waypoints* en temps réel au drôle.

Cependant, cette API comporte un problème majeur. En effet, les programmes utilisant l'API doivent nécessairement être lancés depuis un autre programme, **MAVProxy**. Ce programme sert de proxy entre un drôle et le programme client utilisant l'API. Cependant, il limite grandement les possibilités des programmes clients, car comme les programmes utilisant l'API doivent être lancés depuis une instance de **MAVProxy**, et que chaque instance de **MAVProxy** ne gère qu'un seul drôle, nous ne pouvons pas contrôler de multiples drônes en même temps, depuis un programme unique.

2.5 droneAPI - Java

DroneAPI, dans sa version java, n'est pas utilisable dans notre cas, car (contrairement à ce que son nom indique), l'API java n'est pas l'égale de l'API python, et au lieu de s'intéresser au contrôle d'un drôle par une station au sol, l'API java se concentre sur l'implémentation de l'API REST, qui sert majoritairement à créer des applications web pour les drônes.

Deuxième partie

L'application qDroneShow

Chapitre 3

Travail effectué

3.1 Implémentation C++ du protocole MAVLink

Après avoir analysé le code source de qGroundControl, utilisant l'implémentation en C du protocole MAVLink, il est apparu que cette implémentation n'était pas pratique, et que ce protocole nécessitait une implémentation orientée objet en C++.

De plus, les implémentations du protocole MAVLink sont générées par un générateur¹. Nous avons alors modifié ce générateur pour produire la bibliothèque C++², que nous avons ensuite proposé³ au projet MAVLink afin de l'implémenter dans le générateur « officiel ».

3.2 Application qDroneShow

Nous avons posé la base de ce qui sera une application de gestion de spectacles de drônes. Il s'agit d'une version 0, et même si elle ne peut pas encore être utilisée en production pour créer des spectacles aériens, elle peut déjà être utilisée pour contrôler le décollage et l'atterrissement de plusieurs drônes, ainsi que donner des points de passages au premier drone de la liste.

3.2.1 Pré-requis

Pour faire fonctionner l'application, il convient de remplir plusieurs pré-requis. Tout d'abord, il est nécessaire d'avoir Qt⁴ d'installé (et notamment le compilateur qmake). De plus, il convient de récupérer le code source (disponible sur github⁵ : <https://github.com/TheMrNomis/qDroneShow>).

1. générateur MAVLink : <https://github.com/mavlink/mavlink>

2. voir <https://github.com/TheMrNomis/mavlink>

3. Pull request n° 420 : <https://github.com/mavlink/mavlink/pull/420>

4. <https://www.qt.io>

5. Ne pas hésiter à m'envoyer un mail pour le transfert des droits du dépôt git

Ensuite, une fois l'application compilée et fonctionnelle, il convient d'avoir du matériel disponible. Tout d'abord, un ou plusieurs drônes sont nécessaires, sinon l'application ne sert à rien. De plus, et à fin de communiquer avec les drônes, il est nécessaire d'avoir un transmetteur de télémétrie sur chaque drone, ainsi que sur l'ordinateur faisant tourner qDroneShow.

En dernier lieu, il est important de donner à chaque drone voué à être contrôlé par l'application un **system id** unique. Cela peut se faire depuis Mission Planner ou qGroundControl, en modifiant le paramètre **SYSID_THISMAV** (sur les cartes ardupilot). Le nom du paramètre peut être différent suivant les cartes). Sur Mission Planner, cela se fait depuis le menu **config/tuning**, puis dans **Full Parameter List** (voir figure 3.1).

Veillez à mettre un **system id** unique à chaque drone sur le réseau de télémétrie, en notant que le **system id** doit être compris entre 1 et 254, 255 étant réservé par qDroneShow.

