ENAC

Rapport

Réalisation d'une interface de calibration pour IMU de mini-drones dans le cadre du projet PAPARAZZI

Alinoé ABRASSART, Florent GERVAIS, Christophe NAULAIS, Guillaume SAAS

09/06/2013

Table des matières

Aucune entrée de table d'illustration n'a été trouvée.

I. Système

L'unité drone de l'ENAC développe un système de pilote automatique pour micro-drone baptisé PAPARAZZI. Créé en 2003, ce système gratuit et libre permet de faire naviguer de façon autonome, à l'aide d'un GPS, un ou plusieurs micro-drones à voilures fixes ou rotatives contrôlés par une station au sol. Il comprend l'ensemble des logiciels permettant aux drones de naviguer ou d'être contrôlés et les méthodes de fabrication des composants matériels nécessaires à l'assemblage d'un drone complet. La clé de ce système repose sur la combinaison d'une centrale inertielle, du GPS et d'autres éléments qui nous permettent d'obtenir l'orientation et la position du micro-drone. Aujourd'hui, PAPARAZZI est régulièrement utilisé par de nombreux utilisateurs expérimentés. Il tend à s'élargir de plus en plus vers le grand public et nécessite donc de nouveaux outils pour permettre aux novices de réaliser facilement les tâches nécessaires à la création d'un drone. C'est dans ce contexte que notre projet s'installe.

Une étape importante pour la préparation d'un nouvel engin est alors la calibration de sa centrale inertielle. Notre projet intervient dans la réalisation de cette étape. La méthode actuelle consiste à enregistrer les valeurs que nous renvoient les capteurs puis d'appeler un script de calibration sur ces données. Cette méthode repose sur l'utilisation du bus Ivy qui est un bus logiciel permettant simplement d'envoyer des messages textes entre applications. Pour la calibration du micro-drone, la récolte des données se fait de la manière suivante:

- le drone (réel ou simulé) envoie des données sur le bus IVY
- l'application Link collecte ces données en provenance du drone mais aussi d'autres applications PAPARAZZI telles que la ground control station (GCS) ou le serveur.
- l'application serveur, une fois configurée, récupère les données et les stocke dans un fichier de log contenant l'ensemble des messages émis par le drone.

Ce fichier de log est ensuite utilisé par un script Python qui va lire ce fichier et en extraire les données pour générer le résultat de la calibration.

Cette méthode manque clairement d'intuitivité ce qui la rend complexe pour des utilisateurs novices. De plus même les utilisateurs expérimentés ne sont pas satisfait de cette méthode à cause du manque de retour sur la qualité des données acquises par les capteurs pendant la capture. Notre projet consiste à guider l'utilisateur pendant la méthode de calibration de manière claire et précise pour faciliter la tâche à l'utilisateur et aussi donner un retour des données acquises par les capteurs pour améliorer la précision de la calibration.

On s'intéresse donc ici au développement de ce logiciel de calibration, des contraintes qu'il devait remplir et du niveau d'achèvement atteint par le projet à ce jour. On insistera particulièrement sur la réalisation, les modifications apportées entre la version finale et la conception qu'il en a été faite.

1. Objectifs

L'objectif du projet est de répondre aux besoins des utilisateurs. Pour cela on s'appuiera sur le dossier des besoins. Celui-ci nous fournit un tableau récapitulatif des besoins dans lequel on peut identifier plusieurs axes de réalisation.

a. Scénarios globaux

Identifiant	Besoin	Justification	Quantification	Niveau de priorité
B1.1	Le système permet la réalisation du use case : « calibration magnétométrique » (réalisation de la totalité des use cases de phases dans la configuration magnétomètre)	Scénarios de travail : 1, 3	Ecart des résultats avec la méthode précédente < 5% et temps de réalisation inférieur	100
B1.2	Le système permet la réalisation du use case : « calibration des accéléromètres » (réalisation de la totalité des use cases de phases dans la configuration accéléromètre)	Scénarios de travail : 2, 4	Ecart des résultats avec la méthode précédente < 5% et temps de réalisation inférieur	100
B1.3	Le système permet la réalisation du use case : « calibration des gyromètres »	Bonus	Ecart des résultats avec la méthode précédente < 5% et temps de réalisation inférieur	0

On cherchera donc à construire un système permettant de réaliser les uses cases des accéléromètres et des magnétomètres. Le temps ne nous a permis de ne réaliser que ces deux cas, celui des gyroscopes a donc été abandonné. Il n'en sera plus fait mention dans la suite du rapport mais il sera à nouveau abordé dans la partie vérification et validation (V&V). A ces besoins globaux, on peut ajouter plusieurs besoins nécessaires à leur réalisation qui auront leur importance dans la V&V. On notera qu'ils sont des besoins fondamentaux du système sans lesquels il devient inutilisable.

Identifiant	Besoin	Justification	Quantification	Niveau de priorité
B1.4	Le système permet de choisir entre différents types de calibration	Association des scénarii : 1, 2, 3, 4	Pas de quantification	100
B1.5	Le système permet de générer des résultats de calibration	Tous les scénarii : 1, 2, 3, 4	Pas de quantification	100
B1.6	Le système permet d'enregistrer les données collectées pendant la calibration	Tous les scénarii : 1, 2, 3, 4	Pas de quantification	50

		Point fort des scénarii : 1, 2, 3, 4		

Le besoin 1.6 était présent pour des questions de rejeu. Ce rejeu ne peut se faire directement dans notre application, la fonctionnalité est cependant préservée. En effet, l'utilisateur expert pourra, en cas de besoin, récupérer le log de la calibration comme nous le verrons plus tard.

b. Performances attendues

Identifiant	Besoin	Justification	Quantification	Niveau de priorité
B2.1	Le système a une interface graphique permettant de guider l'utilisateur	Point faible des scénarii : 3, 4	L'utilisateur limite ses phases de recherches de fonctions à 10 secondes en moyenne	100
B2.2	Le système a une interface graphique permettant de juger de la qualité des résultats	Point faible des scénarii : 3, 4 Point fort des scénarii : 1, 2	L'utilisateur est capable de se prononcer immédiatement sur la qualité de la calibration en cours et ce à tout moment	100
B2.3	Le système permet l'arrêt et la reprise d'une calibration	Besoin exprimé lors d'interview contextuel	Pas de quantification	75
B2.4	Le système permet de générer des résultats à partir de résultats partiels de la calibration magnétométrique et des accéléromètres	Besoin exprimé lors d'interview contextuel	Pas de quantification	25
B2.5	Le système permet de choisir entre une calibration des accéléromètres ou des magnétomètres	Association des scénarii : 1, 3 avec les 2, 4	Pas de quantification	100
B2.6	Le système est indépendant des autres éléments de PAPARAZZI et du système d'exploitation	Point fort des scénarii : 1, 2, 3, 4	Pas de quantification	80
B2.7	Le système doit permettre une collecte facile des résultats	Point faible des scénarii : 3, 4	Temps de collecte des résultats pour une autre utilisation inférieure à 5 secondes	90
B2.8	Le système doit indiquer l'ensemble des manipulations à effectuer	Point faible des scénarii : 3, 4	Pas de quantification	100

Tout ces points représentent des besoins en fonctionnalité donnés par l'utilisateur lors des interviews contextuelles et des rendez-vous ultérieurs. Ils vont permettre du juger de l'accomplissement de l'application. Les besoins B1 listent les performances nécessaires au système, les besoins B2 listent les performances qui feront que le système présente un réel avantage par rapport au précédent et qu'il sera utilisé. On cherchera donc autant que possible à décrire l'avancement par rapport à ces points et notamment en partie III.

c. L'interface

Identifiant	Besoin	Justification	Quantification	Niveau de priorité
B3.1	Le système doit proposer une interface conviviale	Point faible des scénarii : 1, 2, 3, 4	L'utilisateur ne doit pas avoir à réfléchir lors d'une première utilisation (temps d'apprentissage < 5secondes)	75
B3.2	Le système donne un feedback temps réel (pas de temps de latence visible sous charge normale du système global) à l'utilisateur	Point faible des scénarii : 1, 2, 3, 4	Feedback < 5secondes par rapport à la dernière mesure	90

On peut s'étonner du nombre limité de contraintes sur l'interface pour une application essentiellement graphique. Dans les faits, c'est ce caractère essentiellement graphique qui pousse à reporter une partie des besoins d'interface dans les performances attendues. On ne liste donc ici que les performances communément attendues d'une interface dite « user-friendly » dans le sens large du terme : une bonne réactivité et une grande facilité d'accès. On liste en fait ici les besoins liés à l'arrivée des utilisateurs novices sur le système.

d. Les contraintes

Identifiant	Besoin	Justification	Quantification	Niveau de priorité
B4.1	Le système fonctionne de manière nominale sous un environnement et une charge de travail normale et une configuration de travail classique	Besoin exprimé lors des interviews contextuelles	Pas de latence supérieure à une seconde sur l'ensemble de l'interface	100
B4.2	Le système garantit que la qualité des résultats sera au moins la même qu'avec le système précédent	Point fort scénarii : 1, 2, 3, 4	Différence inférieure à 5%	95
B4.3	Le système doit garantir un temps de calibration équivalent à celui de la méthode précédente (temps total plus ou moins 5 secondes) ou inférieur	Point fort scénarii : 1, 2	Temps de calibration inférieure de 10 secondes souhaité	50
B4.4	Le système doit informer en temps réel l'utilisateur de son état	Point faible scénarii : 1, 2, 3, 4	Pas d'immobilisation de l'interface sans justification	80

Les contraintes sont le passage obligé de tout système. Ici elles ne sont pas très nombreuses. On relève des contraintes d'ordre de performance essentiellement et de manière comparée avec l'ancien système. Le besoin 4.4 a été litigieux par son aspect d'interface. Cependant l'objectif du projet était clairement la réalisation de ce besoin, le rappeler dans les contraintes nous permettaient de mettre en évidence sa nécessité et de ne pas l'oublier ni dans la réalisation ni dans la vérification et la validation.

