Université de Franche-Comté

Rapport de Stage

Licence Sciences pour l'ingénieur, 3ème année

Étude des centrales inertielles libres pour microdrones

Mohamed Hage Hassan

Institut FEMTO-ST

Remerciements

Je tiens tout particulièrement à remercier mon maître de stage M. Jean-Michel FRIEDT pour son accueil chaleureux, son encadrement, ainsi que pour son aide durant ce stage.

Je tiens également à remercier M. Émile CARRY pour son aide précieuse, ainsi que M. David RABUS pour les renseignements qu'il m'a fournis et son support, et M. Fabian MINARY, M. Gwenhael GOAVEC, M. Berenger OSSETE pour leur soutien technique.

Un grand merci à M. Jean-Philippe CULAS de la société CM-Drones pour son aide et appréciations, et pour avoir fourni la plateforme de travail et périphériques associés.

Je souhaite également remercier toute l'equipe de Temps-fréquence pour son acceuil, ainsi que toutes les personnes qui se sont intéressées au projet.

Résumé

Les drones représentent des grandes opportunités dans le domaine de la recherche, grâce à leur puissance et mobilité. Le développement des centrales inertielles dans les dernières années, et surtout celles implémentés avec des plateformes opensource, était un point-clé d'innovation. En effet, l'équipe Cosyma au sein du département de Temps-Fréquence à FEMTO-ST, recherche l'utilisation créative des vecteurs de vol : Donner la capacité aux drones de sonder des capteurs passifs avec une électronique compacte embarquée, et utiliser les données captées par les détecteurs pour diriger le drone.

La solution envisagée était d'utiliser un calculateur externe, dans un premier temps, pour envoyer des commandes à la centrale inertielle, et diriger le drone après la prise en compte des données du capteur embarqué, suite à une étude approfondie du système de la centrale inertielle.

Nous avons réussi à développer une application tournant sur le microcontrôleur STM32F4 et contrôlant le drone, suite aux nombreuses simulations pour vérifier son comportement.

Malgré le manque de l'interrogateur des capteurs passifs, requis pour l'objectif du stage, qui est toujours en cours de développement, on a pu utiliser un capteur externe (accéléromètre) pour commander le drone et changer sa position.

Ce résultat laisse la possibilité d'exploiter le travail accompli pour de nombreuses applications.

Table des matières

1. Introduction

Le domaine des drones ne cesse de s'améliorer. Son émergence tient à la convergence de plusieurs technologies : moteurs compacts et puissants, batteries de grande autonomie et légères, et centrales inertielles intégrant des composants MEMS à des performances acceptables, avec un système de guidage basé sur GPS et des calculateurs puissants.

Dans le contexte de mesure de capteurs passifs interrogeables sans fil, le département Temps-Fréquence (T-F) de l'institut FEMTO-ST s'intéresse à la possibilité d'automatiser la procédure de mesure de capteurs distribués à l'aide d'un microdrone. C'est un axe de recherche de l'équipe Cosyma au sein de T-F qui porte sur l'utilisation de transducteurs à ondes élastiques comme cibles coopératives de RADAR.

Ces électroniques de mesures sont suffisamment légères et compactes pour permettre leur intégration comme des charges utiles pour des drones.

L'objectif du stage est d'atteindre la capacité des drones d'exécuter un scénario avec la possibilité de fournir une autonomie de décision et d'asservir le mouvement du drone à l'aide d'un signal recueilli au cours de l'interrogation des capteurs.

La première étape de ce stage a consisté à formuler le cahier des charges nécessaire pour l'élaboration des objectifs principaux, puis une documentation pour comprendre le fonctionnement des différentes plateformes matérielles et logicielles.

Une étude préliminaire est nécessaire pour faciliter la compréhension de la méthodologie suivie afin de réaliser des objectifs définis dans le cahier des charges, et la mise en place des outils nécessaires pour le développement lié.

On établit finalement la méthode de contrôle utilisée, avec une analyse des sources exploitées, les différents tests effectués et les problématiques rencontrées, ainsi que des évolutions possibles suite au travail effectué.

2. Institut FEMTO-ST
Laboratoire Temps-Fréquence : présentation

L'institut FEMTO-ST (Franche-Comté Electronique Mécanique Thermique et Optique Sciences et Technologie) fondé le 1er janvier 2004 par la fusion de cinq laboratoires de recherche Franc-comtois, est un institut mixte de recherche associé au Centre National de la Recherche Scientifique (CNRS), à l'Université de Franche-Comté (UFC), à l'École Nationale Supérieure de Mécanique et des Microtechniques (ENSMM) et à l'Université de Technologie Belfort-Montbéliard (UTBM).

La mission du laboratoire est d'associer les sciences et les technologies de l'information avec les sciences pour l'ingénieur. Les thèmes de recherche au sein du laboratoire sont diversifiés, ils couvrent l'optique, l'acoustique, les micro-nanosystèmes, le temps-fréquence, l'automatique, la mécatronique, et l'énergétique.

Il est structuré en six départements qui traitent ces thématiques : AS2M (Automatique et Systèmes Micro Mécatroniques), Énergie, Mécanique appliquée, MN2S (Micro Nano Sciences et Systèmes), Optique et Temps-Fréquence.

FEMTO-ST compte aujourd'hui plus de 700 membres, départements scientifiques, services communs et direction confondus.

Le laboratoire de Temps-Fréquence est un département de l'institut FEMTO-ST, sa mission est de mener la recherche dans le domaine des résonateurs et oscillateurs, des applications capteurs et de la métrologie des sources de fréquences.

Le département pluridisciplinaire combine plusieurs technologies et domaines scientifiques comme l'électronique, l'acoustique, MEMS, photonique, traitement du signal, physique atomique.

Le département Temps-Fréquence est un acteur majeur dans les réseaux d'excellence français LabEX FIRST-TF et ACTION : ce dernier finance la présente étude.

3. Étude des besoins

L'objectif est de sonder la réponse des capteurs par un vecteur de vol mobile, tel que les drones. Sachant que ces capteurs sont statiques, nous concentrons nos efforts sur les multicopters, capables d'un vol stationnaire au cours d'une mesure. Ce type de platforme est intrinsèquement instable, il nécessite une centrale inertielle pour garantir son vol.

Étant donné que notre étude s'intéresse au contrôle des centrales inertielles par des commandes externes, cette étude ne porte pas sur le développement des lois de commandes mais sur un transfert des consignes . On trouve alors que seule une centrale inertielle implémentée au moyen des outils opensource répond à nos exigences de développements.

Une multitude de centrales inertielles existe, on trouve que celles fabriquées par 3DR, comme la platforme matérielle Pixhawk, basées sur architecture STM32, répondent bien à nos besoins, contrairement à celles fournies par DJI, qui malgré une annonce d'opensource semblent limiter l'accès aux codes sources d'algorithmes de contrôle du drone.

Le pilote automatique fourni par 3DR peut supporter 2 logiciels de contrôle libres tournant sur STM32 : Ardupilot et PX4. Nous avons choisi d'utiliser la seconde solution, vis-à-vis de :

- Logiciel de contrôle opensource polyvalent et robuste qui offre plus de fonctionnalités que Ardupilot.
- L'environnement tourne sous le système d'exploitation temps réel multi-tâche NuttX compatible avec les standards **POSIX**.
- Développement très actif et documentation abondante .
- Plusieurs exemples de communication avec le logiciel PX4 sont fournis, avec une bonne documentation.

On peut établir le cahier des charges suivant :

- Se familiariser avec le logiciel PX4 implémentant les centrales inertielles sur Pixhawk.
- Utilisation d'un calculateur externe pour définir les ordres de la centrale inertielle (Pixhawk), soit d'une application tournant sur NuttX dans la Pixhawk.
- Prendre en compte les coordonnées captées de la centrale inertielle et utiliser une loi de commande pour diriger le drone.
- Utiliser les études effectuées pour permettre au drone de réaliser un scénario complètement autonome à l'aide d'un correcteur automatique, qui s'adapte aux données acquises.

4. Autopilote Pixhawk avec PX4 et Protocole MAVLink

Le pilote automatique Pixhawk est un module de contrôle automatique de drone conçu par des chercheurs au sein du laboratoire de vision de l'institut fédéral Suisse de technologie à Zurich[1] en collaboration avec le laboratoire des systèmes autonomes[2] et l'entreprise 3DRobotics[3]. C'est principalement un microcontrôleur **STM32F427** avec un coeur **Cortex-M4F**, comportant plusieurs capteurs comme les gyroscopes, accéléromètres, boussole et un baromètre[4] et un coprocesseur STM32F103 de sécurité.

4.1. Analyse de l'architecture du système PX4

Le système PX4 est très modulaire[A1][29], il est formé de 4 couches essentielles :

- Les pilotes qui contrôlent les capteurs de positions, les accéléromètres, le GPS, la boussole, les moteurs.
- Le système temps réel NuttX, compatible avec le standard POSIX.
- Un système de communication interprocessus du type uORB[A2] (micro Object Request Broker) qui implémente une méthode de publication et souscription : les processus communiquent au travers d'un bus logiciel qui dirige les demandes et les réponses entre eux.
- Le système de contrôle du vol de haut niveau, qui est un ensemble d'algorithmes pour le contrôle de position, d'attitude et d'altitude, de navigation. Chaque fonctionnalité constitue une application séparée, et l'ensemble de ces éléments communiquent entre eux en utilisant l'architecture uORB.

Cette architecture sera reprise en détails dans la Fig. 7, auquel le lecteur peut se référer dès à présent.

