

PROJET DRONE

DOCUMENTATION DEVELOPPEUR

INTRODUCTION

Ce document est prévu à l'usage des développeurs. Il regroupera donc les détails techniques dont pourraient avoir besoin quiconque désire continuer le projet ou simplement comprendre plus en détails les travaux effectués.

Si toutefois vous désirez des informations plus pratiques au sujet de l'utilisation du système sans entrer dans les détails, n'hésitez pas à consulter le document intitulé « Documentation Utilisateur ».

Le Projet Drone est composé de trois parties distinctes et en partie dépendantes :

- Une partie électronique.
- Une partie informatique regroupant l'application Android et le code Python.
- Une partie mécanique.

Ces trois parties sont interfacées de la façon suivante :

La partie informatique permet à l'utilisateur de transmettre des données entre son smartphone qui agira en émetteur et la RaspberryPi. Celle-ci traitera les commandes et, grâce à la partie électronique, diffusera les signaux voulus aux différents composants (moteurs, servomoteurs etc...) par le biais d'une carte électronique. La partie mécanique sert de support physique et permet d'opérer les mouvements choisis.

En cas de problèmes, ou d'incompréhension, vous pouvez nous contacter par le biais de nos adresses mail UPMC (via l'annuaire). Pour rappel, voici nos noms et prénoms :

Clément FESTAL, Gaël DOTTEL et Narimane LOUAHADJ

CARTE ELECTRONIQUE

1. BESOINS

Nous avons réalisé une carte électronique dans le but de tout centraliser, faciliter le montage et pouvoir résoudre les problèmes s'il y en a. Aussi, étant donné qu'il s'agit d'un objet mobile, il vaudrait mieux que tout soit fixé et que rien ne soit débranché à cause de la multitude de câbles présents à son bord.

2. LE SCHEMA

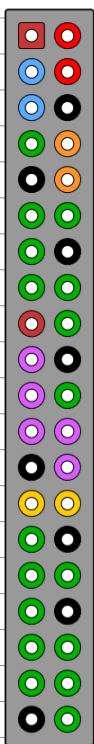
I. REPRESENTATION GENERALE

Le schéma principal représente les composants qui seront placés sur la carte et comment ils sont reliés entre eux. Il faut faire très attention aux branchements pour ne pas se retrouver au final avec une carte soudée qui ne fonctionne pas. C'est le fichier schematic_final_wish.SchDoc . Il s'ouvre avec Altium.

II. BRANCHEMENTS

Dans cette partie, nous allons vous présenter les différentes liaisons effectuées. Il est à noter que nous n'avons pas fait passer la puissance sur la carte (pas de batterie ni de contrôleurs).

A) LA RASPBERRY PI

Raspberry Pi2 GPIO Header					
Pin#	NAME		NAME	Pin#	
01	3.3v DC Power		DC Power 5v	02	
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04	
05	GPIO03 (SCL1 , I ² C)		Ground	06	
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08	
09	Ground		(RXD0) GPIO15	10	
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12	
13	GPIO27 (GPIO_GEN2)		Ground	14	
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16	
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18	
19	GPIO10 (SPI_MOSI)		Ground	20	
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22	
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24	
25	Ground		(SPI_CE1_N) GPIO07	26	
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28	
29	GPIO05		Ground	30	
31	GPIO06		GPIO12	32	
33	GPIO13		Ground	34	
35	GPIO19		GPIO16	36	
37	GPIO26		GPIO20	38	
39	Ground		GPIO21	40	
Rev. 1 26/01/2014 http://www.element14.com					

Nous avons utilisé les pins suivants :

9 : (GROUND) (sortie raspberry pi - entrée controleur gros moteur)

11 : (GPIO) (sortie raspberry pi - entrée controleur gros moteur)

12 : PWM (sortie raspberry pi - entrée servomoteur)

19 : (SPI_MOSI) (sortie raspberry - entrée accéléromètre)

21 : (SPI_MISO) (entrée raspberry - sortie accéléromètre)

23 : (SPI_CLK) (sortie raspberry - entrée accéléromètre)

29 : (GPIO05) SS (Slave Select) connecté à l'accéléromètre, mettre à 0 (sortie raspberry - entrée accéléromètre)

32 : (GPIO12)

37 : (GPIO26)

39 : (GROUND)

B) MOTEURS BRUSHLESS

Nous avons utilisé deux moteurs Brushless que nous avons aussi branché à la carte électronique. Le branchement dans ce cas est simple car il est standardisé. Vous trouverez donc une multitude de tutoriels vous expliquant cela.

