

ENAC

Dossier de conception final

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
23/05/2013

Table des matières

I.	Présentation du domaine.....	3
1.	Description du contexte.....	3
2.	Modèle du domaine	3
3.	Présentation des Use cases	5
II.	Description de l'application par ses Use Cases	7
1.	Phase de prise des mesures.....	7
a.	Stockage des mesures	7
b.	Affichage et filtrage des mesures	8
c.	Guidage utilisateur	10
2.	Arrêt temporaire et résultat.....	11
3.	Continuer ou quitter.....	12
4.	Start-Up et arrêt.....	14
a.	Démarrage de l'application	14
b.	Démarrage de la calibration.....	15
III.	Choix techniques issus de cette description	20
1.	Choix de conception d'interface graphique.....	20
2.	Choix de conception algorithmique.....	23
3.	Algorithmique de l'affichage	24
4.	Matrice conception-besoins.....	25
IV.	Annexes	27
1.	Cas d'utilisation.....	27
a.	Phase de démarrage	31
b.	Use case prise de mesure	32
c.	Use case calibration.....	33
d.	Use case fermeture du logiciel	34
e.	Conclusion.....	34
2.	Prototypage papier d'interface	35
a.	Prototypage accueil	36
b.	Prototypage interface accéléromètre	37
c.	Prototypage interface magnétomètre	40

Figure 1 : Diagramme du domaine.....	3
Figure 2 : Classes du modèle conceptuel	4
Figure 3 : Décomposition des use cases en opérations.....	6
Figure 4 : Diagramme séquence stockage magnétométrique	7
Figure 5 : Diagramme classe stockage magnétométrique	8
Figure 6 : Diagramme séquence filtrage et affichage.....	8
Figure 7 : Diagramme classe filtrage et affichage.....	9
Figure 8 : Diagramme séquence d'affichage des consignes idem précédent.....	11
Figure 9 : Arrêt temporaire de la calibration.....	11
Figure 10 : Diagramme classe pause dans la calibration magnétométrique	12
Figure 11 : Diagramme séquence reprise de la calibration magnétométrique.....	12
Figure 12 : Diagramme séquence arrêt calibration.....	13
Figure 13 : Diagramme classe arrêt calibration.....	13
Figure 14 : Diagramme séquence filtrage et affichage de la calibration des accéléromètres.....	9
Figure 15 : Diagramme classe filtrage et affichage de la calibration des accéléromètres	10
Figure 16 : Diagramme séquence start-up.....	14
Figure 17 : Diagramme classe start-up	15
Figure 18 : Création en cascade.....	16
Figure 19 : Création avec classe intermédiaire.....	16
Figure 20 : Diagramme séquence de démarrage de la calibration.....	17
Figure 21 : Diagramme classe de démarrage de la calibration	18
Figure 22 : Diagramme séquence d'arrêt.....	18
Figure 23 : Diagramme de conception intermédiaire	19

I. Présentation du domaine

Cette partie joue le rôle de résumé par rapport au dossier des besoins, dans l'hypothèse où ce dernier a été lu il n'est pas nécessaire de relire la présente partie. Dans le cas contraire, sa lecture est conseillée afin de pouvoir comprendre la logique sous-jacente à la réalisation du modèle conceptuel de l'application.

1. Description du contexte

Le projet vise à développer une partie du Framework PAPARAZZI dédiée à la calibration de certains des capteurs de calibration des drones. En effet, le Framework PAPARAZZI vise à développer un autopilote pour véhicule aérien léger et sans pilote (microdrones). Pour cela, il s'appuie sur toute une gamme de capteurs et notamment une centrale inertielle (IMU). Cette IMU a besoin pour fonctionner d'être d'abord calibrée. Cette étape comprend la calibration des gyromètres, des magnétomètres et des accéléromètres. La procédure actuellement utilisée consiste à enregistrer un certain nombre de manipulations faites sur le drone en aveugle (sans retour à l'utilisateur sur la pertinence des manipulations effectuées) dans un fichier de log. Un script est ensuite appelé sur ce fichier de log et génère les paramètres de calibration dont on peut seulement alors juger de la pertinence avant de les conserver en les copiant dans le fichier adéquat (AirFrame du drone) ou de les rejeter ce qui force à reprendre toute la manipulation de calibration.

La difficulté pour l'utilisateur est donc essentiellement de faire les manipulations, il aurait donc besoin d'un feedback pour le guider au cours de ces manipulations afin de pouvoir être sûr, dès la prise des mesures, que le fichier généré sera correct. D'autre part, le gain d'audience du Framework fait qu'un nombre croissant d'utilisateurs novices cherchent à construire leurs propres drones. Le risque est qu'un certain nombre soit aujourd'hui rebutés par cette manipulation peu intuitive. L'application devra donc aussi s'intéresser à ces utilisateurs.

Pour un plus grand nombre de détails, il est recommandé de se référer à la présentation faite dans le dossier des besoins.