4.2. Introduction à NuttX

NuttX[5] est un système d'exploitation temps réel compatible avec les standards POSIX et ANSI pour les microcontrôleurs et les systèmes embarqués. Il comporte un micro-noyau avec un support de multitâches, des threads POSIX, un ordonnanceur de type **round-robin**, avec un support d'une variété de microcontrôleurs.

L'utilisation de ce système est bien justifiée : la réaction du PX4 aux changements aérodynamiques, ainsi que le changement d'attitude se fait d'une manière déterministe, et doit se faire en temps réel. La compatibilité de NuttX avec les standards POSIX permet le développement rapide et une réutilisation facile du code.

4.3. Protocole MAVLink

MAVLink (Micro Air Vehicle Link) est un protocole permettant la communication entre une centrale terrestre et un drone, ou entre plusieurs drones. Ce protocole est encore utilisé pour un relais entre un microcontrôleur externe et la Pixhawk.

MAVLink encode des structures qui peuvent représenter la position en GPS, l'altitude, l'état du système et de la batterie, et surtout envoyer de commandes vers une ou plusieurs plateforme ou véhicules sans pilotes.

Ce protocole est créé par le groupe de développement de PX4, et utilisé dans de nombreuses applications. La raison de son utilisation revient à ses performances, sa documentation bien élaborée[6][7] et sa facilité d'utilisation, et d'intégration.

Il est implémenté comme une application séparée dans le firmware PX4, qui traite et envoie les ordres reçus vers les autres applications qui gèrent la position, l'altitude et le contrôle du drone, à travers la communication uORB.

5. Étude préliminaire

Dans notre étude, on ne modifie pas les lois de commandes de la centrale inertielle, mais on désire réaliser un moyen de communication et envoyer des ordres à cette centrale pour qu'elle fasse un scénario avec une capacité de décision en se basant sur les données acquises d'un interrogateur ou d'un capteur simulant ce RADAR chargé de trouver le capteur passif.

5.1. QGoundControl et configuration du système PX4

La première étape pour la mise en place du système PX4 était de configuer la centrale inertielle en cohérence avec l'architecture matérielle du drone, en suivant la documentation [8] pour le développement.

QGroundControl[9] est un logiciel pour contrôler un véhicule sans pilote à partir des coordonnées GPS. Il permet de flasher le système PX4 sur la Pixhawk, la mise à jour des paramètres tel que coefficients des boucles d'asservissements, la configuration des différents capteurs de position, de GPS , d'altitude et de mise à niveau, les paramètres du vol, de la commande RC, du calibrage PID, des sensibilités des commandes RC envoyées, des modes de contrôle. Et surtout, il permet de définir la configuration d'une mission pour le vol autonome du drone.

Vis-à-vis à l'étude du cahier des charges, on doit commander le drone à partir d'un capteur externe pour faire une série de déplacements complètement autonomes. QGroundContol ne peut pas effectuer cette fonctionnalité, sachant qu'on a besoin d'un traitement en temps réel des données du capteur, pour rediriger le drone en se basant sur ces données.

Deux approches sont envisagées pour transférer le contrôle depuis le sol vers une

application embarquée :

D'une part, on peut intégrer l'application de la lecture du capteur dans la plateforme PX4 tournant sur la centrale inertielle (Pixhawk), comme une application de NuttX. Cette méthode a l'avantage de réduire la connectique, et l'utilisation d'un calculateur séparé. Par contre, la méthode de mesure de capteur est semblable à celle d'un RADAR impulsionnel, qui implique certaines étapes de traitement de signal respectant des contraintes temporelles dures. Le programme de contrôle et le fonctionnement de PX4 ne doivent pas rentrer en conflit, ce qui peut entraîner une perturbation dans la stabilité du vol.

Dans la seconde approche, qu'on va développer dans ce qui suit, un calculateur externe est utilisé pour séparer l'interrogation du capteur de PX4 tournant sur la Pixhawk. Cette méthodologie élimine les problèmes liés aux concurrences des ressources de calculs, mais nécessite l'introduction d'un moyen robuste de communication entre le microcontrôleur et la Pixhawk, qui est le protocole MAVLink.

5.2. Procédure du contrôle de la Pixhawk

La platforme PX4 offre un mode de contrôle externe (Offboard)[10] qui permet à un microcontrôleur de commander le drone.

La procédure du contrôle de la pixhawk est illustrée par la figure 1.

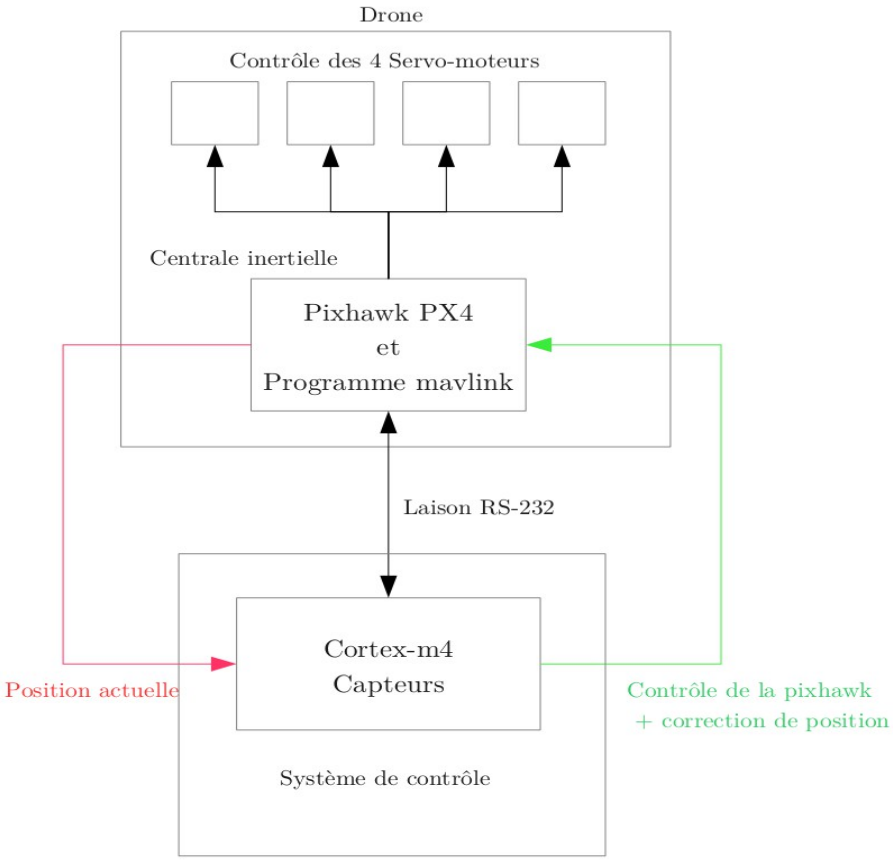


Figure 1: Contrôle du drone à partir d'une STM32F4 et le protocole MAVLink

En premier temps, le code contrôlant l'interrogateur des capteurs passifs tourne sous STM32F4-discovery. La compatibilité de ce code avec celui du contrôle du drone nécessite l'utilisation d'une STM32F4 comme un microcontrôleur externe pour le contrôle du drone avec le mode offboard.

5.3. Mise en place des outils nécessaires pour le développement

Le développement sous microcontrôleur du type STM32 (coeur Cortex-M3 ou 4) nécessite l'installation d'une toolchain spécifique[12] ainsi que l'utilisation d'une bibliothèque bas-niveau libopencm3[13] et les outils stlink[14] pour flasher la STM32.

Le STM32 représente une nouvelle platforme de développement, la prise de main par l'étude des exemples fournis dans les exemples de la bibliothèque libopencm3[15] et les TP STM32[16].

La présence des outils de simulation dans PX4 présente un avantage pour le développement sur PX4 sans prendre en compte le système réel. La simulation (Figure 2) permet le prototypage rapide de la méthode de contrôle de la Pixhawk, ce qui permet de réduire considérablement le temps des tests effectués et celui du déverminage et éliminer la possibilité et les risques de la casse du matériel.

Figure 2: Capture d'écran de l'interface graphique du simulateur de drone.

Une explication de mise en place de l'environnement de développement est élaborée[N3].

6. Élaboration de la méthode de contrôle

Les développeurs de la platforme PX4 ont fourni un exemple simple[11] pour contrôler un drone à partir d'un système *nix (ou compatible avec le standard POSIX). Cet exemple fournit un excellent point de départ pour le travail que nous cherchons à faire.

6.1. Analyse de la structure interne de l'interface C/UART

L'application C/**UART** interface utilise des structures C++ pour construire des messages MAVLink et les envoyer à travers une connexion **RS-232**/UART.

L'application comporte 3 couches (Figure 3) : couche haut niveau des commandes et des instructions à envoyer au drone, une deuxième couche qui utilise l'API de la couche la plus basse (la couche de liaison série asynchrone) pour lire les messages MAVLink, et à partir des commandes fournies, construire des messages et les renvoyer à travers le port série/UART vers le drone.

Les 3 couches représentent 3 niveaux séparés en C++ programmées selon une structure orientée-objet.

Au moment d'exécution, deux threads de la lecture et d'écriture sont lancés avant l'envoi des commandes de changement de position.

On envoie la première commande, qui commande le drone à se mettre en mode offboard, puis on exécute les commandes spécifiques, telles que le changement de position, de vitesse, de la position angulaire.

À la fin d'issue des commandes, on envoie une commande de désactivation du mode offboard, le drone retourne à son mode précédent.