C) CONTROLEURS

Comme vous l'avez vu dans le schéma ci-dessus, il y a des contrôleurs. Nous en avons pris deux (un pour chaque moteur). Ici, le branchement est aussi standardisé.

III. SOUDURES

A) REGULATEUR DE TENSION

Un régulateur de tension est soudé sur la carte avec d'autres composants lui permettant de fonctionner correctement. Le schéma se trouve dans la datasheet du composant.

B) ERREURS A EVITER

- Si vous êtes amenés à refaire une carte électronique, prenez des pastilles assez grandes pour bien pouvoir souder.
- La distance entre deux pistes doit être grande pour éviter d'en dégrader avec des « tâches » d'étain. Cela peut conduire à des court-circuit et donc endommager la carte !
- Eviter de faire des bulles en soudant. La soudure peut paraître parfaite comme ça mais non. La bulle n'est pas étalée sur toute la surface.
- Vérifier les formes de pattes de chaque composant. Le régulateur est spécial, il faut donc prévoir ce cas.

C) VERIFICATIONS

Pour vérifier qu'il y a bien une connexion entre deux pastilles (après soudure), vous pouvez prendre un voltmètre mettre le fil rouge sur une pastille et la masse sur l'autre. Si un son sort, c'est que c'est bien connecté sinon ... Vous devez revoir vos pistes. Nous avons jugé bon de vous montrer cette méthode car une minorité ne la connaissait pas.

3. LE LOGICIEL ALTIUM DESIGNER

Notre carte électronique a été réalisée intégralement par le logiciel Altium Designer. Afin d'optimiser notre carte électronique, corriger certains défauts liés à l'usure ou tout simplement créer une nouvelle solution, vous avez besoin de le maîtriser.

I. PRESENTATION

Altium Designer est un outil complet de développement de produits électroniques incluant :

- Un outil de routage PCB
- Un outil de simulation SPICE
- Un outil de développement FPGA
- Un outil de développement de code embarqué.

Dans notre cas, nous nous intéresserons particulièrement au routage PCB.

II. UTILISATION

Pour construire vos cartes, il faudra d'abord maîtriser le logiciel. Nous vous proposons dans le dossier « Tutoriels » des aides pour une complète maîtrise du logiciel. Il est important de faire ça au lieu d'entrer dans le vif du sujet directement.

Notons certains points importants à garder en tête lors de la construction du PCB via le logiciel :

- Utiliser du Layer Bottom pour tout le circuit. Le Top Layer sera utilisé uniquement s'il est impossible de faire autrement.
- Eviter l'utilisation des « vias » (chaque via = une soudure supplémentaire).
- Faire des pistes plus grandes (en largeur)

APPLICATION ANDROID

1. ANDROID STUDIO

Tout d'abord, il est à noter que l'application Android a été réalisée à l'aide du logiciel Android studio qui est le logiciel phare pour le développement d'applications Android. En effet, cet EDI (Environnement de Développement Intégré) créé par Google eux même est un must-have car il contient tous les éléments nécessaires à la réalisation d'une application Android qui sont :

- Éditeur de texte personnalisable avec l'auto complétion du code
- Compilateur pouvant aussi bien créer une version de débogage en reliant un téléphone à l'ordinateur, une version de débogage en utilisant un émulateur de téléphone tournant sous Android (peu recommandé car c'est extrêmement gourmand en ressources et soumis à des divers bugs et instabilités) ainsi qu'une version « release » exportable en .apk (extension pour installer l'application sur n'importe quel téléphone Android, après avoir préalablement autorisé l'installation des applications de source non vérifiées)
- Débugueur pour pouvoir voir en temps réel les valeurs des variables, que ce soit en local avec un émulateur ou via un câble USB avec un vrai téléphone. Le débogueur permet également de détecter les exceptions et messages d'erreurs éventuels

2. FICHIERS UTILES

Le nombre de fichiers et de dossier peut paraître énorme au début mais il y a très peu de fichiers utiles au final, la plupart étant créés par Android Studio pour des raisons m'étant inconnues. Je vais vous détailler ici les fichiers utiles et dont vous devrez peut-être modifier le contenu.

I. SPIDER_DRONE.APK

Tout d'abord, vous trouverez, à la racine du dossier, un fichier nommé « Spider drone.apk » qui est utilisé pour installer l'application sur un nouvel appareil Android. Pour cette installation, il vous faut aller dans les paramètres -> sécurité (dans la catégorie « personnel » sur mon téléphone) -> autoriser l'installation d'applications issues de sources inconnues (dans la catégorie « gestion de l'appareil » sur mon téléphone).