	Command	Value	Units	Options
SR1_PARAMS	0			
SR1_POSITION	2			
SR1_RAW_CTRL	2			
SR1_RAW_SENS	2			
SR1_RC_CHAN	2			
STB_PIT_P	4,5			3.000 12.000
STB_RLL_P	4,5			3.000 12.000
STB_YAW_P	4,5			3.000 6.000
SUPER_SIMPLE	0			0:Disabled 1:Mode 11:Mode1+2+4 12: 19:Mode1+2+5 20: 27:Mode1+2+4+5 34:Mode2+6 35:M 42:Mode2+4+6 43: 49:Mode1+5+6 50: 56:Mode4+5+6 57: 62:Mode2+3+4+5
SYSID_MYGCS	253			1 255
SYSID_SW_MREV	120			
SYSID_SW_TYPE	10			
SYSID_THISMAV	1			1 255
TELEM_RELAY	0			0-10

FIGURE 3.1 – Édition du *system id* d'un drone sur Mission Planner

Enfin, et avant de faire voler le drone, il est important de regarder l'indice K_p ⁶. C'est un indice dosant les quantités de radiations électromagnétiques

6. consultable à l'adresse <http://www.swpc.noaa.gov/products/planetary-k-index>

arrivant sur terre. Si l'indice K_p est trop important, il donne lieu à des perturbations du signal GPS, ce qui peut être dangereux car les drônes naviguent essentiellement grâce au GPS.

3.2.2 Fonctionnalités

L'application dispose de plusieurs fonctionnalités utilisables. Tout d'abord, une fois l'application connectée (soit en USB à 115200 bauds, soit par télémétrie à 57600 bauds), l'application recherche automatiquement tous les drônes connectés sur le réseau. Pour ce faire, à chaque fois que l'application reçoit un message MAVLink, elle regarde si l'émetteur est connu et, dans le cas contraire, l'ajoute à la liste des drônes connus.

Ensuite, une fois un ou des drônes connectés, l'application affiche dans sa partie gauche la liste des drônes connectés (visible sur la figure 3.2), ainsi que différentes actions réalisables pour chaque drôle.

- Le bouton *arm* sert à armer le drôle, c'est à dire l'autoriser à faire tourner les moteurs si besoin (par soucis de sécurité, les moteurs sont désactivés lorsque le drôle est désarmé). Si le drôle est armé, ce bouton sert à le désarmer.
- Le bouton *stop* sert à arrêter le drôle, et à lui envoyer un signal d'extinction.
- Le bouton *takeoff*, utilisable seulement une fois le drôle armé, sert à demander le décollage du drôle.
- Le bouton *land*, utilisable lui-aussi uniquement quand le drôle est armé, sert à demander au drôle d'atterrir. (Attention : lorsque le drôle atterrit, il change de mode de vol pour passer en mode Landing, et ne peut plus être armé tant qu'il est dans ce mode)
- Le bouton *restart* sert à changer le mode d'action du drôle pour passer en mode vol guidé. Ce bouton est utile pour faire repartir le drôle après l'avoir fait atterrir, pour le remettre dans un mode compatible avec le vol.