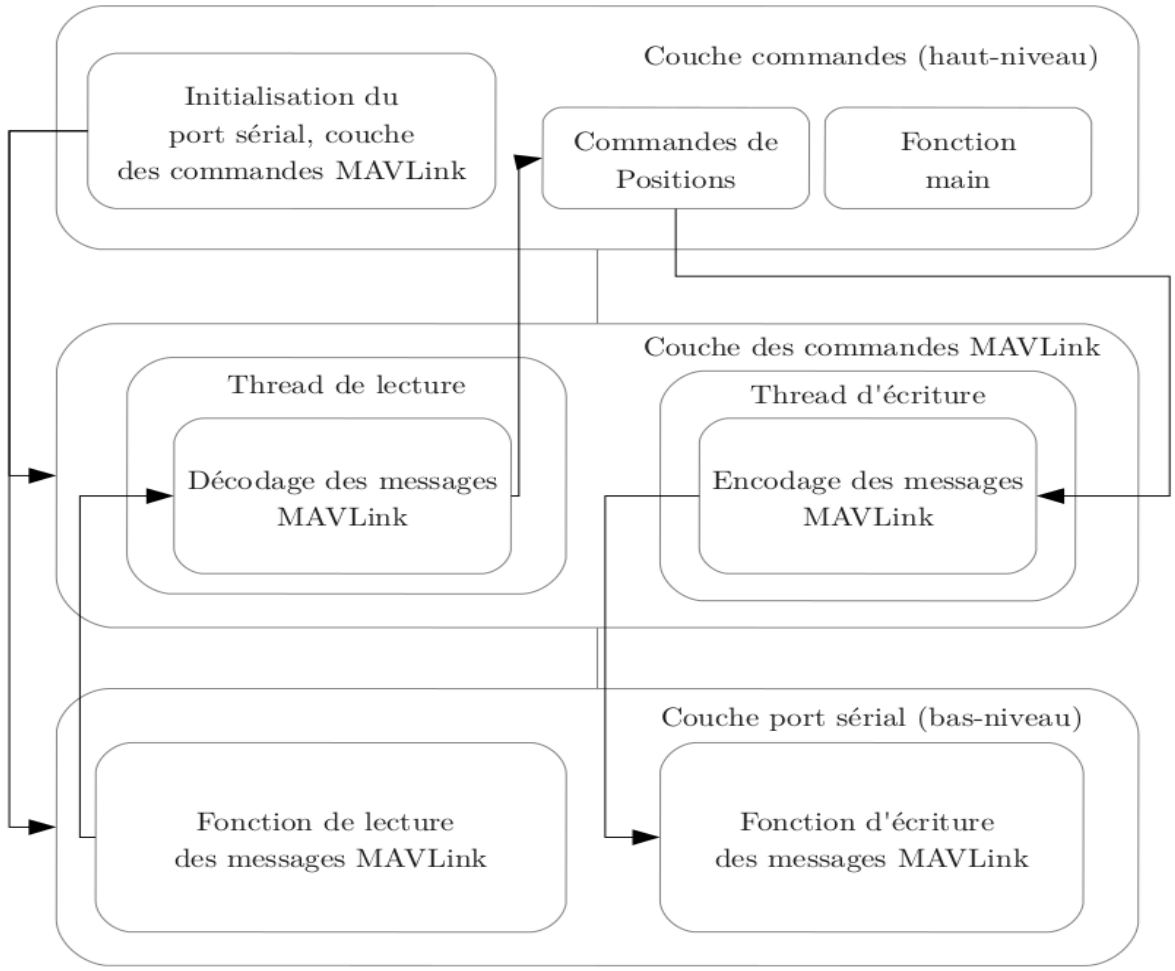


Figure 3: Schéma simplifié de la structure de l'interface C/UART

Un schéma détaillé du programme avant modification est élaboré[A3].

6.2. Contrôle direct du drone en simulation

La compilation de l'interface C/UART résulte dans le programme "mavlink control" L'exécution de ce programme sur la simulation est pour pour objectif tester la communication et le mode de contrôle Offboard, et nécessite une liaison[A6] convertissant le flux de données asynchrones vers une liaison IP/UDP : **netcat** se charge d'effectuer cette conversion et de faire communiquer le programme de commande (émettant des instructions sur une liaison asynchrone) avec le simulateur (qui accepte des instructions sur socket IP en liaison UDP).

Cette méthode est très efficace par rapport aux tests sur le drone réel, et constitue une base solide pour le développement de l'application sur la STM32F4, sachant que le comportement de la simulation est identique au comportement du drone réel.

6.3. Portage de l'interface C/UART vers STM32F4

6.3.1. Problématiques de l'absence d'ordonnanceur sur un système Bare-metal

Un ordonnanceur constitue un élément essentiel dans un système d'exploitation moderne. C'est un programme qui contrôle l'exécution des différents programmes et peut assurer cette exécution de façon concurrente dans une architecture multitâches.

À l'aide de l'ordonnanceur, on peut définir des processus, et des sous-processus ou “ threads”. Une des implémentations des threads est le POSIX threads, qui est utilisée principalement sur Linux.

Dans un système Bare-metal, le déroulement des instructions d'un programme se fait d'une manière purement séquentielle, à cause de l'absence d'un ordonnanceur, ce qui rend impossible l'utilisation des threads et l'exécution en parallèle des tâches.

L'interface C/UART utilise des pthreads pour les tâches de lecture et d'écriture, des ports séries dépendantes d'un système de type *nix, ce qui nécessite une modification pour l'implémentation sous le microcontrôleur STM32F4-discovery.

6.3.2. Méthodes essayées

On a recours à trois choix possibles : soit l'addition d'un ordonnanceur, ou un ordonnanceur avec un système de threads simple et remplacer les pthreads, soit une boucle infinie avec une machine à états.

Pour implémenter chacune de ces méthodes, on a recours aux étapes de réduction de l'interface C/UART expliquées dans la section 6.3.3 (Élimination des couches C++, des pthreads, et des fonctionnalités dépendantes d'un système d'exploitation).

Ordonnanceur :
Dans les exemples de la bibliothèque libopencm3[15], on peut trouver un ordonnanceur simple[17] sans threads. Celui-là est codé partiellement en assembleur, et destiné pour un fonctionnement sur la carte STM32vl-discovery. On a modifié cet ordonnanceur pour fonctionner sur la carte d'évaluation STM32F4-discovery.

Cette méthode a été abandonnée après puisqu'on a trouvé que l'ordonnanceur manque de **préemption** qui est une technique nécessaire pour solliciter les différentes fonctions périodiquement. Le codage de telles fonctionnalités revient à reimplementer un système d'exploitation, d'où la tentative de passage à un **RTOS** minimal.

Atomthreads RTOS :
Atomthreads[18] est un système d'exploitation minimal, présentant un ordonnanceur avec une fonctionnalité de type round-robin, avec des threads minimaux, et une documentation

abondante et des exemples. Ce type de système se présente comme un noyau avec des exemples spécifiques pour chaque architecture (Cortex-m, AVR..).

On a inséré la version de l'interface C/UART réduite comme une application avec le noyau de Atomthreads, puis on a procédé à remplacer les fonctions pthreads avec des threads de Atomthreads RTOS.

Cette méthode est aussi abandonnée, puisqu'on a eu une corruption de mémoire dans la tâche responsable de la lecture des paquets MAVLink. Nous n'avons pas réussi à retrouver la cause de ce bug, à cause des difficultés liées au déverminage des systèmes Bare-metal embarqués.

Une Itération infinie :

Avec cette méthode, on élimine toute fonction qui utilise les pthreads, et on remet le programme entier en exécution linéaire.

6.3.3. Méthode d'une itération infinie

Cette méthode (représentée par la figure 4) consiste à exécuter l'ensemble des instructions du programme dans une boucle infinie avec une machine à états et transition entre états après l'atteinte d'une durée prédéfinie.

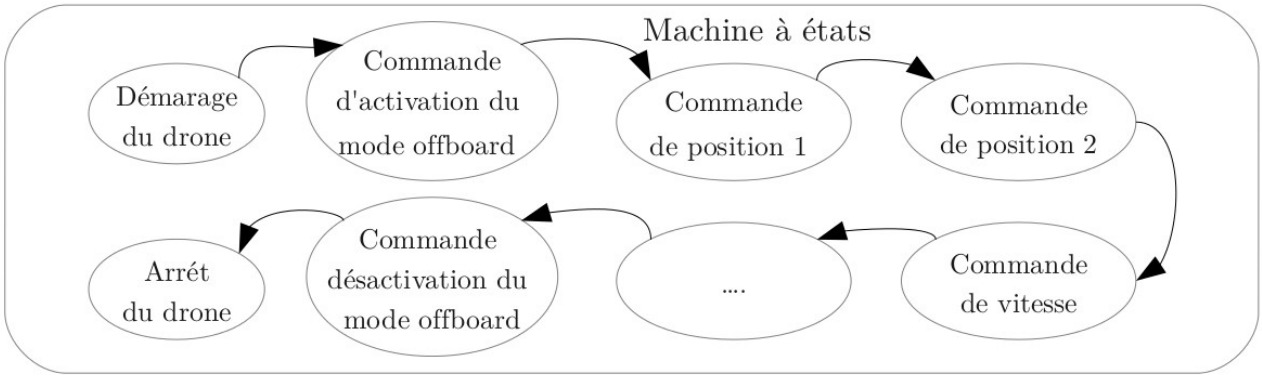


Figure 4: Machine à états représentant la séquence d'execution d'un scénario de contrôle Offboard.

Un schéma détaillé expliquant l'architecture de cette modification[A4] ainsi qu'une procédure pour la mise en place de cette méthode sont élaborées :

Élimination des couches C++

On voit que l'architecture[A3] de l'interface C/UART est séparée en couches C++ avec des variables privées. Cette implémentation rend difficile l'accès aux différentes variables depuis des fonctions de bas-niveau.
La réimplémentation des différentes fonctions, tel que les commandes de déplacement, de l'interface de communication **USART**, de l'interface des commandes d'autopilot en C

facilite l'accès direct de ces fonctions.

