|  |
| --- |
|  |
| Arc-Drone |
| Description de la solution retenue |
|  |
| **Guillaume Jacot & Alexandre Schnegg** |
| **26/01/2013** |

|  |
| --- |
|  |

Contenu

[Introduction 1](#_Toc347131982)

[Résumé du débug des interfaces 2](#_Toc347131983)

[I2C 2](#_Toc347131984)

[Magnétomètre 2](#_Toc347131985)

[PWM 2](#_Toc347131986)

[Attentes actives 3](#_Toc347131987)

[Monitoring 3](#_Toc347131988)

[Envoi des données 3](#_Toc347131989)

[Application labview 3](#_Toc347131990)

[Fonctions 3](#_Toc347131991)

[Fonctionnement 4](#_Toc347131992)

[Acquisition de l'orientation 5](#_Toc347131993)

[Systèmes d'axes des capteurs 5](#_Toc347131994)

[Algorithme d'acquisition de l'orientation 5](#_Toc347131995)

[Régulation 6](#_Toc347131996)

[Gestionnaires 6](#_Toc347131997)

[Démarrage du quadricoptère 6](#_Toc347131998)

[Leds témoin 6](#_Toc347131999)

# Introduction

Le présent document a pour but de résumer le travail effectué sur le quadricoptère afin que le prochain intervenant sache ce qui a été fait. On y trouve les explications concernant la correction des bugs des interfaces ainsi que le fonctionnement du monitoring, de l'acquisition de l'orientation et de la régulation du quadricoptère.

# Résumé du débug des interfaces

## I2C

On a codé une nouvelle fonction permettant la lecture de plusieurs bytes d’un coup. On l’a utilisée pour lire la valeur de tous les axes de l’accéléromètre d’un coup. La lecture multiple ne fonctionne pas avec le gyroscope donc on a laissé comme c’était fait.

On a préféré tester si le bus était libre plutôt que de faire une attente active dans tous les cas.

On a corrigé la fonction d’envoi par I2C. Quand on envoie l’adresse du périphérique souhaité sur le bus I2C, on met le dernier bit à 1 ou à 0 en fonction de si on veut lire ou écrire un registre. Dans notre cas, le dernier bit de la donnée était aussi modifié donc la configuration des périphériques I2C (par exemple la vitesse d’acquisition des données) n’était pas celle que nous voulions.

## Magnétomètre

Une fois qu’une valeur avait été lue, toutes les lectures suivantes nous donnaient les mêmes résultats. On a découvert ceci en faisant des tests avec un aimant :

* Si l’aimant était mis avant le démarrage du programme, les valeurs mesurées étaient grandes et ne bougeaient pas quand on retirait l’aimant.
* Si l’aimant n’était pas mis avant le démarrage du programme, les valeurs mesurées étaient beaucoup plus petites, mais ne bougeaient pas quand on mettait l’aimant.

Ceci était dû au bit LOCK du magnétomètre qui se met à ‘1’ quand on lit la 1re valeur et qui empêche donc l’actualisation des mesures. Celui-ci ne se remettait malheureusement jamais à ‘0’ ce qui empêchait la modification des valeurs.

Pour remédier à ce problème, on a également utilisé notre fonction de lecture multiple par I2C pour lire tous les axes d’un coup. Le magnétomètre remet automatiquement le bit LOCK à ‘0’ quand on a lu toutes les valeurs. Des fois, celui-ci ne se remet pas à ‘0’ automatiquement, on renvoie donc le mode d’acquisition (single ou continuous) au magnétomètre pour forcer sa remise à ‘0’.

On a également modifié le gain, car il était au minimum et le champ magnétique terrestre est très faible (environ 0.5 gauss).

## PWM

La largeur des impulsions variait périodiquement. On a mis pas mal de temps à détecter que ceci venait d’une simple erreur de soustraction dans l’interface du timer. A chaque détection de flanc du signal d’entrée, on sauvegarde la valeur du timer. Ensuite, on calcul la différence entre les 2 mesures. Lorsque le 2e flanc (flanc descendant) est détecté après que le timer soit revenu à 0, il faut prendre la valeur maximale du timer moins la différence entre les 2 valeurs mesurées pour avoir un résultat correct. Le code avait été fait pour une valeur maximale du timer de 65535 pourtant le timer est configuré pour compter de 0 à 62’500 seulement donc les mesures étaient faussées.

Une fois ce problème corrigé, on pouvait encore observer des petites variations dans la mesure du duty cycle alors que le signal d’entrée ne variait pas. Ceci était dû à la façon de sauvegarder la valeur du timer à chaque flanc. En effet, on sauve les valeurs dans des variables globales qui sont ensuite lues par une autre fonction (celle qui fait la soustraction entre les deux valeurs).