2. Moyens

On va ici détailler les moyens à notre disposition pour la réalisation d'un système permettant de répondre à ces besoins.

a. Moyens humains

L'équipe est composée de 4 personnes : Guillaume Saas, Christophe Naulais, Florent Gervais, Alinoé Abrassart. 40 heures par semaines peuvent être dédiées par chaque personne sur la période du projet via des créneaux dédiés ou pris sur du temps personnel. Des impondérables peuvent cependant remettre ce quota en cause comme on pourra le constater en partie II.1. A cette équipe de développement viennent s'ajouter le client défini comme utilisateur expert qui a été force de suggestions et de conseils tout au long du projet mais aussi le soutien professoral par ses relectures attentives durant la phase de spécification et son aide ultérieure. Les ressources humaines sont donc considérables et relativement disponibles. Il convient donc de les utiliser à bon escient.

Tout au long du projet, ces ressources ont été mises à contribution. On détaillera la manière dans la partie II.2.

b. Moyens matériels

Chaque membre de l'équipe disposait à partir des premiers jours du projet d'un ordinateur exécutant une version de Linux (debian) avec Eclipse et Paparazzi opérationnels. Ces moyens matériels étaient nécessaires à l'exécution convenable du programme. Cela a notamment permis les tests avec la fonctionnalité rejeu de PAPARAZZI et le développement avec des fonctionnalités spécifiques aux systèmes Unix, ce malgré la portabilité de java. D'autre part, dans les derniers temps du projet, des tests ont été effectués sur une architecture PAPARAZZI réelle avec un drone réel envoyant les messages sur le bus IVY.

On distinguera donc 2 phases de conception au niveau des moyens matériels que ce soit dans ce rapport ou dans le plan de V&V attenant. Une première phase s'appuyant sur des rejeux et une deuxième sur l'architecture réelle.

La première phase utilise deux entités. La première est la fonctionnalité « replay » de l'interface PAPARAZZI permettant d'exécuter un fichier de log correspondant à une session déjà exécutée. Dans notre cas, Mr Hattenberger a effectué des calibrations avec la procédure actuelle qui génère un fichier .log et un fichier .data. Ce fichier .log est ensuite mis en entrée de la fonctionnalité « replay » qui rejoue alors l'ensemble des messages envoyés durant la calibration réalisée par Mr Hattenberger. La deuxième entité est la classe « Sender » réalisée afin de pouvoir rejouer des fichiers .data. Elle se contente de lire les fichiers .data et d'envoyer les messages qui y sont stockés sur le bus. Son intérêt était de pouvoir tester le filtre ainsi que les algorithmes de calibration de manière rapide et directement sur les données disponibles aux algorithmes de calibration initiaux.

c. Connaissances

Au début du projet, les connaissances en Java de chacun des membres ont permis la première phase de développement et de prototypage comme on le verra en partie II.3. Ces connaissances n'étaient guère suffisantes pour mener à bien le projet plusieurs difficultés abordées rapidement en cours faisant leurs apparitions notamment la gestion de thread et de l'interface graphique. D'autre part, les connaissances concernant PAPARAZZI ont dû être acquises de manière personnelle via des formations proposées par l'encadrant ou par autoformation. Une grande part du projet a donc consisté en cette autoformation.

II. Réalisation

Cette partie s'intéresse concrètement aux différentes étapes de conception de notre application. On va alors expliquer ce que l'on a fait mais aussi pourquoi on l'a fait. Les dossiers de conception et des besoins nous ont permis d'acquérir les démarches de la réalisation de notre application. Cependant, la réalisation du prototype nous a confrontés à plusieurs difficultés qui ont entrainé certaines modifications.

1. Contexte

La réalisation de notre projet s'insère dans un contexte particulier. Dès l'obtention du sujet de notre projet nous avons commencé à planifier et répartir les tâches pour mener à bien celui-ci. Toute cette planification est retranscrite dans le plan de projet rendu il y a un peu plus d'un mois, ici nous allons détailler les échéances réelles de notre projet. Ensuite nous traiterons des risques prévus dans le plan de projet et d'autres imprévus que l'on a rencontré en expliquant pourquoi nous n'étions pas parfaitement préparés à leur arrivée.

a. Les échéances temporelles

Le début de notre projet était essentiellement centré sur la planification de notre projet avec en parallèle la définition et la rédaction des besoins de l'utilisateur dans le but de définir clairement les tâches à accomplir et les objectifs à atteindre. Cette première étape de notre projet s'est vue retardée par le manque d'heures allouées au projet et notre souhait de réaliser un dossier des besoins conséquent et de bonne qualité. Ce retard fut d'environ une semaine et on ne va jamais vraiment réussir à le combler (à part dans la dernière semaine de projet) et va même s'accentuer plus tard à cause de la réalisation du prototype dont on parlera plus en détail dans quelques lignes.

Ces dossiers se sont vus accompagnés peu de temps après par le dossier de conception qui nous permis de spécifier la façon dont sera réalisé l'architecture du logiciel, il détaille les classes principales qui peuvent être déduites de manière abstraite du dossier des besoins. Le rendu de ce dossier s'est fait en temps et en heure au détriment d'autres tâches que l'on devait réaliser au même moment. Ce choix a été fait dans le but d'avoir un dossier de conception de qualité et validé par un encadrant. Le manque d'expérience dans la rédaction d'un tel dossier a cependant causé du retard dans la partie formation aux outils qui vont nous être utile mais aussi dans le développement de notre application.

Avec une semaine de retard sur notre planning et une formation assez rapide aux différents outils, nous nous sommes intéressés au développement du prototype. Cette étape fut la plus handicapante en terme de retard car après une formation inutile sur la librairie graphique (voir Risque), la réalisation concrète de notre prototype s'est vu l'objet de nombreuses difficultés que nous évoquerons dans la partie Réalisé. Ces difficultés vont alors créer un retard considérable afin d'être corrigées pour avoir un prototype fonctionnel. Avant les deux dernières semaines précédant l'échéance finale, on a pu évaluer notre retard à environ une semaine et demie. Ce fut au prix d'efforts et de nombreuses heures de travail dans les deux dernières semaines qu'on a pu enfin combler la totalité de notre retard pour avoir un prototype et les dossiers rédigés pour l'échéance finale. Cependant vis-à-vis des heures de cours allouées au projet, notre planning réel paraît normal car toutes les heures étaient allouées à la fin du projet.

Livrables/Jalon	Date de fin prévue	Date de fin réelle
Début du projet	02/04/2013	02/04/2013
Fin du PdP	15/04/2013	23/04/2013
Fin du dossier de besoins Exigences	16/04/2013	25/04/2013
Fin du dossier de conception	13/05/2013	13/05/2013
Début de la phase de développement/ Fin de la spécification	14/05/2013	14/05/2013
Fin de l'interface graphique	27/05/2013	05/06/2013
Fin du prototypage de calibration	28/05/2013	06/06/2013
Fin du lien entre interface graphique et la calibration	05/06/2013	07/06/2013
Fin de la rédaction du rapport	07/06/2013	09/06/2013

b. Risques

L'analyse des risques dans le plan de projet nous a permis de les définir préalablement afin de réaliser des actions pour empêcher leur apparition dans notre projet. Cependant certains risques sont tout de même apparus et nous ont causés du travail supplémentaire :

Numéro du risque	Description du risque	Cause du risque	Fréquence
2	Perte du travail effectué après la dernière sauvegarde	Mauvaise utilisation de git	20 fois
3	Perte de ressources liée à l'indisponibilité d'un des membres de l'équipe	Imprévus, empêchements	5 fois
4	Perte de performance au niveau de l'équipe	Tension dans l'équipe	3 fois
5	Perte de temps sur la conception de l'interface	Mauvais choix de librairie graphique	1 fois
7	Sous-estimation des heures attribuées à la conception de l'interface	Manque d'expérience pour cet exercice et complexité des librairies graphiques	2 fois
8	Sous-estimation des heures attribuées à l'intégration du système	Manque d'expérience pour cet exercice et complexité de l'intégration	3 fois
10	Mauvaise compatibilité entre l'IHM et le logiciel de calibration	Manque d'expérience en intégration logicielle et complexité de l'application de la calibration	1 fois

Tableau 1: Réalisation des risques

La plupart des risques apparus sont explicables et dus à une mauvaise utilisation des outils de management de projet. Nous avons eu une mauvaise appréciation de ces risques, de leur possible fréquence d'apparition et la réponse prévue n'a pas toujours été appliquée. Cependant certaines réponses envisagées et mises en place n'ont pas toujours eu l'effet escompté et ont été insuffisantes pour répondre aux risques. Les risques 1, 5, 7, 8, 10 sont de cet ordre. Il est dommage que la formation sur git n'accorde pas plus de temps à l'utilisation de fonctionnalités avancées tel que le merging via merge tools ou encore sur la fonctionnalité reset car elle nous aurait permis d'éviter le risque 1. Afin de choisir une bonne librairie graphique, la recherche de celle-ci et son évaluation a été réalisé. Cependant la méconnaissance de certaines librairies graphiques et de leur capacité réelle a provoqué le risque 5 et donc une formation inutile sur une librairie graphique que l'on n'a pas réutilisée par la suite. De plus le manque de savoir-faire en intégration logicielle et la trop grande complexité par rapport à notre formation (utilisation de Threads multiples) a provoqués les risques 7, 8, 10 de manière inopinés. D'autre part, un risque n'a pas été envisagé et nous a forcé à prendre certaines mesures : des relectures plus approfondies ont été mises en place pour pallier aux lacunes en français de certains membres.