Élimination des pthreads

Dans la couche d'interface d'autopilot, on utilise deux pthreads pour la lecture et l'écriture. Ces 2 threads tournent en permanence pour lire depuis la pixhawk, interpréter les différents messages reçus, puis renvoyer des commandes avec l'écriture au port série, à travers le protocole MAVLink.

On élimine cette fonctionnalité pour la remplacer avec des fonctions C de lecture et d'écriture, et ces fonctions seront appelées automatiquement dans l'itération infinie.

Fonctionnalités dépendantes d'un système d'exploitation

Toutes les fonctionnalités et les fonctions dépendantes d'un système d'exploitation comme la fonction printf, les mécanismes de gestion des erreurs (catch and try), gestion des arguments de la commande en ligne, ainsi que la manipulation des ports séries sont enlevées, et ils sont remplacées par des fonctionnalités compatibles avec une exécution sur systèmes embarqués, en particulier au travers de la gestion des interruptions, tel que décrit ci-dessous :

Insertion des interruptions

La STM32F4, ainsi que les autres microcontrôleurs STM32 possèdent des fonctions de manipulation d'interruptions. Ces fonctions sont exécutées d'une manière asynchrone, elles peuvent gérer des **timers**, des USARTs, des ports **GPIO**s.

Pour la méthode de la lecture, qui remplace l'appel système read(), on implémente une fonction bas-niveau qui gère les interruptions USART, et sert à la concatenation des paquets MAVLink dans un tampon temporaire. Ce dernier est transféré vers la séquence du programme principale à chaque fois qu'un paquet complet est acquis.

La méthode d'écriture vers le port série, initialement un appel système write(), est remplacée par une autre qui construit un message MAVLink, puis envoie ce paquet à travers l'USART.

L'interruption timer, est utilisée pour faire une référence du temps en seconde. Ce timer est configuré pour lancer une interruption chaque 250 ms : on incrémente un compteur chaque 4 interruptions pour servir comme une base de temps en secondes.

La configuration de l'horologe du timer est celle de l'horloge interne du STM32F4, avec une fréquence de 16 MHz. Cette fréquence est suffisante pour l'envoi des messages MAVLink, à une fréquence supérieure à 4 Hz, l'intervalle de temps minimum entre deux transactions spécifié par le protocole MAVLink pour prendre en compte les messages de commandes.

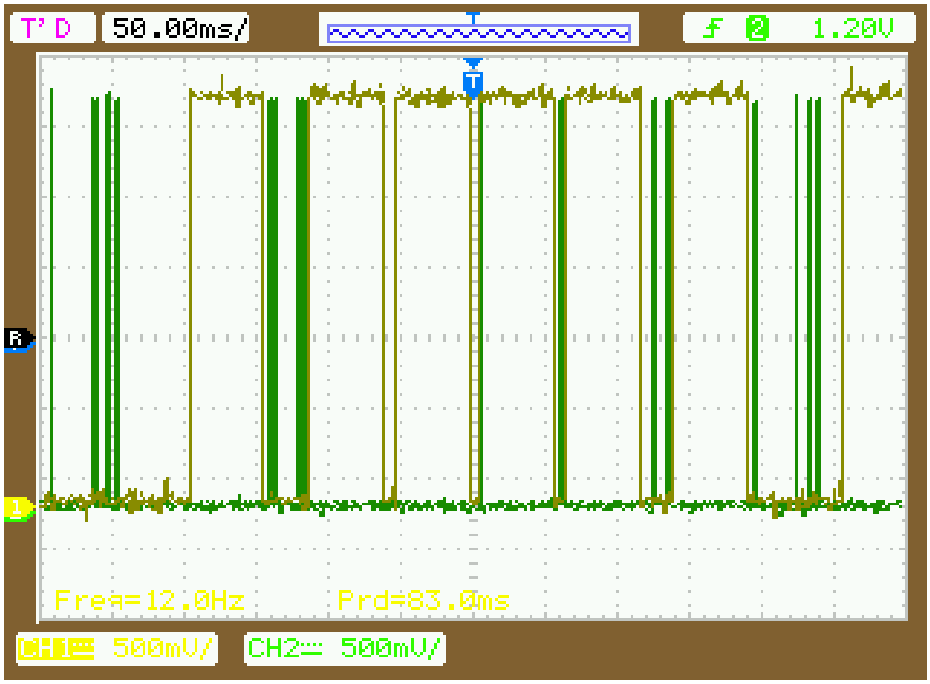


Figure 5: Évolution de l'écriture et de la lecture des messages MAVLink au fonction du temps.

La démarche séquentielle du programme avec une machine à états crée une alternance entre l'écriture d'un message MAVLink de commande, et la réception d'un ou plusieurs messages entre chaque écriture, indiquées par la figure 5.

Modification des méthodes de décodage et d'encodage

La méthode de décodage originale des messages MAVLink était bloquante sur les messages, et s'exécutait comme un thread. Cette fonction a été modifiée pour s'exécuter d'une manière séquentielle et non bloquante, avec une addition d'un délai après lequel la fonction est reinitialsée.

La fonction d'encodage des messages, envoie les messages vers la fonction responsable du port série, avec une fréquence de 4 Hz. Cette fonctionnalité était enlevée puisqu'elle est accomplie automatiquement avec la machine à états avec l'itération infinie.

Couplage avec syscalls et NewLib

Le programme tournant sur la STM32 est compilé avec NewLib. Cette addition nécessite la définition des fonctions qui servent comme « stub » (des points d'entrées vers les appels systèmes non définis). Celles-ci sont ajoutées comme fonctions à parts et linkées avec le programme principal.

Addition de l'acquisition de la position initiale

Les ordres de commande du drone nécessitent une définition de sa position initiale. Cette position est sa position actuelle en pratique. On a ajouté une fonction qui prend en compte les coordonnées locales du drone, ainsi que sa vitesse, et celle-ci est appelée une seule fois au début de séquence de vol.

Méthodes de déverminage

Ces méthodes sont utilisées pour faire face aux divers problèmes rencontrés pendant le développement :

Interception du trafic pour contrôler la nature des messages MAVLink envoyés, vérification l'intégrité de chaque message, utilisation de St-util[14] et **gdb** pour trouver les corruptions de mémoire possibles et les raisons du blockage du programme pendant la phase de développement, et d'un deuxième USART pour remplacer la fonction « printf » de stdio.

Addition de la mise en marche et coupure des moteurs

La mise en marche des moteurs du drone réel et sa désactivation à distance, nécessitent une méthode spécifique. En utilisant un message déjà existant dans l'ensemble des messages MAVLink défini par défaut[19], on peut accomplir le résultat souhaité. L'activation des moteurs est la première commande à exécuter, et la désactivation est établie uniquement quand le drone est au niveau du décollage.

Essais directs sur le drone virtuel

Les essais sur le drone virtuel étaient en passant par la liaison netcat[A7] chargée de convertir une liaison série asynchrone en liaison compatible internet (IP/UDP) pour établir à chaque addition ou modification d'une fonctionnalité essentielle du programme, pour garantir son intégrité et son bon fonctionnement, et éviter des bugs potentiels qui peuvent perturber le contrôle du drone en vol.

6.4. Amélioration du programme et tests réels

Avant le déroulement des tests réels, on a amélioré le programme en ajoutant la compatibilité de l'interface avec un système de type *nix, pour garder une fonctionnalité semblable à l'interface originale. Ainsi, un algorithme sera testé facilement sur un ordinateur personnel (simulateur) avant de passer sur microcontrôleur (drone).

Cette addition a été réalisée en ajoutant un makefile séparé pour une compilation sur le système de type *nix, des conditions présentes le long du code source pour séparer le code tournant sur la STM32F4 de celui qui tourne sur un PC.

Sachant que la machine à états et les fonctions de décodage et d'encodage des messages

MAVLink peuvent tourner sur plusieurs architectures, on n'ajoute que la partie d'initialisation du port série spécifique au système *nix.

Cette partie reprend l'utilisation des appels système déjà éliminés dans la partie STM32F4, pour les fonctionnalités de lecture et d'écriture du port série (appels Read() et Write ()).

Tests réels

Après la mise en place du drone réel et sa configuration[A8], on a testé l'interface depuis un système Linux. La première séquence testée était un simple décollage puis un atterrissage. Durant les tests, on a eu des problèmes de réception des commandes par la Pixhawk, ce qui nous a amenés à l'addition des mécanismes d'acquittement en vue de garantir l'intégrité des transactions.

Introduction des mécanismes d'acquittement

Les défauts de réception des commandes de contrôle par la Pixhawk nécessitent l'addition d'un système d'acquittement. À partir de la documentation des messages MAVLink[19], on ajoute un mode de vérification basé sur la transition des états du système PX4, fournie par le message MAVLink HEARTBEAT qui indique l'état général de la Pixhawk.

Sachant que chaque phase du système (mise en marche, décollage, acceptation des commandes de contrôle externes, atterrissage..) possède une séquence bien définie, on compare la séquence reçue, qui indique l'état actuel du système, à une autre déjà définie dans l'interface C/UART.

On utilise plusieurs scénarios de vol pour tester le fonctionnement correct de l'interface C/UART modifiée, avec l'acquittement des messages de contrôle (décollage et atterrissage, décollage, mouvements selon les axes x, y, z, et atterrissage) depuis un système Linux, puis à partir d'une STM32F4.