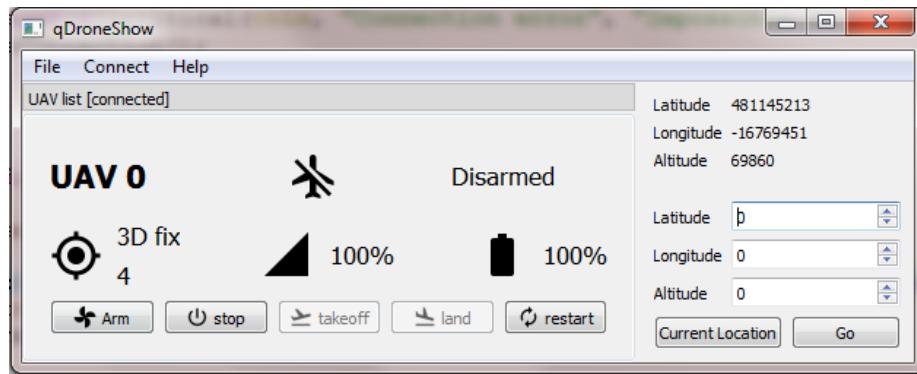


FIGURE 3.2 – L'application qDroneShow, avec un drôle de connecté

Enfin, dans la partie droite de l'application se trouve l'affichage de la position GPS du premier drône de la liste, ainsi qu'un formulaire pour modifier cette position. Attention : les coordonnées GPS indiquées correspondent aux coordonnées réelles multipliées par 10^7 pour la longitude et la latitude, et par 10^3 pour l'altitude, tel que définit dans le protocole MAVLink, car cela permet d'utiliser des entiers à la place de flottants pour la transmission, et gagner en précision.

Chapitre 4

Futurs ajouts

Le projet est déjà bien avancé et dispose d'une V0 fonctionnelle, cependant plusieurs ajouts sont à prévoir avant de pouvoir sortir une V1.

4.1 Affichage en temps réel des positions des drônes

Pour l'instant, nous récupérons certes la position des différents drônes connectés, mais nous n'affichons que la position du premier drone, et la notation « latitude, longitude, altitude, » affichée directement à la sortie des capteurs, ne facilite pas la compréhension de l'utilisateur.

C'est pourquoi il serait utile d'implanter un système de visualisation en temps réel des positions des différents drones. Pour ce faire, nous avons envisagé d'utiliser une carte, soit en deux dimensions comme sur les autres stations de contrôle (qGroundControl et Mission Planner), soit en trois dimensions, qui serait plus facile à lire par l'utilisateur (notamment pour les altitudes des drones).

4.1.1 Affichage en 2D

Sur internet, de nombreuses cartes en deux dimensions sont disponibles, telles que la carte communautaire libre d'utilisation Open Street Map¹, ou les cartes satellitaires professionnelles telles que Google Maps² ou Bing Maps³.

De plus, plusieurs bibliothèques permettent de mettre en place rapidement ces cartes dans une application Qt, telles que `qt-google-maps`⁴ pour Google Maps, `Bing Maps API`⁵ pour Bing Maps (uniquement pour les applications du store windows 8 par contre), et enfin de nombreuses bibliothèques pour

-
1. <https://www.openstreetmap.org>
 2. <https://www.google.fr/maps/>
 3. <https://www.bing.com/maps>
 4. <https://code.google.com/p/qt-google-maps/>
 5. <http://www.microsoft.com/maps/choose-your-bing-maps-API.aspx>

Open Street Map, telles que **QMapControl**⁶, ou **Marble**⁷ (liste complète disponible sur le wiki d'Open Street Map : <http://wiki.openstreetmap.org/wiki/Frameworks>).

C'est l'option choisie par les deux stations de contrôles libres les plus utilisées (*Mission Planner* et *qGroundControl*, voir figure 4.1).

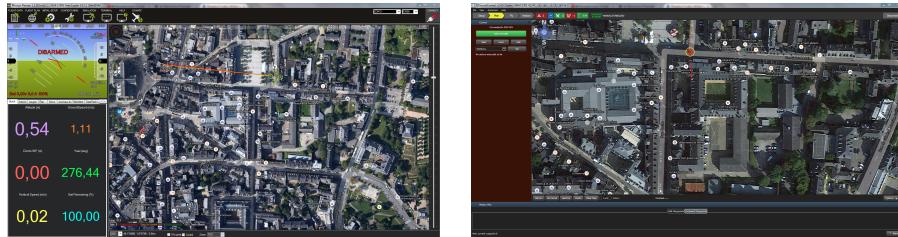


FIGURE 4.1 – Mission Planner et qGroundControl avec leurs cartes en 2 dimensions

Avantages

- Solution facile et rapide à implémenter.
- De nombreuses cartes sont disponibles.

Inconvénients

- L'altitude des drônes n'est pas représentée.
- Si nous donnons des formations aériennes aux drônes, il est très difficile de voir le rendu du point de vue d'un spectateur au sol.