Globalement l'évaluation des risques a été bien réalisée, mais la gestion des risques a quant à elle été juste partiellement établie et s'est trouvée insuffisante pour répondre efficacement aux risques. La mauvaise utilisation des outils de management de projet sont à l'origine de ce manque.

2. Moyens

On va ici détailler les moyens que l'on a mis en place. On distinguera deux phases de mobilisation des moyens. Une première phase s'appuyant sur la conception et le prototypage et la seconde sur l'implémentation et les tests.

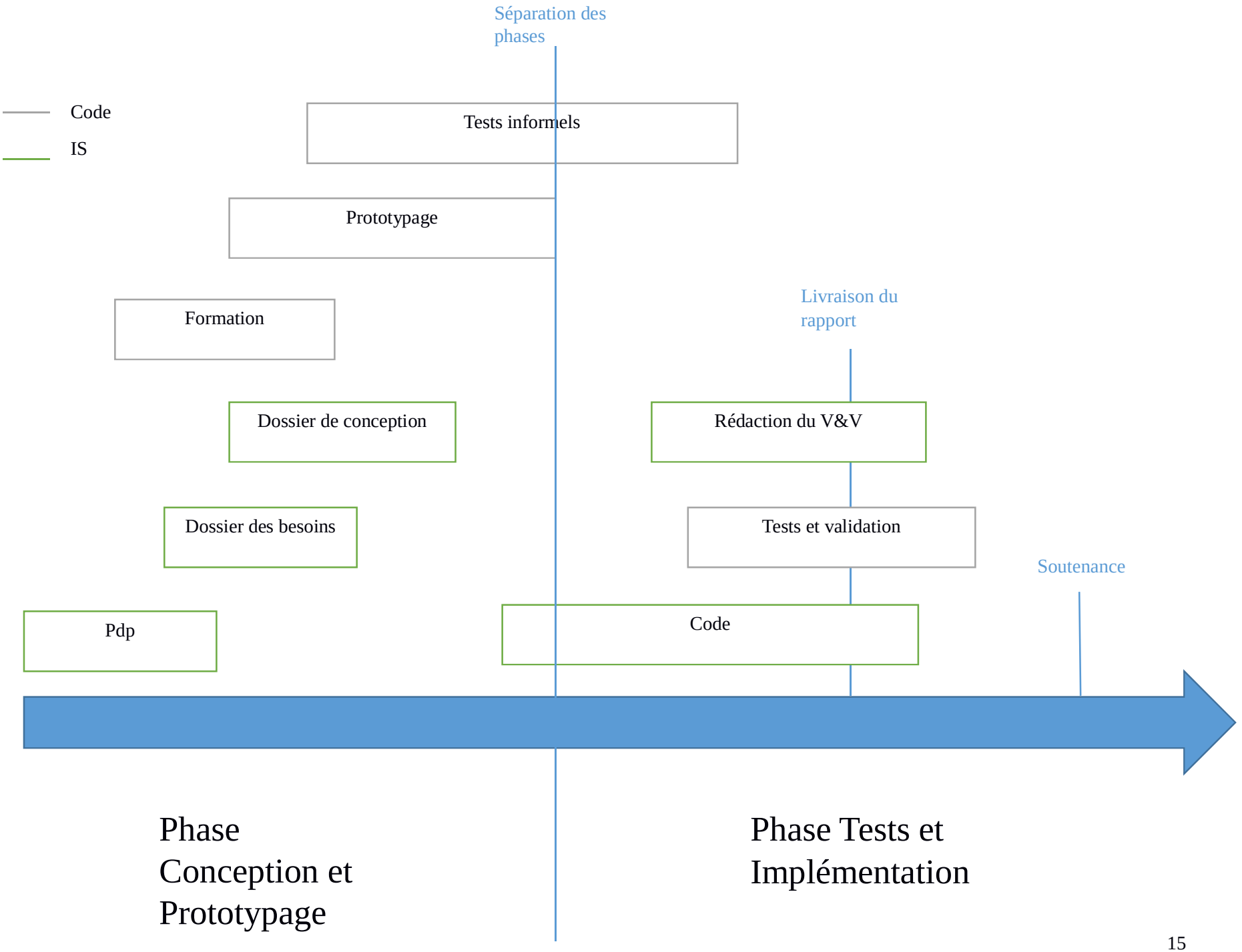


Figure 1 : Déroulement du projet

a. Pendant la phase de conception et de prototypage

Durant cette phase, l'équipe, composée de 4 membres, s'est renseignée auprès de différents contacts. Tout d'abord, notre projet exigeait une étude et une définition des besoins des utilisateurs de notre application, c'est pourquoi nous nous sommes renseignés auprès de M.Gauthier Hattenberger pour récolter ses besoins, ses contraintes, ses exigences mais aussi auprès d'autres utilisateurs de Paparazzi à l'aide de forums. A la fin de la rédaction du dossier des besoins, nous avons demandé une vérification de celui-ci par Mme Hélène Gaspard-Boulinc afin d'avoir son avis et ses conseils pour l'améliorer.

En parallèle, le développement exigeait certaines connaissances que nous ne possédions pas d'où plusieurs formations pour nous permettre de les acquérir. M Fabien André a permis à Florent de comprendre le fonctionnement du bus Ivy qui va être utilisé pour pouvoir créer une application communicante avec un drone. Guillaume apprenait à maitriser la librairie graphique SWT pendant que Christophe se renseignait sur jython et Alinoé sur les différentes méthodes de filtrage des données.

Avant la finalisation du dossier des besoins, la conception a pu être pensée et le dossier de conception a été analysé par Mme Catherine Letondal qui nous a permis de valider la spécification de notre application et ainsi nous permettre de commencer le développement de l'application. Dans le même temps, nous commencions le prototypage papier de l'interface que nous avons présenté à M.Gauthier Hattenberger afin d'avoir son avis.

A partir de là, les membres de l'équipe ont réalisé plusieurs tests pour appliquer les connaissances acquises. Ces tests nous ont permis d'abandonner et de confirmer certains outils. Nous avons abandonnés la librairie SWT au profit de SWING plus simple d'utilisation pour une performance presque équivalente et largement suffisante pour notre projet. Nous avons aussi abandonné Jython parce que ce binding Java/Python ne permettait pas d'utiliser les fonctions nécessaires à l'exécution de la calibration. Ce sont ces étapes qui représentent la transition entre la phase de conception et de prototypage à celle de la phase tests et implémentation.

b. Pendant la phase de tests et implémentation

Cette phase consiste à écrire les classes que l'on va utiliser et à les tester. Pour cela on a fait évoluer le système Paparazzi de nos ordinateurs pour qu'on puisse tester notre application dans le cas d'un drone réel à l'aide de M.Gauthier Hattenberger. Ainsi nous pouvions tester toutes les étapes de la calibration du drone c'est-à-dire de la communication du drone avec notre application à la réalisation de la calibration en passant par la récolte de données.

Cependant la création de notre interface nous a posé quelques soucis, essentiellement dans l'implémentation de threads multiples, alors nous avons rencontré M.Yannick Jestin qui nous a donné des conseils pour la mise en œuvre d'une telle interface.

De plus, la calibration a été très problématique (3 implémentations différentes de la calibration qui ont échoués). C'est pourquoi, nous avons rencontré le créateur du script de calibration M. Antoine Drouin afin qu'il puisse nous donner des conseils pour résoudre ce problème mais aussi pour nous expliquer son code cependant par manque de connaissance sur le sujet et par manque de temps nous avons opté pour un appel système qui reste une solution peu convaincante comme nous le verrons en partie III.

L'application étant en cours de développement, nous avions la nécessité de la tester et ainsi Christophe et Florent ont rédigé le plan de Vérification et Validation accompagné de fiches de tests afin de les réaliser par la suite.