6.5. Contrôle du drone à travers d'un capteur externe

L'objectif du stage est d'utiliser les données fournies par l'interrogateur des capteurs, embarqué sur le drone, pour lui donner une autonomie dans la décision et une correction de position pour retrouver les capteurs. Étant donné que ce type d'interrogateur est en phase de développement, on a recours à un autre type de détecteur pour démontrer la capacité de l'interface à prendre en compte des données générées par celui-ci, et changer la position du drone.

Addition de l'accéléromètre LIS3DSH

La carte d'évaluation du STM32F4 possède le composant LIS3DSH qui est un accéléromètre à 3 axes, susceptible de communiquer avec le noyau Cortex-m4 de la carte

au travers d'une liaison **SPI**. À partir d'une modification du code fournie par un exemple de libopencm3[21], on intègre cette modification dans la partie STM32F4 de l'intérface C/UART, et qui est initialisée au début du programme, et on introduit un scénario qui prend en compte les valeurs d'accélérations fournies par l'accéléromètre pour modifier la position du drone. Cette procédure est visualisée par la figure 6.

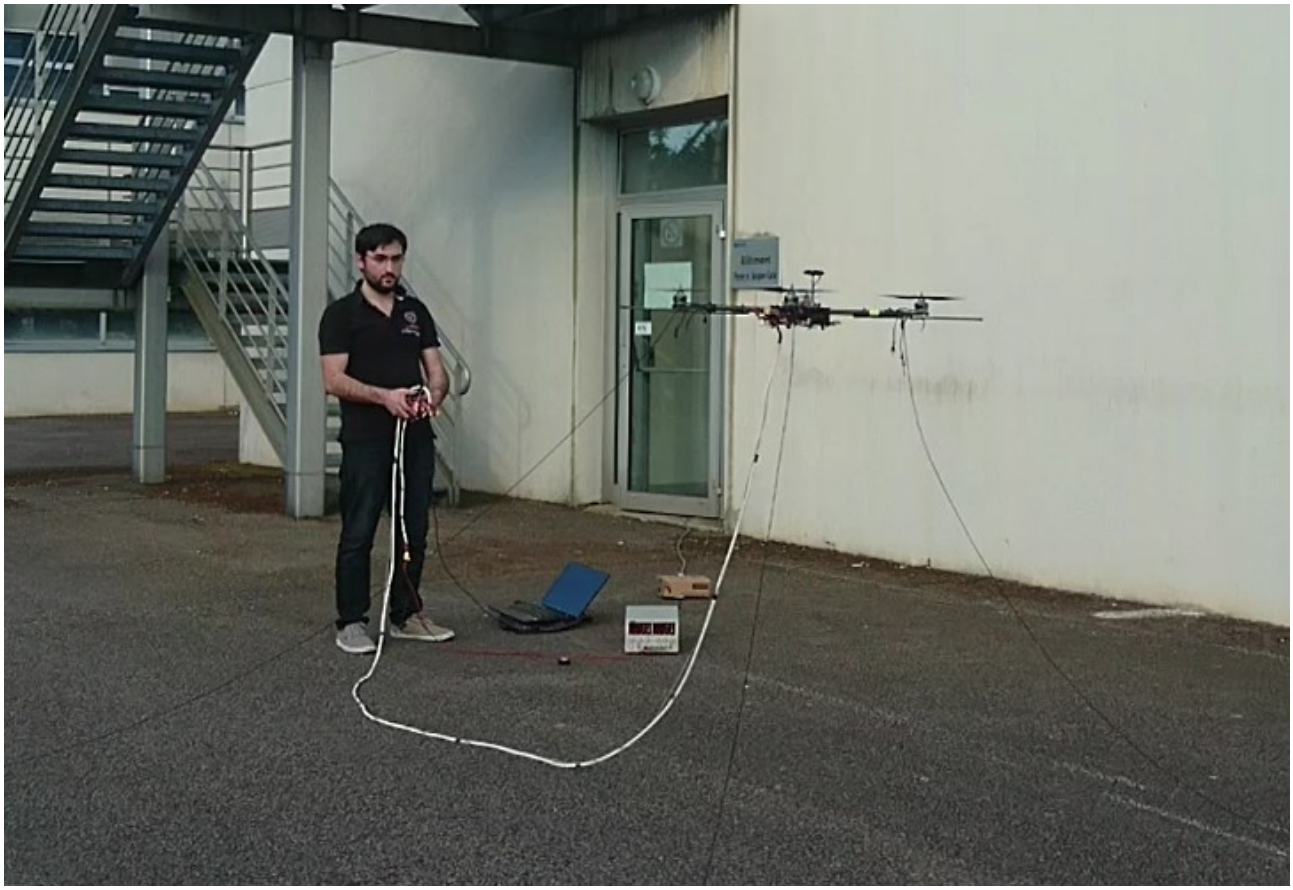


Figure 6: Contrôle du drone à partir de l'accéléromètre LIS3DSH de la STM32F4

Une explication de la mise en place du drone avec les équipements pour les tests est élaborée[A8], ainsi que la démonstration[22] de cette capacité.

7. Évolutions possibles

7.1. Contrôle depuis NuttX et création de Mcontrol

Dans la section 5.1, on a indiqué que le contrôle de la Pixhawk était possible depuis une application tournant sur cette plateforme (au lieu de l'utilisation d'un calculateur externe).

Cette approche réduit la connectique et élimine la nécessité d'avoir un calculateur externe, mais peut présenter des difficultés et des conflits potentiels avec la plateforme PX4 contrôlant la Pixhawk et le vol du drone.

Cette application nommée MControl[23] communique avec les autres applications à travers le uORB, indiqué par la figure 7.

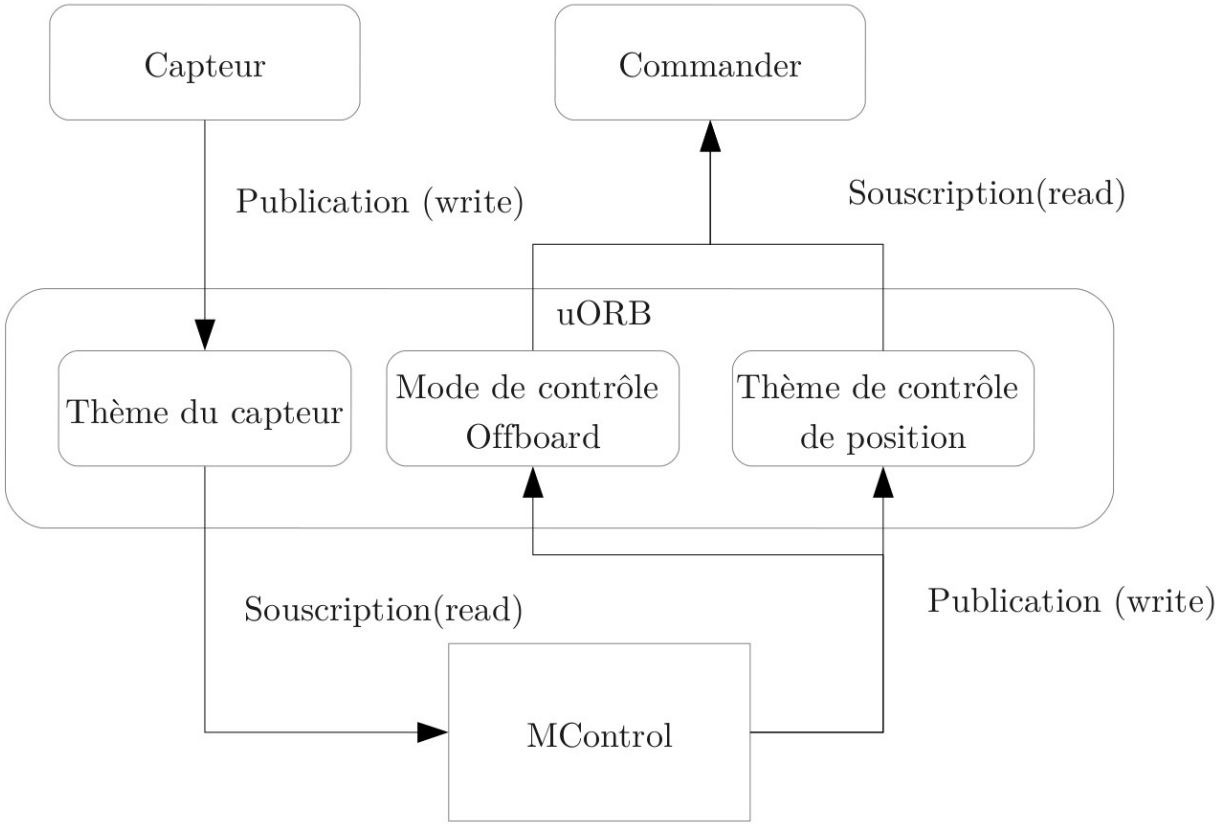


Figure 7: Schéma montrant l'architecture de MControl

Un capteur embarqué, possédant un pilote PX4/NuttX ou une application PX4, peut publier les données vers un thème spécifique, et c'est la mission de l'application MControl de récupérer ces données, et les utiliser dans sa propre machine à états pour envoyer les commandes vers d'autres applications PX4 contrôlant le vol du drone.

7.2. Intégration avec d'autres capteurs

L'interface C/UART modifiée peut servir de base pour plusieurs applications. On peut utiliser un sonar et traiter ses données, puis établir une loi de commande pour guider le drone.

Sachant que l'interrogateur RADAR des capteurs est en cours de développement, il est possible de simuler son comportement, en fabriquant des coordonnées à partir des données fictives (Phase et amplitude des échos renvoyés par la cible coopérative sondée par le RADAR embarqué sur le drone).