Une fois cette autorisation effectuée, il vous faut installer l'apk, pour ce faire, vous pouvez déplacer ce fichier dans un dossier de la mémoire du téléphone, puis ouvrir ce fichier avec l'explorateur de fichier Android. Sinon, vous pouvez également utiliser des applications telles que airdroid pour installer à distance vos applications.

Une fois le fichier ouvert, que cela soit via l'explorateur ou airdroid, on vous demandera de confirmer l'autorisation de l'application d'accéder à certaines fonctionnalités du téléphone. Acceptez, patientez le temps que l'installation se finisse. Vous pourrez ensuite lancer l'application Android nommée « Spider Drone » avec le logo Android (pour la suite des explications, vous pouvez vous référer à la documentation utilisateur)

II. SPIDER_DRONE.JAVA

Vous trouverez ce fichier ici :

« SpiderDrone/app/src/main/java/fr/upmc/polytech/spiderdrone/Spider_Drone.java »

C'est le fichier contenant le code java servant à la réalisation de l'application. Au début, je fais tout une liste d'include nécessaires à l'application. Ensuite, tout le code se trouve dans une unique classe nommée « Spider_Drone » (mais cette classe contient différentes classes que je détaillerai plus tard). Ensuite, vous pouvez lire que j'initialise le port de communication pour les paquets UDP avec le port 2000 (qui est un port appartenant à une plage libre donc qui a peu de chance d'être pris. Cependant, vous serez peut-être amenés à le modifier). Ensuite, je mets l'adresse IP de la machine qui devra recevoir le paquet UDP (ce n'est donc pas votre ou vos téléphones mais la raspberry pi servant à commander le drone). Pour obtenir cette adresse IP, il suffit de taper la commande « ifconfig » sur la raspberry pi. Il est à noter que malgré le fait que cette adresse IP soit fixée dans le code, elle est modifiable directement depuis l'exécution de l'application.

Le premier bloc est uniquement la déclaration d'un type de listener personnel. Il servira uniquement, quand il est appelé à identifier sur quel bouton l'utilisateur a touché. Selon le bouton, il changera le champ de texte contenu sur l'application (qui sert à l'utilisateur pour voir quelle est la commande qu'il émet vers le robot). Ce

champ de texte sera réutilisé par la suite. De plus, si l'utilisateur relâche le bouton, on refait passer le champ de texte à « aucun ».

Pour la suite, il faut d'abord comprendre les bases du système d'exploitation Android. En effet, cet OS très spécial n'utilise pas de fonction « main » classique comme dans n'importe quel autre programme java mais des fonctions aux noms imposées appelées à différents moments. En effet, le schéma suivant (merci à openclassroom, le site remplaçant le Site Du Zéro pour ce schéma) montre comment « l'activité » (c'est comme cela qu'un programme est appelé sous Android) marche.