3. Réalisé

Dans cette partie, nous allons détailler les implémentations des classes, les difficultés rencontrées et les solutions apportées mais aussi comment l'application est conçue. Pour cela, regardons le diagramme de classe de notre application :

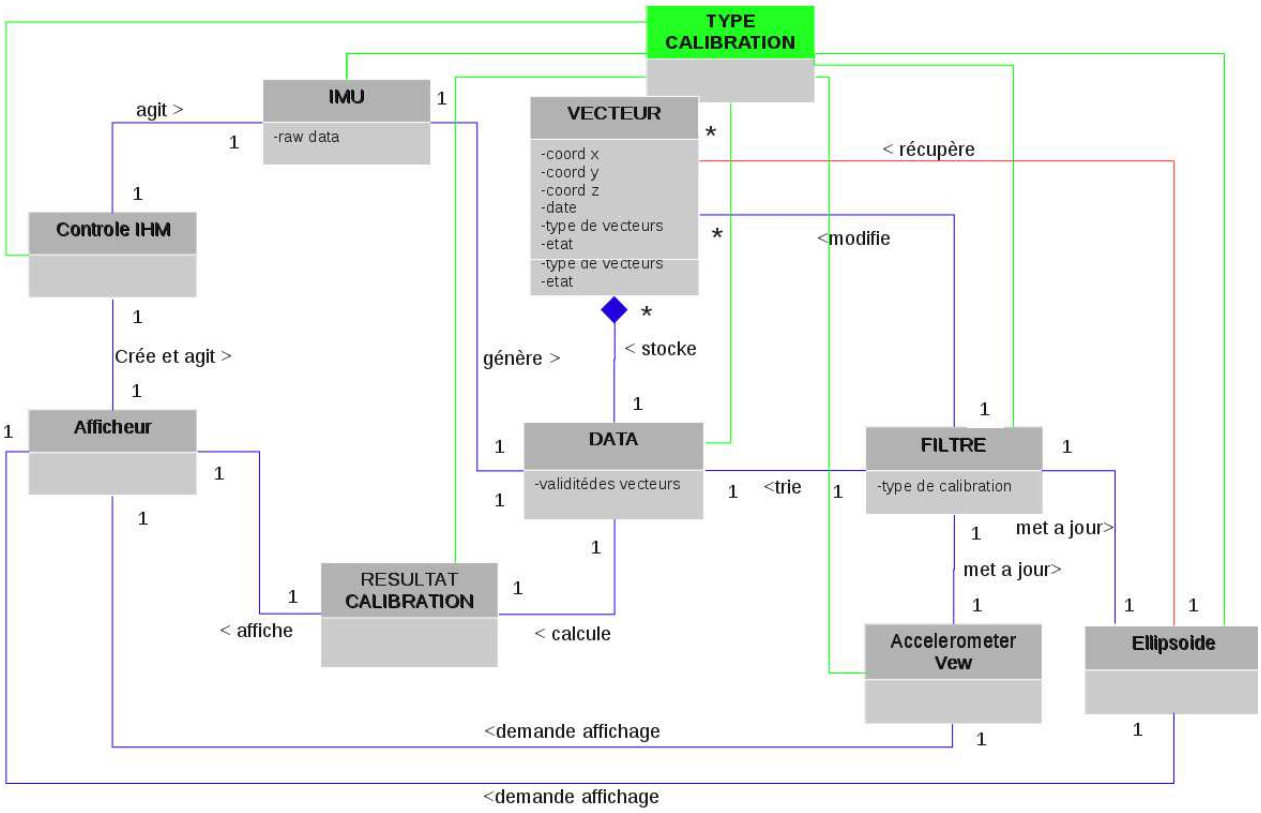
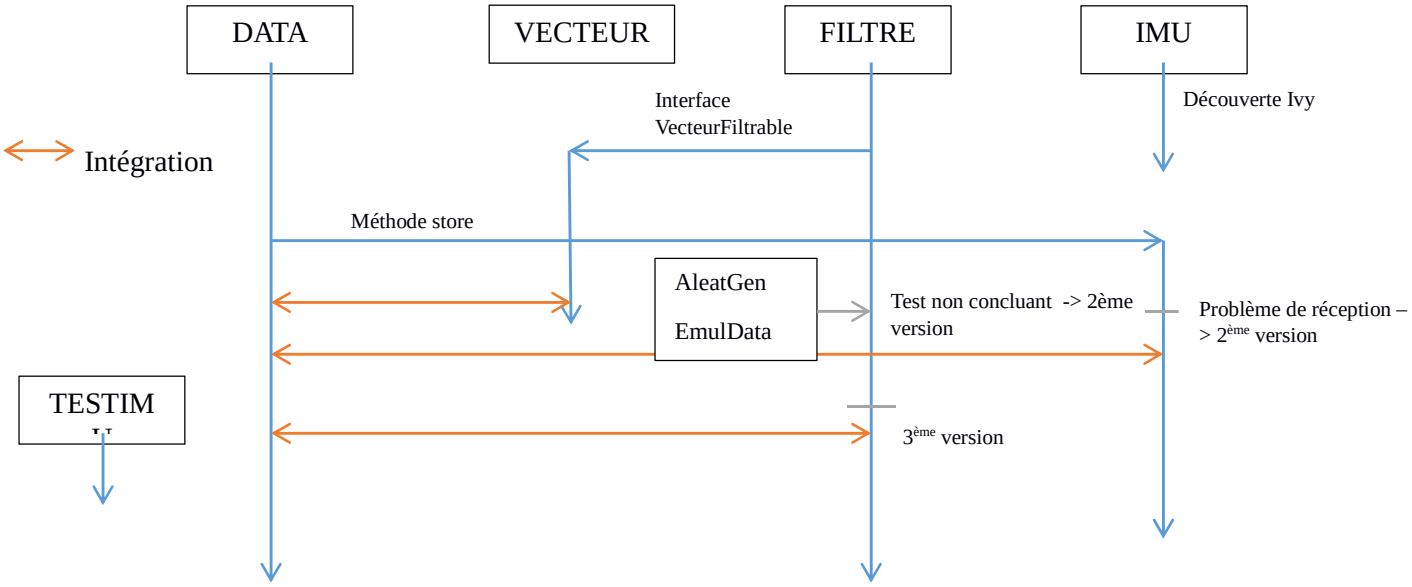


Figure 2 : Diagramme conceptuel

Nous pouvons regrouper le réalisé en 6 parties où chaque partie représente une étape dans le développement de notre application. Premièrement, nous avons commencé par la chaine IMU-Data-Vecteur-Filtre qui est le modèle de notre application. Ensuite, la première version de notre application a été créée contenant des actions et la possibilité de communiquer avec le drone. Puis début d'une visualisation de la récolte de données. Suivi de la création d'un start up pour lancer la vue de la

calibration magnétométrique. Après, nous sommes passés à la réalisation de la visualisation et du calcul du résultat. Et pour finir, la visualisation de la calibration des accéléromètres et son implémentation.

a. Réalisation de la chaîne IMU-Data-Vecteur-Filtre



Cette chaine fut la première étape dans notre développement et fut réalisée très tôt dans le projet. Florent a commencé à créer la classe IMU juste après sa formation au bus Ivy. IMU sert à envoyer les données du drone à notre application. La première version de celle-ci va envoyer des données à Data. Data implémente donc au préalable une méthode permettant de lui ordonner de stocker les données qu'elle lui envoie (méthode store de Data). Pendant ce temps Christophe et Alinoé travaillent en parallèle pour créer Data et Filtre. Filtre va contenir un certain type de vecteur : implémentant l'interface VecteurFiltrable qui va contraindre la classe Vecteur crée par la suite.

A partir de là, Vecteur a été intégré à Data ce qui veut dire que Data va stocker des données de type Vecteur. Nous allons créer alors 2 classes : AleatGen et EmulData qui vont nous permettre de tester si Filtre est bien conçu. Malheureusement pour nous les tests furent non concluants et l'algorithme a été modifié pour inclure un seuil de bruit (un point est refusé si la mesure du bruit sur les n précédents vecteurs dont ce point dépasse un certain seuil). Pendant ce temps-là, la classe IMU se voyait aussi modifiée à cause d'un mauvais filtrage ce qui entraina la modification des expressions régulières utilisées pour filtrer les données passant sur le bus.