On est en cours de développement d'une méthode d'asservissement qui dirige le drone à partir de ces données, pour simuler le comportement vis-à-vis de la mise en place de l'interrogateur. Cette méthode est testée avec la simulation.

8. Conclusion

Les objectifs initiaux du stage étaient d'atteindre la capacité des drones d'exécuter un scénario complètement autonome et d'asservir le mouvement du drone à l'aide d'un signal recueilli au cours de l'interrogation des capteurs.

Le premier objectif était bien atteint : après la compréhension du fonctionnement de NuttX, PX4, la pixhawk et l'interface C/UART, on a pu commander le drone à exécuter un scénario autonome depuis un calculateur embarqué qui est la STM32F4.

Pourtant que l'interrogateur est toujours en phase de développement, on a utilisé un capteur externe (accéléromètre) qui peut simuler le comportement de l'interrogateur.

Nous avons observé expérimentalement, au cours de plusieurs vols, que la consigne de position issue de l'accéléromètre est convenablement prise en compte, et que les commandes sont convenablement interprétées par la centrale inertielle. Ce résultat démontre incontestablement notre capacité à commander le drone depuis un capteur ~~-~~ici un accéléromètre au lieu du lecteur de capteurs passifs interrogeables par liaison radiofréquence. Ainsi, nous sommes capables d'insérer dans la séquence des ordres transmis à la centrale inertielle par commandes Mavlink une mesure dont le résultat commande la position du drone. De ce fait, l'objectif du stage est réalisé, le cahier des charges est complété en respectant les contraintes de latences et de robustesse entre un calculateur externe et la centrale inertielle.

Une méthode rigoureuse de travail, représentée par la nécessité de tester tous les conditions sur simulateur avant d'aller expérimenter sur le matériel réel , a permis de faire voler avec succès un drone dans les conditions représentatives de l'application finale.

Ce stage fut stimulant et enrichissant. Il m'a introduit au monde des drones et des véhicules sans pilotes, il m'a permis de développer et consolider mes compétences en électronique et surtout la programmation des microcontrôleurs et les environnements embarqués.

Il m'a également introduit aux notions des ordonnanceurs, des systèmes temps réels embarquées, au déverminage des programmes, à la programmation dans les systèmes d'exploitation multitâches, ainsi que l'utilisation des outils opensource, surtout le logiciel de gestion de version Git et le débogueur GDB.

Il a aussi été très intéressant, puisqu'il m'a introduit au monde de la recherche, et m'a appris aussi l'autonomie du travail et de la prise d'initiatives.

9. Références

[29] Lorenz Meier, Dominik Honegger and Marc Pollefeys **"PX4: A Node-Based Multithreaded Open Source Robotics Framework for Deeply Embedded Platforms"**, ICRA (Int. Conf. on Robotics and Automation) 2015**.**

Sites Internet :

[1] Computer Vision and Geometry Lab of ETH Zurich
http://cvg.ethz.ch/

[2] Autonomous Systems Lab of ETH Zurich
http://www.asl.ethz.ch/

[3] 3DRobotics
https://3dr.com/

[4] Pixhawk Autopilot
https://pixhawk.org/modules/pixhawk

[5] NuttX
http://nuttx.org/

[6] Documentation du protocole MAVLink
https://pixhawk.ethz.ch/mavlink/

[7] Documentation MAVLink site de QGroundContol
http://qgroundcontrol.org/mavlink/start

[8] Guide des developpeurs de PX4
http://dev.px4.io/

[9] Documentation de QGroundControl
http://dev.px4.io/qgroundcontrol-intro.html

[10] Documentation du mode Offboard
http://dev.px4.io/offboard-control.html

[11] Code source de l'interface C/UART
https://github.com/mavlink/c_uart_interface_example

[12] Guide d'installation de Summon Arm Toolchain
http://jmfriedt.free.fr/summon_arm.pdf

[13] Bibliothèque libopencm3
http://libopencm3.org/wiki/Main_Page

[14] Répertoire du code source de St-link
https://github.com/texane/stlink

[15] Sources des examples de libopencm3
https://github.com/libopencm3/libopencm3-examples

[16] TP de programmation en STM32
http://jmfriedt.free.fr/tp3_m1.pdf

[17] Ordonnanceur pour STM32vl-discovery
https://github.com/libopencm3/libopencm3-examples/pull/98/files

[18] Système d'exploitation minimal temps-réel Atomthreads
http://atomthreads.com/

[19] Documentation des paquets MAVLink
https://pixhawk.ethz.ch/mavlink/

[20] Schéma de câblage de la pixhawk
http://px4.io/docs/hw-wiring-diagrams/

[21] LIS3DSH exemple dans libopencm3
https://github.com/libopencm3/libopencm3-examples/pull/101/files

[22] Démonstration du contrôle avec LIS3DSH et la STM32F4.
http://sendvid.com/ziebmszp

[23] Source code de MControl (Application sous NuttX)
https://github.com/MHageH/mcontrol

[24] Source code de l'interface C/UART modifie
https://github.com/MHageH/c_uart_interface

[25] Référence Technique de la modification
https://mhageh.gitbooks.io/c-uart-interface-technical-details/content/

[26] Guide de mise en place de l'environnement de développement
https://mhageh.wordpress.com/2016/05/28/stm32f4-dev/

[27] Présentation du laboratoire FEMTO-ST
http://www.femto-st.fr/fr/L-institut/Presentation/

[28] Présentation de laboratoire TF
http://www.femto-st.fr/fr/Departements-de-recherche/TEMPS-FREQUENCE/Presentation/

10. Glossaire

POSIX : Portable Operating System Interface est une famille de normes techniques définie par l'Institut des ingénieurs électriciens et électroniciens (IEEE). Elle porte sur la standardisation des interfaces de programmation de logiciels tournant sous système de type UNIX.

STM32 est une famille de microcontrôleurs de 32-bits fabriquée par STMicroelectronics. Cette famille est basée sur les noyeaux 32-bits Cortex-M.

Cortex-M est un groupe de noyaux ARM du type RISC (Reduced Instruction Set Computer) destinée à l'implémentation dans des microcontrôleurs.

UART : universal asynchronous receiver/transmitter est un composant utilisé pour établir la communication entre l'ordinateur et un port série d'une façon asynchrone.

RS-232 : ou **EIA RS-232** est une norme qui standarise une voie de communication série sur 3 fils minimum.

Préemption représente la capacité d'un système multi-tâche à exécuter, arréter, ou continuer l'exécution d'une tâche planifiée en cours.

RTOS : Real Time Operating System (ou système d'exploitation temps-réel).

Round-robin est un algorithme d'ordonnancement utilisé dans les systèmes travaillant en temps partagé.

USART : Universal Synchronous/Asynchronous Receiver/Transmitter est un type d'interface sérialе qui est programmable pour la communication synchrone ou asynchrone.

Netcat : Utilitaire permettant d'ouvrir des liaisons réseaux, de type UDP ou TCP. Il existe sur plusieurs types de système d'exploitation.

Timer : Un périphérique matériel pour la génération des durées, est souvent implémenté dans des microcontrôleurs pour la synchronization des fonctions qu'ils sont chargés d'accomplir.

GPIO : General-Purpose Input/Output est un circuit intégré generic contrôlable par l'utilisateur ou le programmeur pour communiquer avec d'autre composant électronique.

Appels système : ou **SysCall** est un ensemble d'instructions primitives fournies par le noyau et utilisées dans les programmes s'exécutant dans l'espace utilisateur.

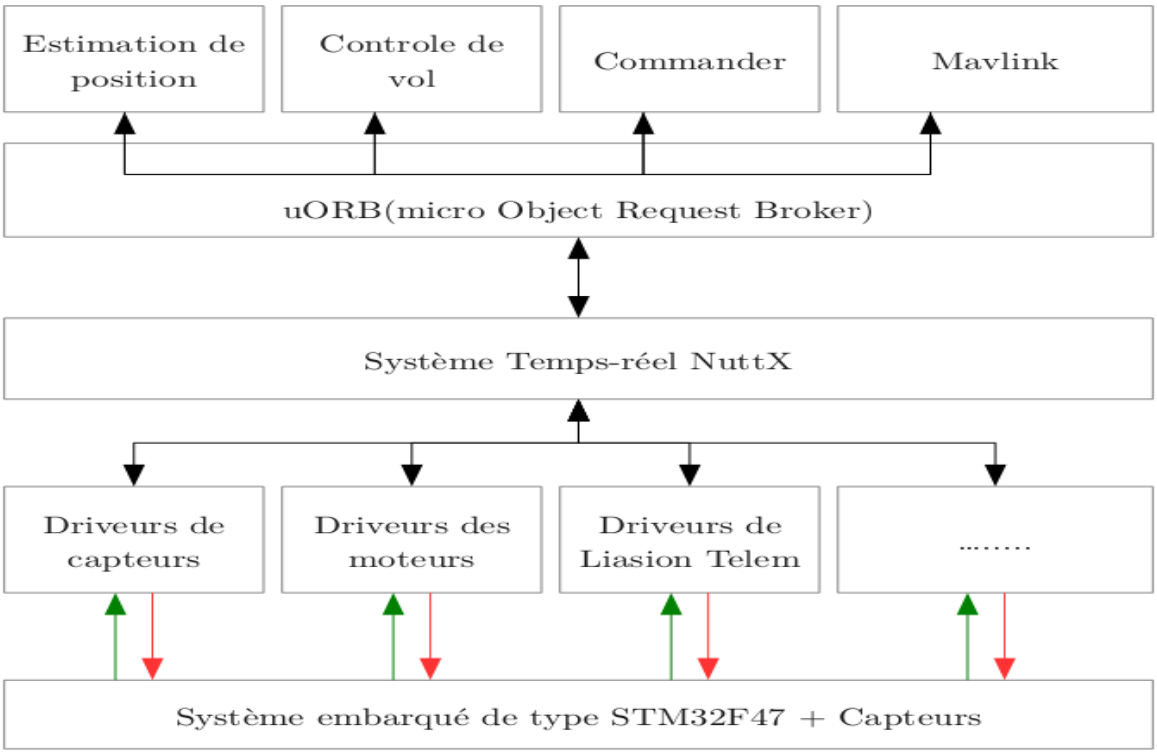
Gdb : GNU Debugger est le débogeur standard du projet GNU, et portable sur de nombreux systèmes de type *nix.