Lorsqu’une nouvelle impulsion arrivait (détection du flanc montant), on sauvait la première valeur du timer. Le problème se posait dans le cas où l’autre fonction venait lire, avant le flanc descendant, les deux variables. On avait donc la valeur du timer au flanc descendant de l’ancienne impulsion et au flanc montant de la nouvelle impulsion, ce qui nous donne un résultat faussé.

Pour remédier à ce problème, on met la première valeur dans une variable temporaire et à la fin de l’impulsion, on sauve les deux valeurs dans les variables globales (les deux changent donc en même temps et on a toujours les deux valeurs se rapportant à la même impulsion).

Quand on lit un signal venant de la télécommande, celui-ci a une période de 20 ms, mais l’impulsion ne dure qu’entre 1 et 2ms. Ceci nous donne donc un duty cycle entre 0.05 (5%) et 0.1 (10%). La précision des float n’étant pas très bonne, on a préféré travailler directement avec la valeur en UInt16 (qui représente la soustraction des 2 valeurs du timer). On convertit ensuite cette valeur dans une plage allant de 0 (0%) à 1 (100%).

## Attentes actives

Nous avons remplacé toutes les attentes actives par des machines d’états pour gagner en rapidité car rien que le capteur de pression prenait environ 70ms avant de renvoyer une valeur par exemple.

# Monitoring

## Envoi des données

Les données sont envoyées à l'application labview par le bai d'une communication série sans fil sous forme de chaînes de caractères.

La trame a le format suivant:

1. Start
2. Capteur
3. Type de donnée (entier ou flottant)
4. Données
5. Stop

Lorsque le capteur nécessite l'envoi de plusieurs valeurs (par ex: 3 axes pour l'accéléromètre), les données valeurs sont séparées par le caractère '|'.

La génération des trames se fait avec le l'utilitaire *generateTrame.h.*

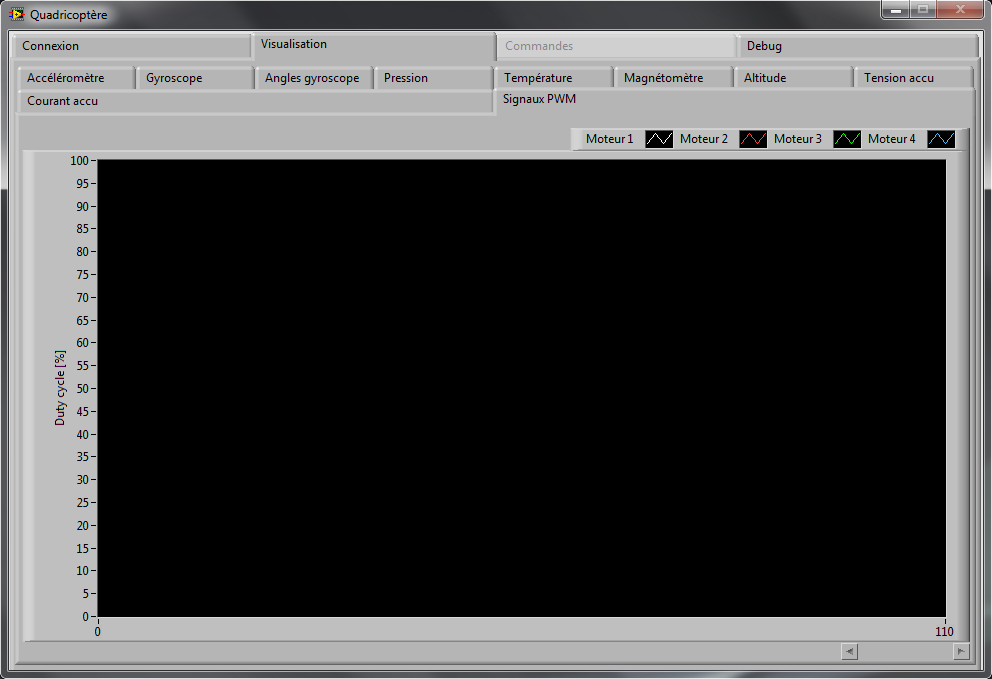
## Application labview

### Fonctions

Voici une liste des capteurs dont les valeurs peuvent être lues par l’application Labview :

* Accéléromètre (3 axes)
* Gyroscope (3 axes)
* Magnétomètre (3 axes)
* Pression
* Température
* Altitude (pas encore fonctionnel)
* Tension de la batterie
* Courant consommé
* Orientation du quadricoptère (3 axes, valeurs après filtrage)
* Signaux PWM (commande des 4 moteurs)

Voici à quoi ressemble l’application :



Fonctions des 4 onglets principaux :

* **Connexion** : permet de sélectionner sur quel port COM est connecté le récepteur RF, de s’y connecter/déconnecter et de quitter l’application.
* **Visualisation** : permets de visualiser un graphique par capteur (chaque capteur est sur un onglet séparé) représentant l’évolution des mesures de celui-ci (plusieurs courbes sur le même graphique pour les capteurs avec plusieurs axes ou pour les signaux PWM des 4 moteurs).
* **Commandes** : permets d’envoyer des commandes au quadricoptère. Cet onglet n’est pas encore fonctionnel.
* **Débug** : permets de visualiser tout ce qui a été reçu par RS232 et détecter d’éventuelle erreur dans les valeurs envoyées par le quadricoptère. On y voit également la dernière commande reçue ainsi que l’état de l’application.