4.1.2 Affichage en 3D

L'affichage en 3D serait utile pour afficher les drônes car il permet de prendre en compte les différences d'altitudes. De plus, grâce à la bibliothèque **osgEarth**⁸, il est possible d'afficher la topologie du terrain à l'emplacement considéré. Cela permettrait de prendre en compte le relief du terrain sur lequel la prestation se déroule.

Sur la figure 4.2, on peut voir deux exemples de rendus par la bibliothèque **osgEarth** : une modélisation de Seattle (où l'on peut voir la modélisation des routes et des bâtiments), ainsi qu'une modélisation du Mont Ranier (où l'on peut voir la modélisation du relief du terrain).

6. <http://www.medieninf.de/qmapcontrol>

7. http://wiki.openstreetmap.org/wiki/KDE_Marble

8. <http://osgearth.org>



FIGURE 4.2 – Exemples de rendu de osgEarth : Seattle et le mont Ranier

Avantages

- L'altitude des drônes est visible graphiquement.
- Il est aisément de visualiser une formation ou une erreur de positionnement d'un drone.
- L'ergonomie de l'application est améliorée (notamment pour les formations en 3D des drônes).

Inconvénients

- Solution plus compliquée à mettre en œuvre.
- Moins de cartes sont disponibles, et ne disposent pas forcément d'une résolution suffisante (voir <http://docs.osgearth.org/en/latest/data.html>).

4.2 Contrôle d'effets montés sur les drônes

Dans le cadre d'un spectacle animé par des drônes, il peut être intéressant que ces drônes incluent des effets de sons et lumières. De plus, il peut être intéressant que ces effets soient coordonnés avec la station de contrôle, afin d'obtenir non pas des effets locaux pour chaque drone (ce qui inclurait des problèmes de synchronisation), mais un effet global constitué de multiples effets lancés au même moment par tous les drônes.

4.2.1 Montage des effets à bord des drônes

Pour monter les effets, il paraît utile d'inclure un microcontrôleur (arduino, teensy, ou même carte personnalisée) qui gère les effets, et auquel il suffit de donner des ordres afin d'obtenir les effets désirés.

Pour commander ce microcontrôleur, les cartes APM (et sûrement les cartes pixhawk, mais cela reste à confirmer) disposent d'un port i2c, qui permet de donner des ordres à une chaîne de composants. Il serait donc possible de brancher

le microcontrôleur “effets” sur le bus `i2c`, afin que la carte autopilote du drône gère les effets.

Enfin, il sera nécessaire de modifier le firmware de la carte pour générer les commandes `i2c` nécessaires.

4.2.2 Coordination avec la station de contrôle

Commander les effets à bord du drône est important, mais il est encore plus intéressant si la station de contrôle donne des ordres aux drônes, afin que ceux-ci gèrent leurs effets.

Grâce au protocole MAVLink, et à sa possibilité d’ajouter des messages personnalisés⁹, il est possible de créer son (ou ses) propre(s) message(s). Ainsi, il sera possible d’ajouter des messages demandant, par exemple :

- d’allumer ou déteindre les LEDs,
- d’allumer ou d’éteindre des lasers,
- de pointer les lasers vers un autre endroit, etc.

Enfin, il sera encore une fois nécessaire de modifier le firmware de la carte autopilote, afin de lui ajouter les nouveaux messages, et de faire réagir le drône en conséquence.

4.3 Planification et déroulement automatique d’un script de spectacle

Cette fonctionnalité est la fonctionnalité centrale de l’application. Il s’agit de créer d’un côté un scénario de déroulement d’un spectacle, et d’un autre côté, de dérouler ce scénario afin de donner les ordres aux drônes en temps réel.

4.3.1 Création des scénarios de spectacles

Ce premier mode de fonctionnement de l’application ne nécessite aucune gestion des drônes. Il s’agit d’afficher un éditeur de points de passages, que l’utilisateur pourra modifier.