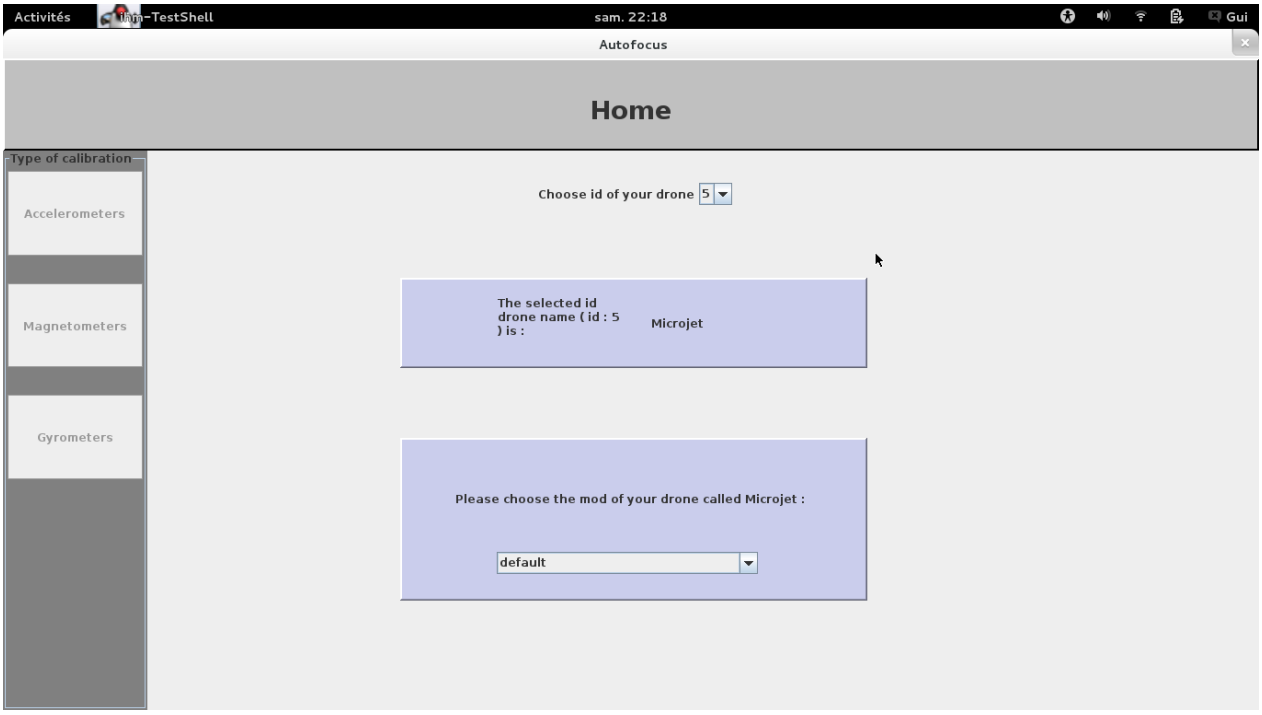
Quand toutes les versions furent fonctionnelles, Data, Filtre et IMU furent intégré ensemble et une autre classe TestIMU fut créé pour tester cette intégration.

b. Première version de l'application

Après avoir fait plusieurs essais avec la librairie graphique SWT, nous nous sommes aperçus que cette librairie était beaucoup trop complexe pour un gain très mineur. Ainsi la librairie graphique Swing fut privilégiée pour remplacer SWT. Guillaume a alors commencé à concevoir notre première application. Elle contenait les premières actions pour réaliser les différents types de calibration (changement de panel, boutons fonctionnels) sans pour autant avoir la visualisation de ces calibrations.

Alinoé quant à lui tentait de faire un affichage en temps réel avec swing avec succès et Florent pour répondre aux besoins croissants de l'IHM de notre application créa trois classes différentes de IMU : IvyIdListener, IvyConfigListener, IvyRawListener qui en réalité collectaient de la même façon les données du drone mais les traitaient de façon différente (on s'en apercevra plus tard dans le développement). Ensuite, nous avons eu la volonté de récupérer les modes possibles du drone pour permettre à l'utilisateur de les changer suivant ses désirs d'où la création de extractRawData. Ainsi notre application pouvait déjà obtenir l'id de tous les drones présents sur le bus, obtenir son nom et enfin changer son mode pour le passer en mode Raw-Data et enfin activer la possibilité de la calibration.

En parallèle de tout cela, Florent et Christophe développaient Ellipsoide et EllipseV2 : deux implémentations différentes au niveau de la chaine d'affichage dont leurs buts étaient de réaliser la visualisation de la représentation de l'ellipsoïde.



c. Visualisation de la récolte de donnée

Ellipsoïde et EllipseV2 étaient non fonctionnels car l'algorithme de régression en ellipsoïde du nuage de point généré par IMU ne fonctionnait pas bien. Alors Florent a conçu Sphere : sa différence est qu'il utilise un Min-Max dans les 3 directions de l'espace au lieu d'une régression. A l'aide de GuiHelper qui a servi au débugage pour tout ce qui est affichage, nous avons pu rectifier l'affichage de la sphère. La Sphère est représentée par des zones, que nous avons choisi de découper de manière angulaire pour une raison de simplicité. Ces zones sont ensuite définies par une liste de points qui sont

enfaite la projection 2D des points du contour de ces zones. La projection que nous avons choisi d'utiliser est celle de Mollweide encore une fois pour des raisons de simplicité, mais aussi car elle permet une bonne visualisation de l'ensemble de la sphère. Chaque zone contient une densité Density qui détermine la couleur de la zone en fonction de la surface de la zone et du nombre de points dans la zone. Nous avons privilégié un code couleur classique : La zone est de couleur rouge lorsque le quota de points dans la zone est insuffisant et vert lorsqu'il l'est.

Avec cette avancée significative, nous nous sommes aperçus que les inputs nécessaires à la mise à jour de accéléromètres et sphère n'étaient pas les mêmes (accéléromètres nécessitent plus d'inputs car en plus d'une sphère elle contient un barre de progression et des consignes). C'est pourquoi, deux nouvelles classes ont hérité de Filtre pour couvrir la calibration des magnétomètres et celle des accéléromètres : FilterAccel, FilterSphere.

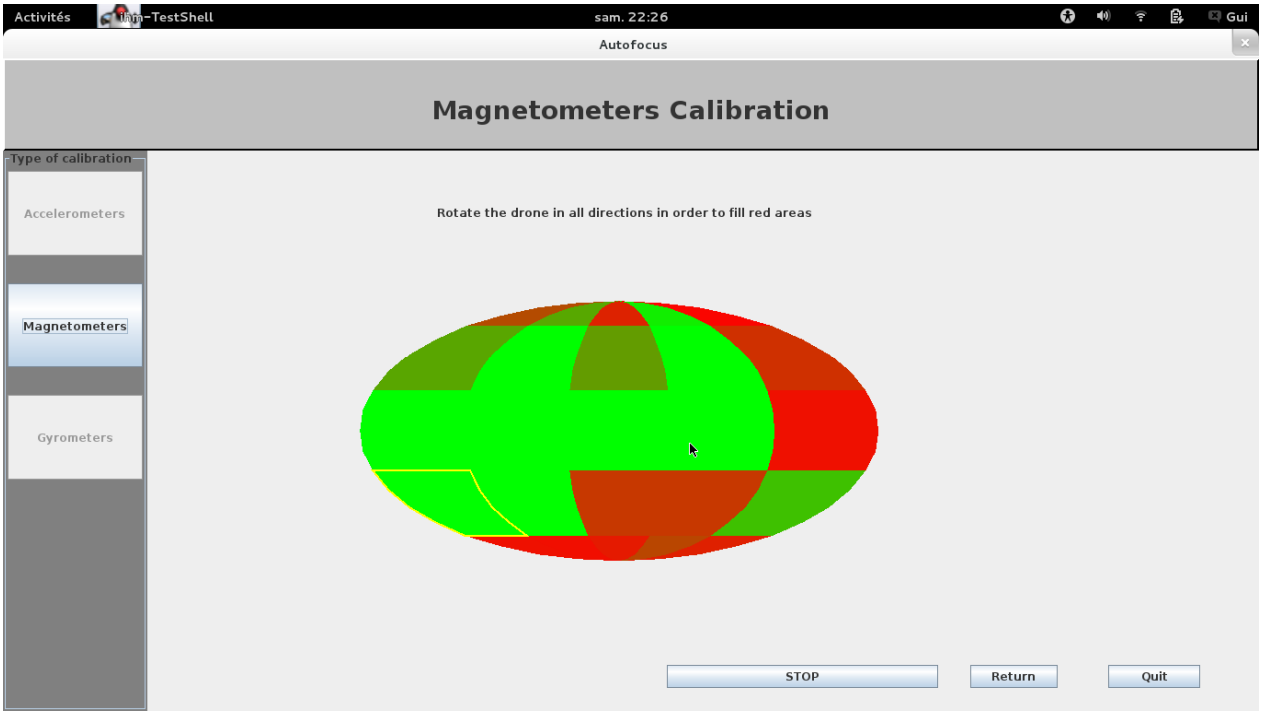
A ce stade du développement de l'application, l'application peut seulement communiquer avec le drone certaines informations mais celles-ci ne possèdent pas encore de visualisation de la récolte de données car elles n'ont pas encore été intégrées : c'est la prochaine étape

d. StartUp

Dans un souci d'intégration de la visualisation, une classe StartUp a été créée pour tout d'abord s'intéresser au cas de la calibration des magnétomètres. Elle va permettre d'instancier les différentes classes utiles à la calibration des magnétomètres tels que DATA, FilterSphere, Sphere tandis que l'IMU sera instanciée dans le Shell et envoyée dans l'appel à StartUp. Cette étape d'intégration nous a posé des problèmes car elle utilise un nouveau thread qu'il faut lancer au préalable dans le Shell. On va alors avoir notre première réelle visualisation de collecte de données magnétométriques. Après des tests avec le principe de rejeu concluant (à l'aide de Sender), nous avons présenté notre avancée au laboratoire drone pour la tester avec un drone réel et après ajustement le test fut réussi et les membres du laboratoire enthousiaste.

D'autre part, nous nous sommes (enfin) aperçu grâce à M.Gauthier Hattenberger que les 3 classes IvyIdListener, IvyConfigListener, IvyRawListener avaient le même comportement mais que la seule différence était le traitement finale des données. Ainsi, nous les avons regroupés en une seule classe IMU qui gère toutes les interactions avec le bus de données.