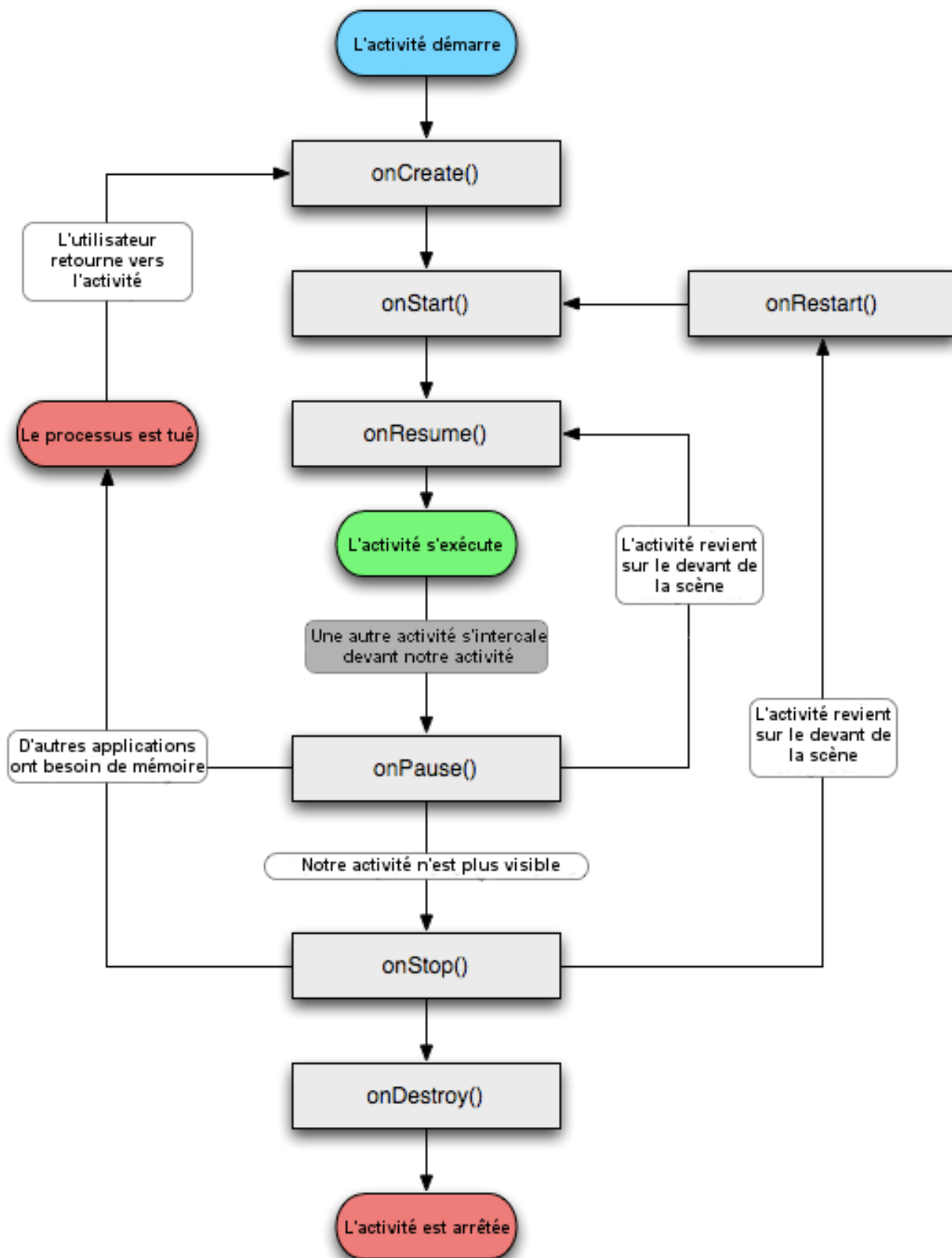


Figure 1 Schéma tiré d'openclassroom expliquant le cycle de vie d'une activité Android

La première méthode est donc « onCreate », son rôle est d'instancier les boutons et différents champs, définis par leur ID (dans un fichier que nous verrons dans la prochaine partie). On leur ajoute ensuite le listener créé précédemment (pour factoriser le code et ne pas créer un listener spécifique à chaque bouton, ce qui aurait été une optimisation très peu efficace).

Ensuite, nous passons aux méthodes « onResume » et « onPause » dont le fonctionnement est complémentaire. « onResume » sert à créer un thread (processus parallèle au processus principal) dont le rôle sera détaillé après. On appelle ensuite la fonction start de ce thread (ce qui fait que le programme principal est bloqué jusqu'à l'exécution complète de ce thread). « onPause », au contraire interrompt le processus et gère ensuite les possibles cas d'erreurs (grâce à un lever d'exception).

Dans le thread nommé « ThreadUDP », je crée juste un thread donc le but va être de créer un autre thread avec une certaine fréquence (grâce au sleep et à toutes les gestions d'exception associées). Tant que le thread n'est pas interrompu, il crée donc un thread nommé « ThreadEnvoiPaquetUDP » et lance ce thread dans un processus secondaire (grâce à la méthode run, qui n'est pas bloquante contrairement au start vu précédemment). La périodicité peut être réglée manuellement grâce au sleep (actuellement, un paquet UDP est envoyé toutes les 1 ms)

Enfin, dans le thread nommé ThreadEnvoiPaquetUDP, nous récupérons la commande utilisateur, contenue dans le champ de texte dont j'ai parlé précédemment. Ce thread risque d'être compliqué à comprendre tant que vous n'aurez pas eu vos cours de réseau mais en gros, ce qu'il faut retenir, c'est qu'il envoie à la raspberry pi une information sous la forme d'un string. Nous avons utilisé des paquets UDP et non des paquets TCP car nous travaillons en temps réel, le renvoi de paquet n'ayant pas atteint leur cible est donc inutile (et même nuisible). Il faudra donc faire avec les caprices possibles du réseau et espérer qu'un grand nombre de paquets arriveront. Là encore, de nombreuses exceptions sont levées.