SPI : Serial Peripheral Interface est un bus de données synchrone capable de les envoyer dans un sens bidirectionnel entre 2 (ou plusieurs) périphériques selon un schéma maître-esclave(s).

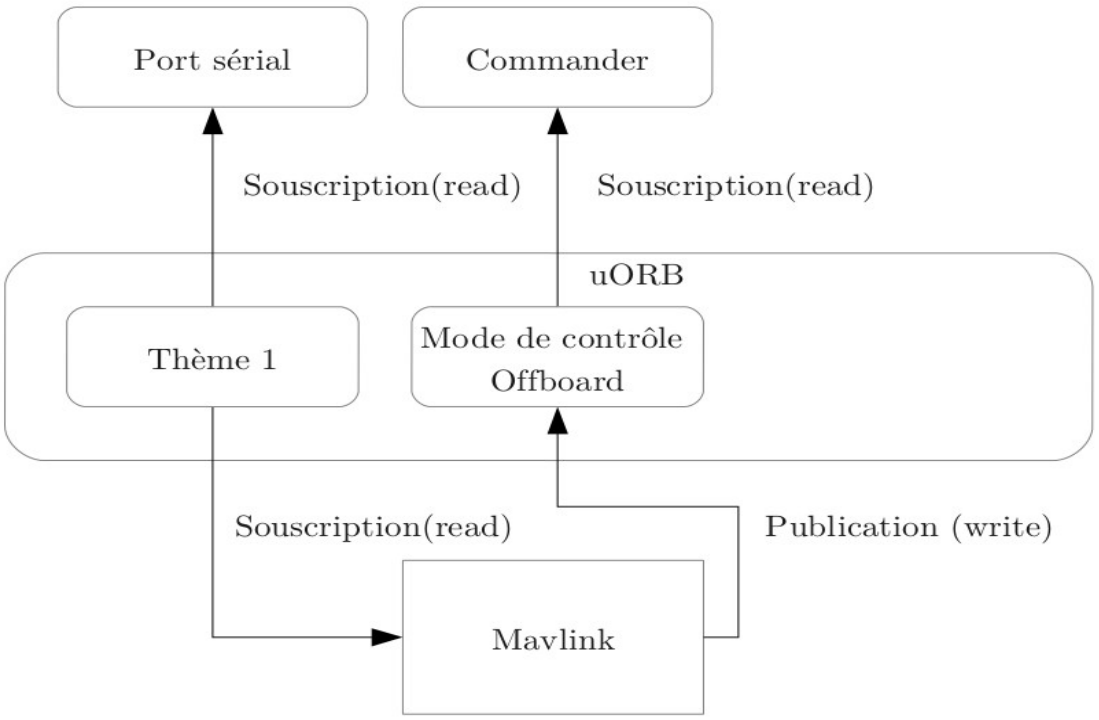
11. Annexes

Le code source[24] de l'interface C/UART modifiée, ainsi qu'une explication entière du code[25] et un guide d'utilisation[24] sont fournies.

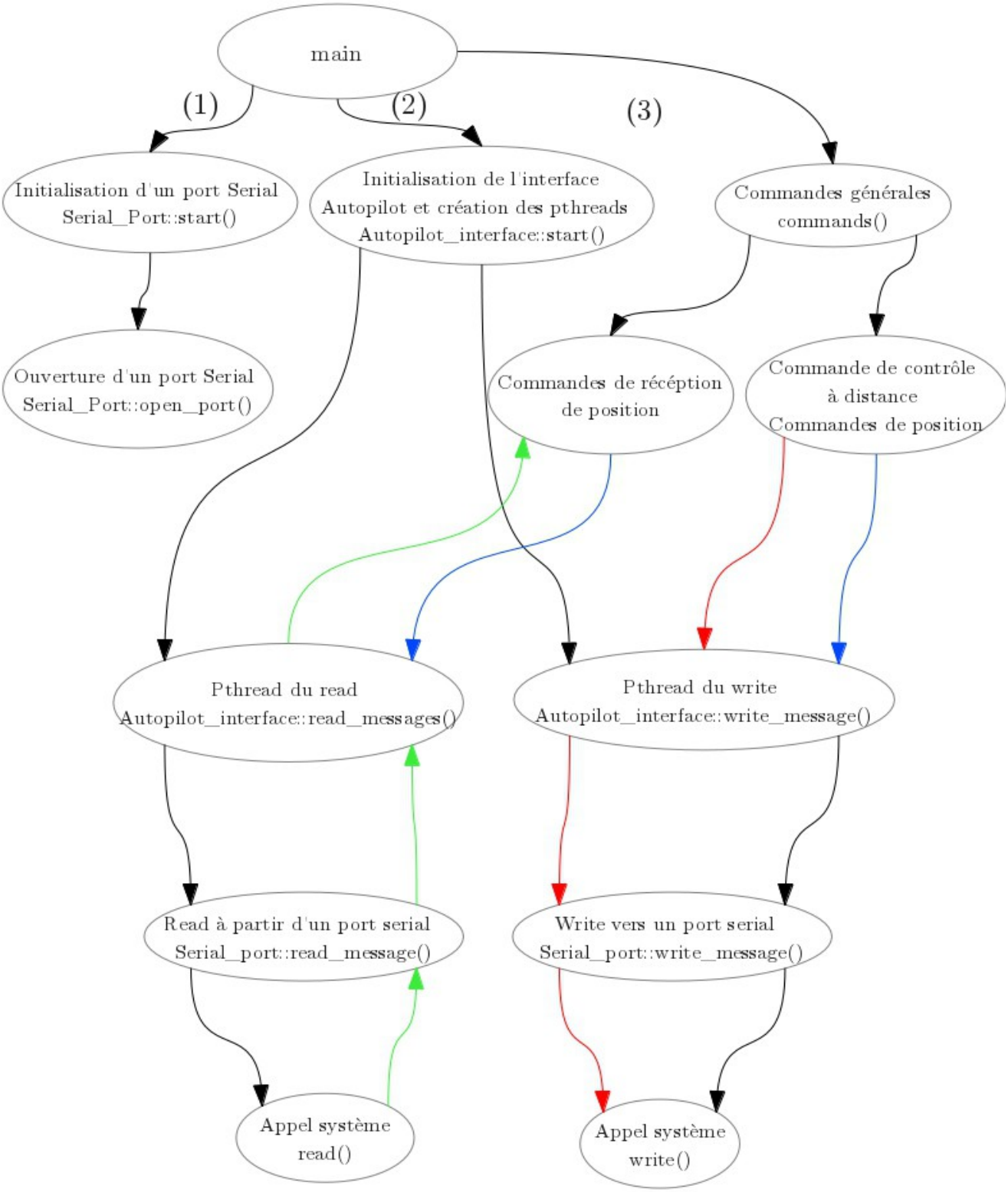
[A1] Schéma des couches de la platforme PX4



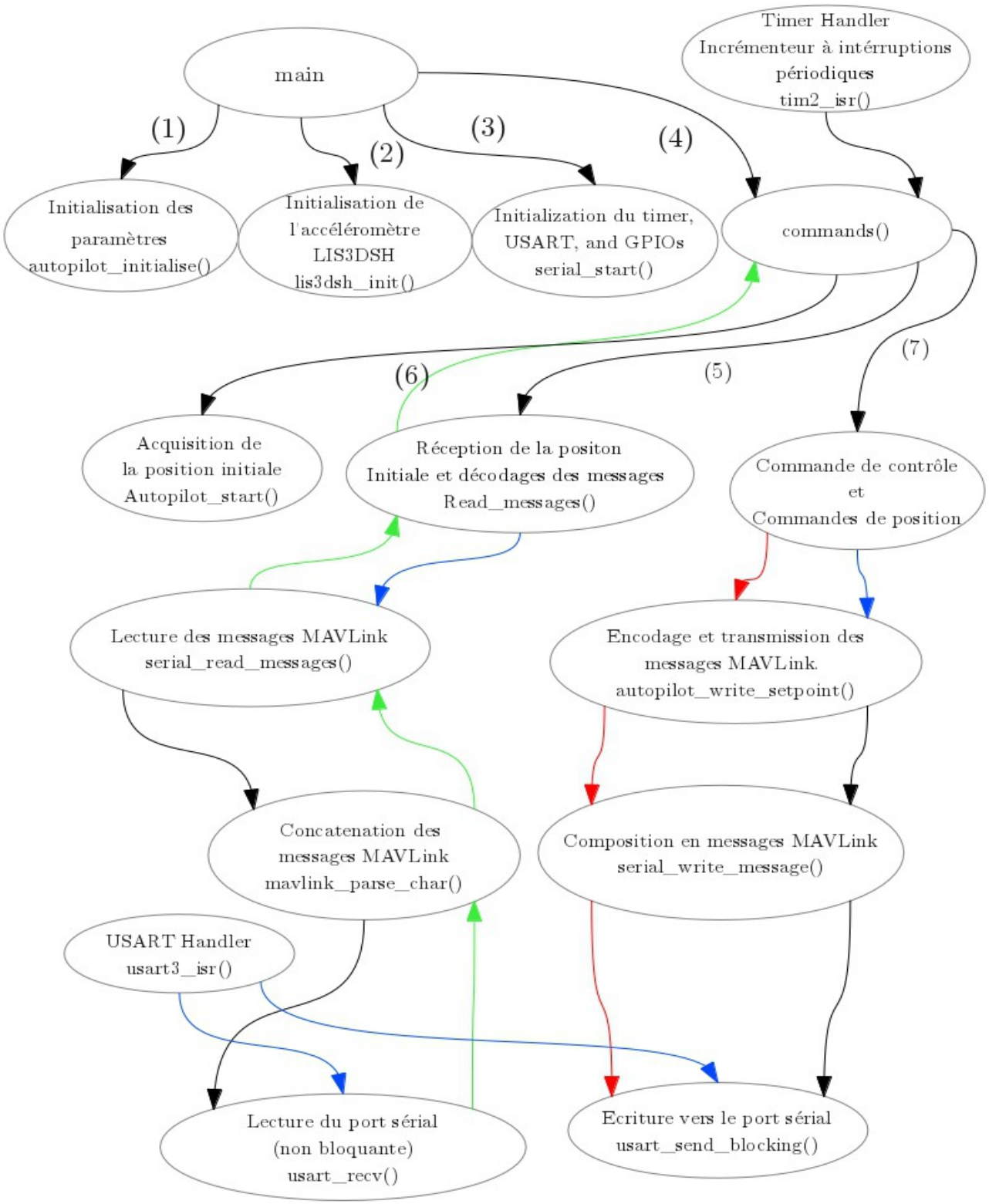
[A2] Schéma de la communication interprocessus au travers de uORB