Afin de pouvoir réaliser la calibration, nous avons besoin d'appliquer des algorithmes de minimisation de fonctions. La première version de Calibrate utilisait une librairie intégrée sans succès. Ce fut le début de la principale difficulté à laquelle nous avons été confrontés.

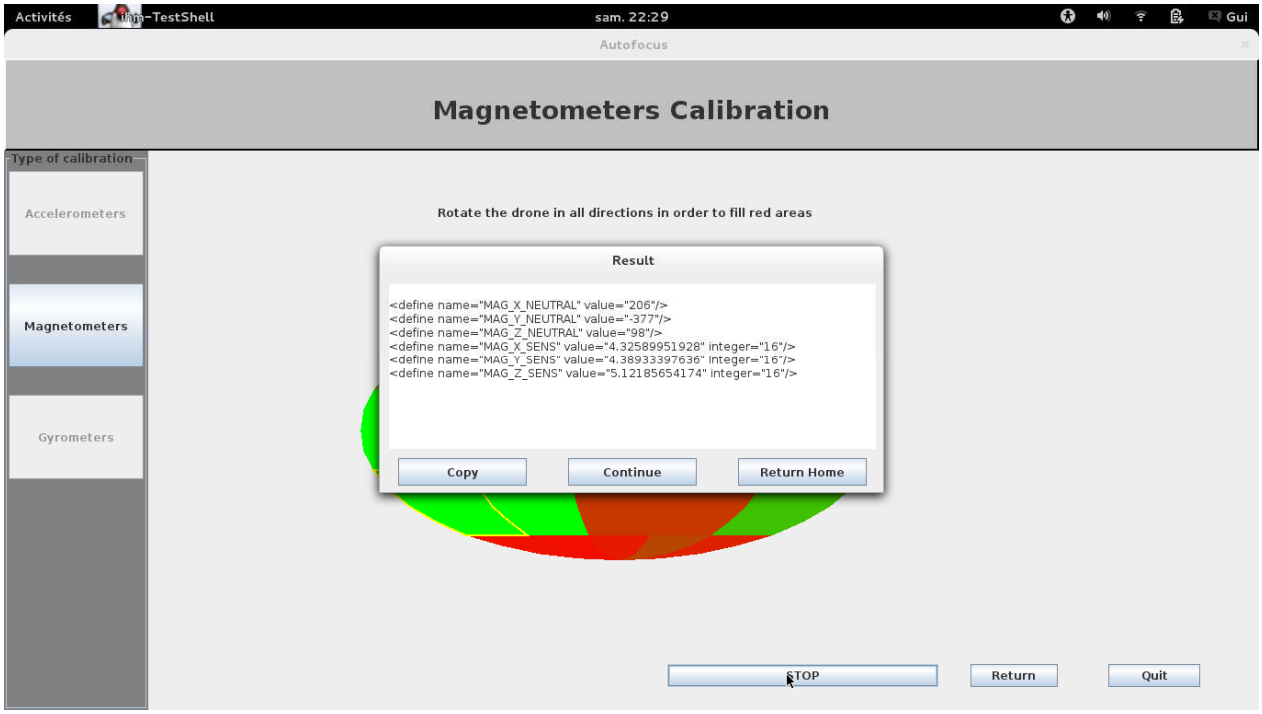


e. Affichage du résultat

La calibration n'étant pas fonctionnelle, du retard a été pris pour corriger ce problème. Ainsi une classe Calibrate2 fut créée et utilisait une fonction de minimisation codée à la main. Après plusieurs tentatives à l'aide de JeuDeTest pour faire fonctionner cette classe, nous nous sommes renseignés auprès de M. Antoine Drouin mais par manque de temps et de connaissances sur le sujet nous avons abandonné l'idée de concevoir l'algorithme de calibration en JAVA et nous allons juste faire un appel système pour appeler les algorithmes de calibration Python même si cette solution nous parait peu convaincante.

Le but final de notre application est de rendre des paramètres XML à copier dans l'Airframe. Pour cela, nous allons utiliser une classe Result, qui utilise TextTransfer afin d'afficher le résultat de la calibration à l'écran et offrir la possibilité à l'utilisateur de copier le résultat, de continuer ou encore de quitter la calibration en cours. La difficulté de cette étape est l'utilisation d'un nouveau thread pour afficher ce résultat. Pour résoudre ce problème, nous avons étudié plus précisément le fonctionnement de Java, notamment grâce à Internet.

A la fin de cette étape, notre application est capable d'afficher le résultat d'une calibration magnétométrique de manière simple et efficace.



f. Ajout de la visualisation de la calibration des accéléromètres

La dernière étape du développement de notre application et non la moindre est l'ajout de la visualisation de la calibration des accéléromètres. Pour cela, nous allons concevoir une classe AffichAccel qui va être l'intermédiaire entre FilterAccel et les différents composants de la vue accéléromètre (sphère, barre de progression, texte d'instruction). FilterAccel va mettre à jour AffichAccel au fur et à mesure que lui-même va mettre à jour la sphère et les autres composants pour avoir un affichage en temps réels de la récolte de données des accéléromètres.

Afin de mettre en place cette vue, nous avons, comme pour la sphère seule, utilisé la classe StartUp qui va être lancée lors de l'action du bouton Accelerometers et contenir FilterAccel, Data, et IMU comme pour la calibration des magnétomètres.

A la fin de cette étape, nous avons une application fonctionnelle qui sera après quelques petites modifications notre prototype final.

III. Retours

Après avoir détaillé les besoins auxquels répondre, nous avons détaillé le déroulement du projet, la façon dont nous avons mis en place cette réponse et ses différents aspects, il convient maintenant de dresser un retour d'expérience sur ce projet. Cette partie ne vise pas à se substituer au plan de vérification et de validation attenant, elle se contente de dresser un rapport d'avancement qualitatif sur le projet, l'état dans lequel l'équipe estime avoir porté le projet. Elle s'appuiera sur les tests qualitatifs effectués en phase de développement mais ne pourra pas pleinement s'appuyer sur le plan de validation. En effet, celui-ci aurait dû être mis en place peu après la phase de conception sanctionnée par le dossier de conception. Il aurait permis de valider au fur et à mesure la spécification de chaque

classe et de chaque groupe de classe pour valider de manière formelle un avancement du projet. Les contraintes de l'emploi du temps ne nous ont pas permis de le mettre en place dans les temps. Cette partie vient donc tenter de combler cette lacune. Etant donné le nombre d'heures allouées entre la remise de ce rapport et la soutenance du projet, il ne fait aucun doute que le plan de validation joint à ce rapport et réalisé à posteriori sera effectué et permettra de dresser un réel tableau de l'avancement final du projet. Il permettra de plus de vérifier que chaque classe remplit bien son rôle et est correctement implémentée.

1. Avancement final

On va ici reprendre le plan des besoins explorés en partie I.2 et les valider point par point. Cette partie, avec la III.2 forment en quelque sorte le rapport d'un plan de vérification et de validation conçu en début de projet mais non formalisé.

a. Scénarios globaux

Les scénarios globaux ont été testés en utilisation avec un utilisateur expert sur une plateforme réelle. Au moment des tests, vendredi 07/06/13, les résultats n'étaient pas encore fonctionnels et la vue accélérométrique non encore intégrée. Le test a donc essentiellement porté sur la capacité de la vue à apporter un retour util à l'utilisateur quant à la qualité de sa calibration. Le test a été concluant, après modification de certaines valeurs de l'application jugées trop élevées telles que le nombre de point par zone qui forçait un arrêt long sur certains endroits.

Le même jour un test d'intégration des résultats a été effectué par l'équipe sur l'intégration des résultats pour la calibration magnétométrique. Le test a été concluant, l'application avait la capacité de réaliser le use case calibration magnétométrique de manière complète.

A ce jour, la vue de la calibration des accéléromètres n'a été testée qu'en simulation sur plusieurs jeux de données. Son intégration est effectuée. La simulation en question est concluante, les résultats fournis sont en accord avec la méthode précédente. Il n'a cependant pas été possible de tester sa capacité à apporter un guidage suffisant à l'utilisateur même si le prototypage papier était concluant. Un test en condition réelle est prévu pour le lundi 10/06/13. Il doit s'inscrire dans la campagne de test du plan de vérification et de validation. Il devrait permettre de valider le deuxième scénario.

Du point de vue de l'avancement sur les scénarios globaux l'équipe se trouve donc en fin de réalisation, le prototype réalisant correctement ces tâches. On considère donc B1.1 et B1.2 comme validés.

Ces besoins globaux étant validés, le B1.4 concernant le choix de la calibration est lui aussi validé, l'utilisateur expert pouvant déjà opérer un choix lors du test de vendredi. Les résultats sont générés correctement sur les jeux de tests utilisés d'où la validation de B1.5.

Concernant B1.6, un fichier de log est bien généré, il se nomme « test.data » et est stocké dans «home_paparazzi » de l'utilisateur. Il contient l'ensemble des messages relativement à la calibration précédemment effectuée. Il peut être rejoué à l'aide de Sender mais cette possibilité n'est pas directement proposée à l'utilisateur. Ainsi, si B1.6 est correctement effectué, on ne peut pas réellement conclure qu'il est validé. Des modifications sont à apporter à l'application sur ce point. La campagne de test de validation de la semaine du 10/06/13 devrait notamment permettre de valider ces modifications.