III. ACTIVITY_SPIDER_DRONE.XML

Vous trouverez ce fichier ici : « SpiderDrone/app/src/main/res/layout/activity_spider_drone.xml ». Ce fichier est un fichier de ressource, type de fichier là encore spécifique au système d'exploitation Android. C'est dans ce fichier que je définis la liste des éléments utilisés, leur positionnement ainsi que leur contenu et leur nom. Ainsi, je vais détailler un seul des cas, le reste étant assez facile à comprendre. Le premier bouton aura l'ID stabilisation. Cet ID est important car c'est grâce à cela que je peux le trouver dans le fichier java. Ensuite, je précise que ce bouton prendra toute la taille nécessaire pour s'afficher entièrement. Ensuite, je précise quel est le contenu de ce bouton grâce au champ text. Enfin, je le centre au milieu de la fenêtre (ensuite, je placerais tous les autres composants les uns par rapport aux autres, afin de faciliter l'utilisation de cette application sur un grand nombre de téléphone, ce qu'un positionnement absolu n'aurait pas permis).

IV. ANDROIDMANIFEST.XML

Vous trouverez ce fichier ici : « SpiderDrone/app/src/main/AndroidManifest.xml ». Ce fichier est extrêmement simple, malgré sa syntaxe très spéciale (c'est un document xml, comme le précédent). Ce fichier a pour seul but de lister les autorisations que l'application demandera lors de son installation (des autorisations nécessaires à l'utilisation du réseau dans notre cas).

CODE PYTHON

1. RASPBERRY PI ET PYTHON

Grâce à la puissance de la carte Raspberry Pi utilisée, il est possible d'utiliser un langage interprété (c'est à dire non compilé avant utilisation) sans avoir de problème de ralentissement. La carte assure alors une flexibilité d'utilisation et ce, notamment lors des phases de développement.

La partie code python est composé d'un fichier unique appelé « main.py ». Il permet à la carte de recevoir correctement les données envoyées par l'application Android grâce à une communication Wi-Fi en UDP puis de les traiter. Pour cela, plusieurs fonctions sont utilisées et appelées autour ou dans la boucle principale de traitement de donnée.

2. FONCTIONS UTILISEES

Il s'agit ici de décrire les fonctions créées par nos soins. Pour les fonctions déjà présentes, elles ont soit un nom explicite, soit une documentation florissante sur internet. Elles ne seront donc pas détaillées.

Les fonctions utilisées ne sont pas nombreuses et restent assez évidentes.

La fonction **PWM_Fun (pin, duty_cycle)** permet de passer le port GPIO numéro « pin » en mode PWM avec un duty cycle de « duty_cycle ». On trouve ici l'une des raisons de l'utilisation du langage Python : les bibliothèques sont très fournies et bien documentées. Ainsi, on inclut « RPI.GPIO » (en tant que « GPIO » tout court pour simplifier la lecture) et on peut utiliser les fonctions permettant de générer une PWM de façon très simple. Cinq fonctions sont ainsi appelées et, grâce à la part d'abstraction introduite ici par le langage, il n'est pas nécessaire de comprendre leur fonctionnement, seulement leur fonction.

Cette fonction permettra donc de configurer les ports nécessaires lors du fonctionnement du système. Elle sert de fonction d'initialisation des ports.

La fonction **PWM_Upd (pwm , duty_cycle)** permet de passer le duty_cycle de la pwm déjà active à « duty_cycle ». Bien que n'ayant plus vraiment lieu d'être, cette fonction fait office de substitue parodie de la fonction « ChangeDutyCycle » de la bibliothèque « RPI.GPIO ». Auparavant, plus de modification étaient effectuées mais il s'est avéré que cela n'était pas nécessaire et nous avons gardé les vestiges de la fonction encore en utilisation.

Cette fonction permettra donc de mettre à jour la valeur du duty_cycle de la PWM choisie, c'est-à-dire de changer directement le signal transmis.

3. FONCTION PRINCIPALE

La fonction principe est composée de deux parties : une phase d'initialisation et une boucle « while True » (« tant que vrai ») qui ne s'arrêtera donc pas.

La phase d'initialisation permet d'aider le développeur en améliorant la lisibilité du code et en facilitant les phases de test. Outre la fonction `signal.signal(signal.SIGTERM, signal_term_handler)` qui permet juste d'appeler la fonction `signal_term_handler(signal, frame)` avant de terminer le processus, on utilise une série de définition de variable qui permettront de définir le pas de variation des `duty_cycle` des différentes PWM.