L’idée actuelle de fonctionnement de cet éditeur a été de fournir une *timeline* du spectacle, composée d’une succession d’*images-clés*, où chaque image-clé correspond à une formation aérienne (emplacements des drônes, et gestion des effets). Entre chaque image-clé, la station de contrôle ordonne aux drônes de se déplacer vers les positions correspondantes à l’image-clé suivante, et leur laisse le temps de se positionner.

L’éditeur d’image clé a été inspiré par le plugin qu’a créé l’équipe d’*Ars Electronica* pour commander leurs *Spaxels* (voir figure 4.3), ainsi que le mode d’édition des points de passage du logiciel U[g]CS (voir figure 4.4). Ainsi, pour chaque *image-clé*, un espace en 3 dimensions serait créé, affichant le relief du

9. voir : http://www.qgroundcontrol.org/dev/create_new_mavlink_message

terrain sur lequel le spectacle aura lieu, et à chaque drône est associé un cube coloré (dont la couleur peut représenter l'état des effets de ce drône à cet instant), que l'utilisateur peut bouger, afin de créer les formations.

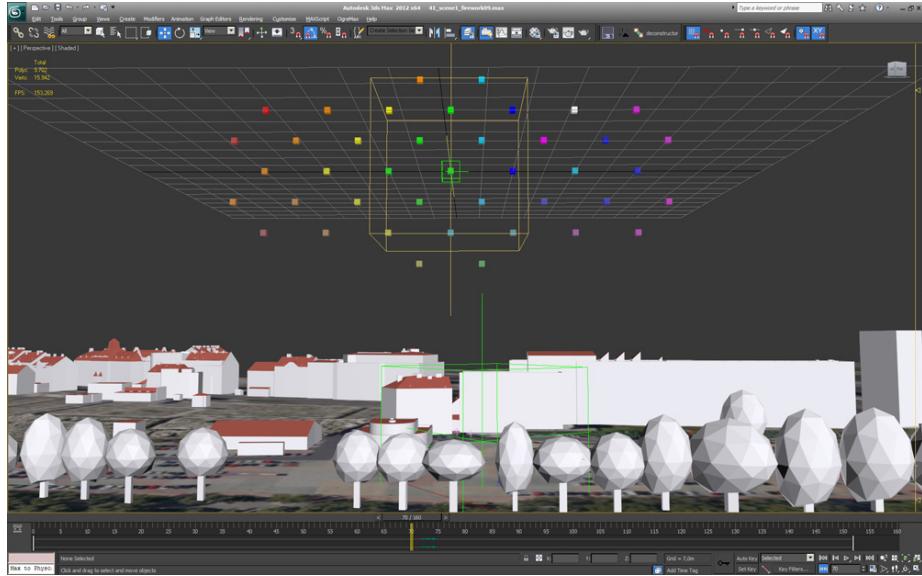


FIGURE 4.3 – Plugin 3ds Max utilisé pour gérer les formations par Ars Electro-nica

4.3.2 Déroulement d'un scénario

Une fois le scénario créé et enregistré par le mode de création, le logiciel servirait de « lecteur de scénario » : le logiciel lit l'état des drônes pour l'image-clé, envoie les informations de déplacement aux drônes, attend que chaque drône soit positionné, puis envoie les informations d'effets (ce qui permet d'obtenir les lancements des effets en simultané). Puis, le logiciel recommence pour l'image clé suivante, jusqu'à la fin du spectacle.

Comme possibilité future, il est aussi envisageable que la station de contrôle ne réagisse pas uniquement aux ordres du scénario, mais puisse aussi réagir à des événements extérieurs, comme par exemple un appui sur un bouton lance des effets, ou autre (un exemple qui avait été cité était le port de bracelets par certaines personnes du public qui, couplés à une détection des mouvements et une géolocalisation du bracelet, fasse bouger un des drônes juste au dessus du porteur de bracelet).



FIGURE 4.4 – Édition des points de passages du logiciel U[g]CS