[A3] Schéma détaillé du programme avant modification avec les ordres d'exécution.

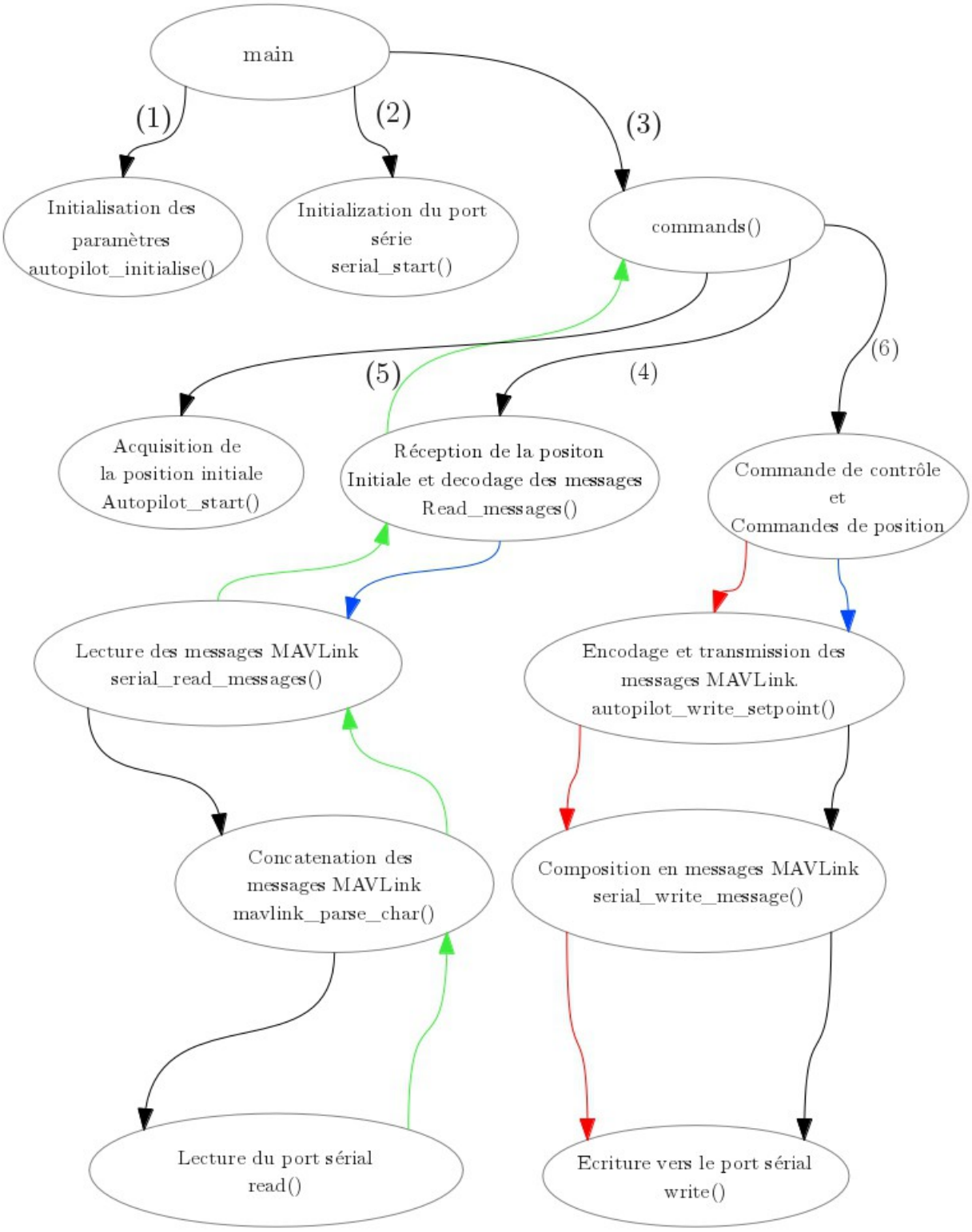


[A4] Schéma détaillé du programme après modification (sous STM32F4) avec les ordres d'excécution.



Aprés l'acquisition de la position initiale (6), la séquence des commandes boucle sur (5) et (7).

[A5] Schéma détaillé du programme après modification (sous PC)



Aprés l'acquisition de la position initiale (5), la séquence des commandes boucle sur (4) et (6).

[A6] Liaison Netcat/Socat

L'application de l'interface C/UART sur le drone en simulation présente une problématique :

La simulation ouvre 2 ports UDP locaux pour la communication avec le monde extérieur (port 14556 en réception, port 14540 en émission).

Sachant que l'interface C/UART communique avec la platforme pixhawk à travers une liaison série, il en est nécessaire d'établir un moyen de communication UDP-sérial bidirectionnel.

On utilise netcat pour la procédure :
Pour diriger les informations fournies par le drone vers le monde extérieur, on commande netcat à écouter sur le port 1450 pour la machine locale, puis renvoyer tous les informations reçues vers le port série ttyUSB1

```
nc -l -u -p 14540 127.0.0.1 "|" cat ">" /dev/ttyUSB1
```

Pour rediriger les commandes vers le drone virtuel depuis l'interface, on commande netcat à prendre tous les informations qui sortent du port série ttyUSB1 vers le port de réception 14556 local

```
cat < /dev/ttyUSB1 | nc -u 127.0.0.1 14556
```

Sur le système Linux, il se trouve qu'on ne peut pas utiliser un port série simultanément avec 2 processus différents. Pour exécuter l'interface C/UART, on établit une liaison physique entre le port ttyUSB1 et ttyUSB0.

On peut toujours utiliser l'application Socat pour créer 2 ports virtuels, qu'ils ont une fonctionnalité semblable à une liaison physique :

```
socat -d -d pty,raw,echo=0 pty,raw,echo=0
```

[A7] Liaison Netcat vers STM32F4

La liaison Netcat est aussi nécessaire pour une communication avec la STM32F4.

On doit établir une liaison au microcontrôleur au travers d'un adaptateur USB/série, vers les ports GPIO spécifique à l'USART3 (Broches D8 (TX), D9 (RX) et GND).

Ainsi, on peut initialiser la connexion comme prévu :

Pour envoyer les messages MAVLink sortant du drone virtuel vers la STM32F4 :

```
nc -l -u -p 14540 127.0.0.1 "|" cat ">" /dev/ttyUSB0
```

Pour récupérer les commandes envoyées par la STM32F4 et les rediriger vers la simulation :

```
cat < /dev/ttyUSB0 | nc -u 127.0.0.1 14556
```

[A8] Mise en place du drone

À partir de la plateforme multicopter fourni par le laboratoire, et le schéma de câblage de la pixhawk[20], on construit le drone pour tester l'interface C/UART en réel. La configuration et la calibration des moteurs sont faites avec QGroundControl[9]. Pour les premiers tests, on a utilisé une alimentation DC pour remplacer les batteries, mais on a repris leur utilisation après.

La liaison de chaque moteur avec la pixhawk doit suivre une méthode définie dans la configuration par QGroundContol (figure 8)

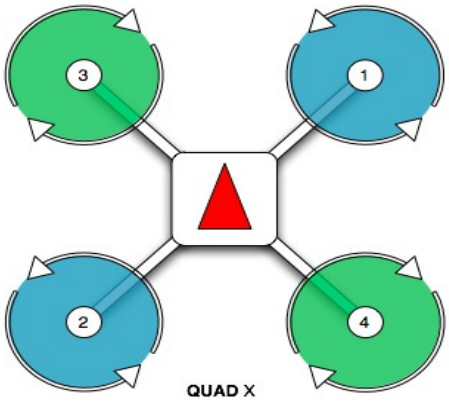


Figure 8: Schéma des connexions des moteurs
vers les canaux de la pixhawk

Pour réduire les dégâts des chutes causées par les dysfonctionnements possibles de l'interface C/UART, des barres en fibre de carbone sont ajoutées, avec des ficelles comme mesures de sécurité.