Ces variables permettront donc au développeur d'ajuster le pas de traitement en fonction de la vitesse de réception des données pour avoir une accélération ou une décélération des vitesses des moteurs conforme à une certaine maniabilité par exemple. Elles sont accompagnées de variables MAX permettant de fixer des valeurs limites à la largeur de la PWM. Ces variables vont en générale de 2 à 12 pour des valeurs allant de 0 à 100% mais il faut les redéfinir précisément à chaque changement de matériel car elles peuvent varier significativement d'une pièce à l'autre.

Les initialisations suivantes mettent en place les structures utilisées dans la boucle principale. Pour la première, il s'agit d'une liste de liste (soit une liste à deux dimensions) qui représente une liste de caractéristique de PWM. Les caractéristiques de cette PWM sont représentées par quatre éléments :

0. Le numéro de port.
1. Le `duty_cycle`.
2. Un bit d'initialisation indiquant si la PWM a déjà été instanciée.
3. Un bit de modification indiquant si la valeur de `duty_cycle` a besoin d'être mise à jour.

La deuxième structure est une simple liste de PWM définies. En utilisant ces deux structures, on peut donc garder un code lisible tout en modifiant de manière simple le bon signal au bon moment.

La partie de communication intervient ensuite et clôture les initialisations. Grâce à des socket, on accepte une communication UDP en réception. Une fois de plus, la magie de Python fait son office et on n'a pas vraiment besoin de comprendre tous les mécanismes derrière les fonctions appelées de la bibliothèque « socket ». En sachant que « `data, addr = sock.recvfrom(1024)` » récupère l'information transmise dans la variable « data » à chaque appel, on peut mieux comprendre le fonctionnement de la boucle principale.

La boucle principale est effectuée à l'infini et opère de la manière suivante. Après avoir récupéré la donnée envoyée dans « data » (avec une temporisation pour ralentir l'opération) on la soumet à une série de « elif » ou « else if » dans la plupart des autres langages. Ceux-ci permettent de mettre à jour la bonne PWM à la bonne valeur de `duty_cycle` grâce aux structures décrites plus haut. En effet, la liste de liste est mise à jour dans les « elif » et, en fin de boucle, une boucle for permet de mettre à jour dans la liste des PWM toutes les valeurs ayant été modifiées.

PARTIE MECANIQUE

1. STRUCTURE GLOBALE

Réalisée avec du carton actuellement, la maquette est fonctionnelle mais reste à l'état de prototype.

Le fonctionnement du système repose sur des principes mécaniques de bases. Un plan de dimensions plutôt larges créé la base de la structure sur laquelle reposera le reste du système. Les roues sont placées de façon classique de manière à « couvrir » l'hélice de propulsion en cas de contact avec un mur. Il reste alors deux points qui pourront être critique dans la reproduction de la maquette ou son amélioration.

2. MOTEUR ET HELICE DE PROPULSION

Il est nécessaire d'avoir une transmission de la force engendrée par la rotation de l'hélice tout en respectant des contraintes de positionnement. L'hélice doit en effet rester à une certaine hauteur du sol pour pouvoir tourner librement. Il faut donc créer un support permettant la suspension du moteur et de l'hélice à une certaine hauteur. Une telle réalisation est possible assez aisément mais il ne faut surtout pas oublier la transmission d'énergie. Pour permettre à la structure d'avancer il faut plus renforcer la base du support du moteur et rajouter de la matière de manière à exercer une force sur des parties plus avant du drone.

Le prototype respecte ces deux contraintes dans une certaine mesure et même s'il n'est pas parfait, en prendre la base pour exemple peut permettre de bonnes améliorations.

3. ESSIEU

Pour permettre aux roues avant de tourner de manière naturelle, il faut utiliser un système d'essieu. Celui-ci permet à un servomoteur seul de diriger les deux roues en transmettant la force via une structure relativement simple. Réalisé sur le prototype, on peut constater son efficacité mais aussi ses limites. En l'occurrence, le mouvement des roues est fortement entravé par la pression de la maquette sur l'essieu.

Malgré la noblesse inhérente à ce matériau, le carton ne permet pas la maîtrise des points de pression sur des petites surfaces, ni une grande flexion et l'essieu ne laisse pas la place à un renforcement structurel très poussé du fait de la mobilité des différentes pièces. Il faudra donc passer à un matériau différent pour mieux exploiter le système d'essieu.