La conclusion globale à ce moment du projet concernant les scénarios globaux est donc la réalisation d'une réponse correcte à ces besoins qui rendent donc le système fonctionnel.

b. Performances attendues

Concernant les performances attendues, le bilan est lui aussi plutôt positif.

L'interface graphique proposée à l'utilisateur est jugée satisfaisante par le client qui a réussi en tant qu'utilisateur expert à réaliser facilement une calibration en respectant les délais quantifiés. Les temps de recherche des fonctions est quasi nul, la latence de l'application n'excède pas 2 secondes sur la machine de test. De plus, des membres du labo ont regardé le test en cours. Leur retour sont positifs, ils étaient capables sans formation de donner une approximation de l'avancement de la calibration en cours en regardant l'écran. On peut donc considérer les besoins B2.1, B2.2, B2.7 comme validés en l'attente des tests plus exhaustifs.

Le test effectué dans l'après-midi le même jour a permis de valider la reprise et l'arrêt ainsi que la génération des résultats sur application réelle et sur les jeux de tests à notre disposition. On valide ainsi les besoins qui sont liés à ces fonctionnalités soit B2.3, B2.4, B2.5. Les résultats, dans la manière où ils sont présentés, sont très faciles à récupérer et à réutiliser. On valide donc aussi le besoin B2.6 sans plus de tests.

Concernant B2.8 qui indique que le système doit indiquer l'ensemble des manipulations à effectuer, le bilan est plus difficile à établir en l'absence de tests utilisateurs qui permettraient de détecter d'éventuelles hésitations quant à la procédure à suivre.

Le bilan pour B2.6 est mitigé, si le lancement est grandement facilité par l'intégration du choix du mode dans la fenêtre de l'application comme on peut le voir en consultant les scénarios du dossier des besoins, l'indépendance au système d'exploitation n'est pas forcément acquise ni même l'intégration totale des composants. En effet, l'application utilise des commandes UNIX. Ce problème n'est actuellement pas gênant car PAPARAZZI ne peut s'utiliser que sur des systèmes de ce type. En revanche, l'intégration des scripts de calibration n'a pas été faîte. Cela force la machine hôte à posséder Python afin d'exécuter les scripts de calibration. A nouveau Python est nécessaire au fonctionnement de PAPARAZZI, ce problème n'est donc pas gênant. En revanche, la nécessité pour l'utilisateur d'ouvrir son fichier Airframe pour y placer les paramètres de calibration est problématique. Elle nuit à l'unité de l'application. En définitive, il n'est pas possible de conclure positivement pour ce besoin.

Globalement, les performances attendues sont correctement réalisées. Une campagne de test plus importante est à prévoir pour le confirmer mais hors besoins B2.6 et B2.8 l'ensemble est correct. Le besoin B2.8 sera testé la semaine du 10/06/13 et des solutions pour apporter une réponse au besoin B2.6 sont proposées en III.3.

c. L'interface

Si, de manière empirique, ces besoins semblent résolus, aucun test réalisé pour le moment ne nous a permis de conclure quant aux besoins B3.1 et B3.2 qui considèrent essentiellement des tests sur des utilisateurs sans expérience avec le système. L'équipe n'a aucune difficulté à manipuler le logiciel et l'utilisateur expert n'a montré aucune hésitation quant à son utilisation ce qui laisse présager de bonne performances. On attendra néanmoins la prochaine phase de tests pour conclure quant à ces besoins.

d. Les contraintes

Le système a connu des tests sur plusieurs ordinateurs différents durant son développement, sa dépendance au matériel en terme d'exécution est donc testée et est très limitée. La qualité des résultats est strictement la même sur les jeux de tests. On peut donc valider, sans plus de procès, les besoins B4.1 et B4.2 en attendant d'éventuels autres tests.

Concernant B4.3, le manque de mesures temporelles sur l'actuelle procédure nous rend incapable de la comparer à la nouvelle, il convient donc de spécifier une procédure de test et pour cela de se référer au dossier de validation.

Le problème concernant le B4.4 est le même, il nous est impossible de vérifier facilement que l'objectif quantifié est atteint. On attendra donc la réalisation d'une fiche de test de validation pour conclure sur ce besoin.

En conclusion, le bilan des contraintes est pour le moment mitigé. Si l'expérience de l'équipe pousse à l'optimisme, l'absence de tests réels est problématique pour une réelle validation. Puisque ce n'est pas réellement le sujet de cette partie qui envisage plutôt de faire un récapitulatif de l'avancement du projet, nous conclurons sur un bilan positif sur les contraintes avant d'aborder le bilan global.

e. Bilan d'avancement

Nous venons donc de faire un récapitulatif point par point de la réponse apportée aux besoins définis au début du projet. Si ce bilan demande encore, comme on l'a dit, à être confirmé par une campagne de tests en bonne et due forme, il nous permet déjà de tirer plusieurs conclusions.

La première c'est que l'avancement du projet est bon, on peut l'estimer à 95%. Les quelques fonctionnalités non encore développées pouvant, semble-t-il être ajoutée sans surcoût trop important c'est un bilan positif en terme de réalisation qui est à retenir.

La deuxième est que le retard que l'on a vu précédemment, n'a pas posé de problème du point de vue de la validation. Si des raccourcis ont été pris pour rattraper le temps perdu, ils ont été pris dans l'implémentation des classes et ce sans soucis pour le résultat final hors besoin B2.6.

La troisième est que le prototype livré aujourd'hui est fonctionnel. Si des améliorations restent à apporter et des tests à faire pour confirmer les assomptions faîtes ici, le système est d'ores et déjà utilisable pour réaliser des calibrations. De plus, la quasi-totale réalisation des performances attendues, place normalement ce système à un meilleur niveau que la procédure actuellement utilisée ce qui nous pousse à dire que les objectifs de ce projet ont été tenus.

2. Difficultés

On va maintenant chercher à expliquer pourquoi il nous est pour l'instant impossible de conclure de manière plus approfondie et quelles sont les améliorations à attendre pour la soutenance.

a. Défauts de validations

La validation et la vérification, dans le but d'en tirer un bénéfice durant la phase de codage, doivent être mises en place au plus tôt, comme il nous l'a été enseigné en cours, chaque document, chaque partie du code doit avoir en parallèle son propre projet de vérification et de validation. Cependant, l'emploi du temps ne nous a pas permis de recevoir cette information à temps. Des tests ont pourtant été faits, par habitude, lors des premières phases du projet. Ainsi Vecteur et Data ont été testés en utilisant un devices d'IMU, de même pour filtre en utilisant un devices de DATA. Puis leurs interactions ont été testés permettant, sinon de les valider du moins de les vérifier partiellement.

Malheureusement, cette méthode n'a pas été appliquée de manière systématique, une perte de temps a donc été constatée notamment sur la classe IMU. Cette classe recueille des informations sur un bus de données en les filtrant avec des expressions rationnelles. Nous avons eu la mauvaise surprise de découvrir après des tests sur DATA que ces expressions rationnelles étaient fausses. La découverte de cette erreur a été couteuse puisque toutes les interactions entre les deux classes avaient été testées avant de s'intéresser aux problèmes dans la classe IMU qui semblait bonne après revue.

D'autre part, l'ajout a posteriori sur le projet de la validation tend à rallonger de manière conséquente la phase de terminaison du projet puisqu'une réelle campagne de validation et de vérification reste à faire ou, du moins, à réaliser en partie. Si des revues ont déjà été organisées de manière spontanée, aucune fiche de revue n'avait été mise en place, elles restent donc à refaire. De même écrire les tests blackbox à posteriori risque d'être plus coûteux pour le faire honnêtement.

Ces retards cumulés dans la mise en place de la validation viennent s'ajouter à ceux précédemment mentionnés. Le plan de projet et le WBS auraient donc dû être mis à jour convenablement. Cette modification n'a pas été effectuée. En effet, il nous a été rappelé la nécessité de faire évoluer ces éléments à 4 semaines du rendu. Les choses se sont donc précipitées, une mise à jour du planning a été effectuée, un compte rendu (en Annexe 1) a été envoyé à Mr Hattenberger et à Mme Gaspard-Boulinc afin de les informer de la situation du projet à cette date. Nous avons ensuite été informés de la nécessité du plan de V&V. Ce plan a été incorporé de manière informelle sans l'inclure au plan de projet. Un second rapport d'avancement a été fait de manière orale à Mr Hattenberger lors d'un test. Sans mise à jour du plan de projet qui est donc toujours dans l'état où il se trouvait 4 semaines avant la remise.

Ces écarts dans la gestion de projet se sont trouvés problématiques uniquement pour la gestion des indicateurs. En effet, la taille limitée du projet permet d'avoir en permanence une vision globale de l'avancement et des tâches à accomplir. Ils auraient néanmoins été problématiques dans un projet de plus grande ampleur. L'équipe en a pris note et veillera à ne plus reproduire ces écarts.

Le diagramme de Gantt joint avec ce rapport constitue donc un avancement à 4 semaines de la remise du projet. Le rapport d'avancement fait à l'oral à M. Hattenberger peut-être trouvé en annexe. Le plan de validation joint à ce dossier est donc réalisé a posteriori avec les nombreux biais que cela peut impliquer notamment la connaissance de l'implémentation dans les tests blackbox.

b. Performances attendues

Concernant les performances attendues, le bilan est lui aussi plutôt positif.