### Fonctionnement

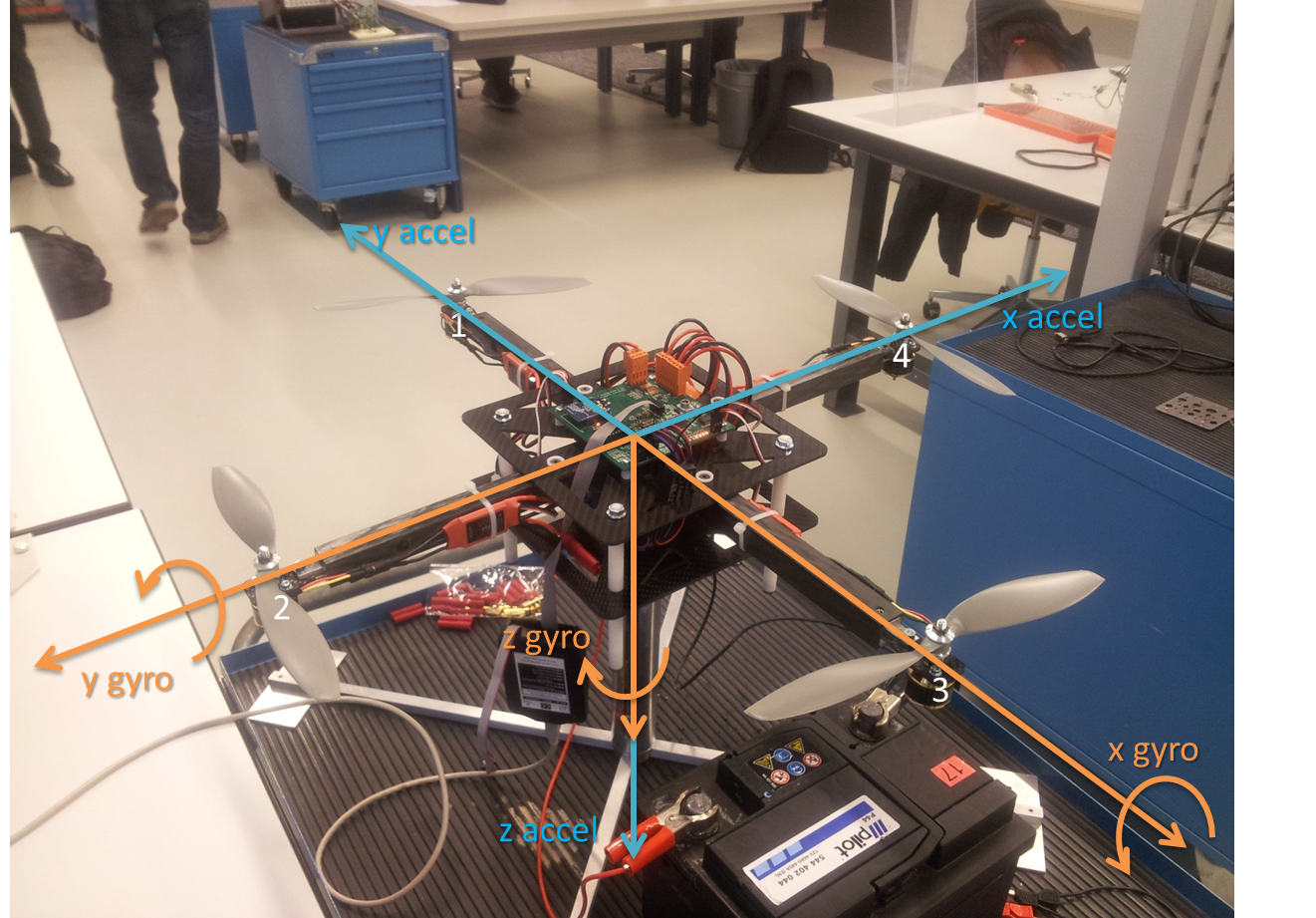
L’application est basée sur une machine d’état. On passe d’abord par un état d’initialisation, puis par la connexion au RS232 et enfin on va lire les données reçues en boucle. Un état est aussi prévu pour la déconnexion du RS232.