L'interface graphique proposée à l'utilisateur est jugée satisfaisante par le client qui a réussi en tant qu'utilisateur expert à réaliser facilement une calibration en respectant les délais quantifiés. Les temps de recherche des fonctions est quasi nul, la latence de l'application n'excède pas 2 secondes sur la machine de test. De plus, des membres du labo ont regardé le test en cours. Leurs retours sont positifs, ils étaient capables sans formation de donner une approximation de l'avancement de la calibration en cours en regardant l'écran. On peut donc considérer les besoins B2.1, B2.2, B2.7 comme validés en l'attente des tests plus exhaustifs.

Le test effectué dans l'après-midi le même jour a permis de valider la reprise et l'arrêt ainsi que la génération des résultats sur application réelle et sur les jeux de tests à notre disposition. On valide ainsi les besoins qui sont liés à ces fonctionnalités soit B2.3, B2.4, B2.5. Les résultats, dans la manière où ils sont présentés, sont très facils à récupérer et à réutiliser. On valide donc aussi le besoin B2.6 sans plus de tests.

Concernant B2.8 qui indique que le système doit indiquer l'ensemble des manipulations à effectuer, le bilan est plus difficile à établir en l'absence de tests utilisateurs qui permettraient de détecter d'éventuelles hésitations quant à la procédure à suivre.

Le bilan pour B2.6 est mitigé, si le lancement est grandement facilité par l'intégration du choix du mode dans la fenêtre de l'application comme on peut le voir en consultant les scénarios du dossier des besoins, l'indépendance au système d'exploitation n'est pas forcément acquise ni même l'intégration totale des composants. En effet, l'application utilise des commandes UNIX. Ce problème n'est actuellement pas gênant car PAPARAZZI ne peut s'utiliser que sur des systèmes de ce type. En revanche, l'intégration des scripts de calibration n'a pas été faîte. Cela force la machine hôte à posséder Python afin d'exécuter les scripts de calibration. A nouveau Python est nécessaire au fonctionnement de PAPARAZZI, ce problème n'est donc pas gênant. En revanche, la nécessité pour l'utilisateur d'ouvrir son fichier Airframe pour y placer les paramètres de calibration est problématique. Elle nuit à l'unité de l'application. En définitive, il n'est pas possible de conclure positivement pour ce besoin.

Globalement, les performances attendues sont correctement réalisées. Une campagne de test plus importante est à prévoir pour le confirmer mais hors besoins B2.6 et B2.8 l'ensemble est correct. Le besoin B2.8 sera testé la semaine du 10/06/13 et des solutions pour apporter une réponse au besoin B2.6 sont proposées en III.3.

c. L'interface

Si, de manière empirique, ces besoins semblent résolus, aucun test réalisé pour le moment ne nous a permis de conclure quant aux besoins B3.1 et B3.2 qui considèrent essentiellement des tests sur des utilisateurs sans expérience avec le système. L'équipe n'a aucune difficulté à manipuler le logiciel et l'utilisateur expert n'a montré aucune hésitation quant à son utilisation ce qui laisse présager de

bonnes performances. On attendra néanmoins la prochaine phase de tests pour conclure quant à ces besoins.

d. Les contraintes

Le système a connu des tests sur plusieurs ordinateurs différents durant son développement, sa dépendance au matériel en terme d'exécution est donc testée et est très limitée. La qualité des résultats est strictement la même sur les jeux de tests. On peut donc valider, sans plus de procès, les besoins B4.1 et B4.2 en attendant d'éventuels autres tests.

Concernant B4.3, le manque de mesures temporelles sur l'actuelle procédure nous rend incapable de la comparer à la nouvelle, il convient donc de spécifier une procédure de test et pour cela de se référer au dossier de validation.

Le problème concernant le B4.4 est le même, il nous est impossible de vérifier facilement que l'objectif quantifié est atteint. On attendra donc la réalisation d'une fiche de test de validation pour conclure sur ce besoin.

En conclusion, le bilan des contraintes est pour le moment mitigé. Si l'expérience de l'équipe pousse à l'optimisme, l'absence de tests réels est problématique pour une réelle validation. Puisque ce n'est pas réellement le sujet de cette partie qui envisage plutôt de faire un récapitulatif de l'avancement du projet, nous conclurons sur un bilan positif sur les contraintes avant d'aborder le bilan global.

e. Bilan d'avancement

Nous venons donc de faire un récapitulatif point par point de la réponse apportée aux besoins définis au début du projet. Si ce bilan demande encore, comme on l'a dit, à être confirmé par une campagne de test en bonne et due forme, il nous permet déjà de tirer plusieurs conclusions.

La première c'est que l'avancement du projet est bon, on peut l'estimer à 95%. Les quelques fonctionnalités non encore développées pouvant, semble-t-il être ajoutées sans surcoût trop important c'est un bilan positif en terme de réalisation qui est à retenir.

La deuxième est que le retard que l'on a vu précédemment, n'a pas posé de problème du point de vue de la validation. Si des raccourcis ont été pris pour rattraper le temps perdu, ils ont été pris dans l'implémentation des classes et ce sans soucis pour le résultat final hors besoin B2.6.

La troisième est que le prototype livré aujourd'hui est fonctionnel. Si des améliorations restent à apporter et des tests à faire pour confirmer les assomptions faîtes ici, le système est d'ores et déjà utilisable pour réaliser des calibrations. De plus, la quasi-totale réalisation des performances attendues, place normalement ce système à un meilleur niveau que la procédure actuellement utilisée ce qui nous pousse à dire que les objectifs de ce projet ont été tenus.

f. Perspectives de validation

Malgré ces problèmes, il reste possible de valider au moins partiellement ce projet via une campagne de test. Le détail de cette campagne, ses objectifs et ses modalités peuvent être trouvées dans le dossier des besoins attenant. Cette campagne se trouvera probablement validée lors de la soutenance ce qui devrait nous permettre d'y présenter les résultats.

On devrait alors pouvoir convenablement conclure concernant les besoins B2.6 et B2.8 notamment. En revanche, les problèmes de vérification des classes mentionnées précédemment ne devraient pas trouver de solution.

3. Perspectives

De nombreuses améliorations semblent d'ores et déjà possibles pour le logiciel tant certains besoins semblent pouvoir recevoir une meilleure réponse. De plus quelques bugs sont à recensés sur la version actuelle du code. On distinguera donc deux parties : les Fix-me du nom du tag présent sur la ligne et les améliorations pour mieux répondre aux besoins.

a. Fix-me

Plusieurs problèmes apparaissent pour le moment. Nous n'avons pas eu le temps de les régler pour l'instant mais ils devraient être résolus pour la soutenance.

Le retour sur l'écran d'accueil ne détruit pas certaines classes. Sur des tests avec rejeu, des données parasites apparaissent sur la visualisation probablement dues au fait qu'une des classes de la simulation ne termine pas convenablement. Il suffirait d'appeler la méthode de terminaison de cette classe au moment adéquat pour résoudre ce problème. Le moment adéquat n'a simplement pas encore été trouvé.

Un autre problème est la détection des données raw sur le bus. Dans l'écran d'accueil, comme on a pu le voir, l'application attend la réception des données raw avant de débloquer les boutons de choix de la calibration. Cette détection est parfois problématique et peut nécessiter une relance de l'application. Un bouton de réactualisation des données dans la fenêtre d'accueil permettrait de régler le problème.

L'absence de consignes dans le cas des accéléromètres est actuellement figée. Pour l'instant le panel dédié aux consignes reste à « maintain your drone in a stable position not yet explored » Normalement, cette consigne varie en fonction de l'avancement de la barre placée sous l'ellipse. Cependant l'absence de données de calibration d'accéléromètres nous a empêchés de tester et de mettre en place les changements de consigne. Cet élément devrait être achevé sans soucis pour la soutenance.

b. Evolutions

Outre ces problèmes réglés incessamment sous peu, plusieurs améliorations nous sont venues à l'esprit durant la réalisation du projet. Parmi elles, trois nous paraissent intéressante.

La première consisterait à retravailler l'intégration des scripts de calibrations, les voies déjà explorées pouvant servir de base de départ. Une version intégrée de ces scripts réduirait les dépendances du système et permettrait de le rendre moins sensible aux variations de système d'exploitation.

La seconde est la création d'un système évitant à l'utilisateur de devoir lui-même copier-coller les résultats de la calibration. En effet, le fichier xml est accessible depuis l'application de calibration, il suffirait d'aller le modifier directement en faisant une sauvegarde du précédent.

La troisième est l'amélioration de la visualisation. Actuellement, la taille de la sphère affichée est fixe, or sa taille réelle évolue en fonction des mesures et certains points changent de zones suite à ce changement de taille. Cela entraine une difficulté à se repérer sur la sphère lors des premières secondes de la calibration.

Conclusion

Ce projet est donc une expérience formatrice tant sur le management et la gestion que du point de vue du code Java. Il nous a permis de nous rendre compte de la complexité que peut atteindre une application avancée et la nécessité d'un management de projet efficace dans ce cadre. La validation reste à faire mais on peut déjà conclure que l'application fournie permettra aux futurs utilisateurs un accès plus facile à la calibration de leur drone.