Chaque fois qu’une trame est reçue, l’application va la placer dans une file d’attente. C’est une deuxième boucle qui tourne en parallèle qui va récupérer cette trame, la décoder et ajouter les valeurs qu’elle contient sur le bon graphique.

A chaque fois que la machine d’état passe à l’état « Init », « Connect » ou « Disconnect », une notification est envoyée à une troisième boucle qui tourne en parallèle. Celle-ci s’occupe d’effacer les graphiques et d’activer/désactiver certaines fonctions selon l’état de l’application.

# Acquisition de l'orientation

## Systèmes d'axes des capteurs



## Algorithme d'acquisition de l'orientation

Premièrement, les données brutes des capteurs sont transformées en grandeurs physiques (g pour l'accélération et deg/s pour le gyroscope. La dérive du gyroscope est compensée à ce moment-là.

Le vecteur d'accélération est ensuite transformé en angles dans le référentiel du gyroscope. La valeur de ces angles est convertie en degrés.

Dès lors, les valeurs sont filtrées par un filtre du premier ou du second ordre, à choix. Les paramètres du filtre sont réglés dans les définitions du fichier *mOrientationProcessing.h*. On peut régler la fréquence d'échantillonnage (kDt), le Tau du filtre (kTau) et les coefficients du filtre (kA et kB, respectivement pour le passe-haut (gyroscope) et le passe-bas (accéléromètre)).

# Régulation

Tous les paramètres de la régulation sont réglables dans *mRegulation.h.*

La régulation fonctionne sur le principe suivant : une puissance constante est appliquée à tous les moteurs. La régulation va calculer la puissance à rajouter ou à enlever. L’erreur est calculée sur les angles xz et yz, qui correspondent aux deux axes d'inclinaison possible; la régulation est calculée pour ces deux angles. La puissance à appliquer au moteur qui sont sur un axe est calculée comme ceci: pour le premier on additionne la puissance calculée par la régulation à l'offset, pour l'autre on la soustrait.

Chaque régulateur (P et I) est implémenté séparément dans une fonction.

Il est nécessaire de régler l'angle maximum que peuvent avoir les angles d'orientation. Lorsque le quadricoptère et sur le support il ne peut s'incliner que de 20°.

# Gestionnaires

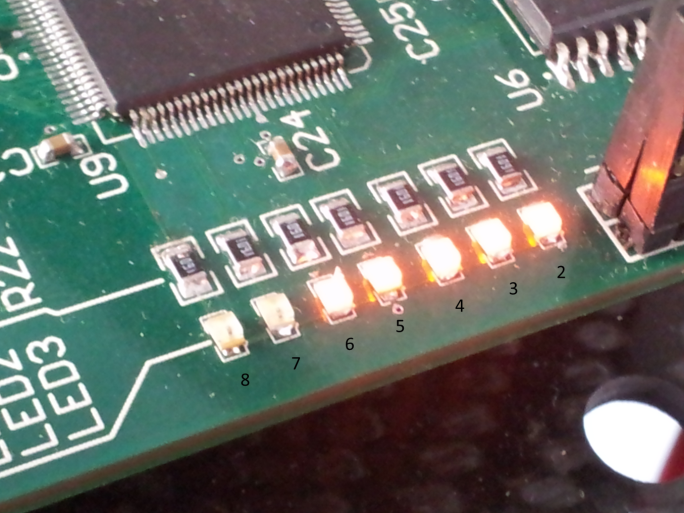
La régulation devant être exécutée à un temps régulier précis, le code doit être exécuté dans une interruption. Pour se faire, nous avons créé trois fonctions *execute* spéciale, *gInput\_ExecuteInt*, *gCompute\_ExecuteInt*, *gOutput\_ExecuteInt* destinées à être appelée dans l'interruption *iPit\_INTPit1*. Les autres fonctions *execute* sont appelées dans la boucle principale du *main*.

# Démarrage du quadricoptère

Quand on branche l’alimentation, les drivers moteurs se configurent en fonction du signal PWM qu’ils reçoivent. Ils considèrent que celui-ci représente 0%. Ceci permet d’utiliser correctement toute la plage de puissances. On doit donc envoyer un signal PWM constant représentant le 0% au démarrage du programme avant de pouvoir faire tourner les moteurs à la vitesse souhaitée.

Par mesure de sécurité, le quadricoptère ne démarre pas tout de suite lorsque le programme est démarré. Il faut maintenir les joysticks de la télécommande en position 0 (en bas à gauche) pour que les moteurs démarrent. Sitôt qu’un joystick est relâché, les moteurs s'arrêtent.

# Leds témoin

1. Allumée après le setup des modules
2. Allumée après l'open des modules
3. Allumée après le setup des gestionnaires
4. Clignote à chaque cycle du main
5. Clignote à chaque appel de l'interruption *iPit\_INTPit1*
6. Témoin d'activation des moteurs