2. Modèle du domaine

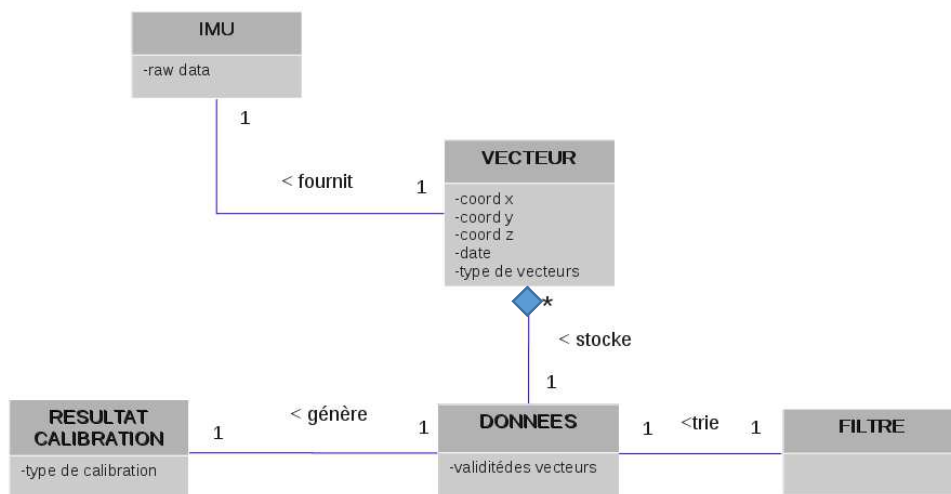


Figure 1 : Diagramme du domaine

Classe	Description	Justification
IMU (Inertial Measuring Unit)	L'IMU fournit les données nécessaires à la calibration. C'est elle qui fait le lien entre l'architecture physique du système PAPARAZZI, c'est-à-dire le drone et l'ordinateur, et le logiciel que nous développons.	Une seule classe qui permet de faire l'abstraction de toute la partie non-logicielle, l'interface du système n'étant conceptuellement pas nécessaire. Ainsi, dans la réalité le drone est relié à l'interface logicielle via le bus IVY qui, grâce à un mécanisme d'abonnement nous permet de récupérer les données émises par le drone concernant les mesures des accéléromètres et du magnétomètre inclus dans le drone (les mesures de son IMU). Seules ces données nous intéressent. La classe IMU est chargée de les récupérer. Elle représente donc une abstraction du drone et des mécanismes de communication qui lui sont associés. Le concept de drone apparaissant dans les scénarios de travail est donc englobé par cette classe.
VECTEUR	La classe Vecteur représente les données fournies par l'IMU.	Format au plus proche de la représentation réelle des vecteurs tridimensionnels qu'ils soient magnétique ou d'accélération.
DONNEES	Assure le stockage des vecteurs. Elle est l'expert d'information concernant les vecteurs.	Comme on peut le voir dans les scénarios actuellement cette fonction est réalisée par le fichier de log. Cette classe n'est que l'abstraction de ce fichier auquel on ajoute des méthodes génériques de traitements de données. On pourra donc la traiter sous une forme quelconque.
FILTRE	La classe qui assure le filtrage des données afin de ne conserver que les vecteurs corrects	On aurait pu l'intercaler entre les vecteurs et données comme son nom l'indique mais le fait que DONNEES soit l'expert d'information concernant les vecteurs voudrait qu'on la place plutôt ici. De plus, il apparaît dans les scénarios que le traitement de filtrage se fait sur les données groupées.
RESULTAT_DE_CALIBRATION	Cette classe représente le résultat de la calibration. Elle contient les informations permettant à l'utilisateur de finaliser la calibration	Le choix le plus contestable. L'idée de conception est que les résultats ne doivent pas être reliés à DONNEES pour des raisons de cohésion. Pour cela, de la même façon que pour FILTRE, on crée une classe dédiée qui prend en charge ces opérations et la gestion du résultat. On gagne en plus en matière de couplage car ces résultats apparaissent de manière indépendante au DONNEES comme on pourrait s'y attendre en voyant les scénarios.

Figure 2 : Classes du modèle conceptuel

Le diagramme précédent et les classes sont issus des scénarios de travail que l'on peut trouver dans le dossier des besoins. Le nombre réduit de classes s'explique par une grande similarité des scénarios de ces derniers qui implique aussi un nombre limité de use cases. L'absence de certaines classes peut sembler problématique. Ainsi, dans ce modèle le drone et les positions qu'il adopte durant les phases de la calibration n'apparaissent pas directement alors qu'ils étaient présents dans les scénarios de travail. En analysant de manière plus fine ceux-ci, il se trouve qu'ils doivent être abstraits ce que réalise IMU. En effet, les positions du drone ne sont en aucun cas fixées par notre application. La seule

information dont elle dispose est celle présente sur le bus IVY. On préfère donc faire disparaître le drone et l'utilisateur du modèle du domaine malgré leur rôle prépondérant dans les scénarios de travail.

D'autre part, notre application ambitionne de réaliser un retour sur les données collectées. Ce retour n'existe pas actuellement d'où l'absence d'une éventuelle classe chargée de le réaliser, cette classe apparaîtra en revanche dans le diagramme de l'application.

3. Présentation des Use cases

En effet, les use cases sont au nombre de deux, un troisième pouvant voir le jour lorsque l'implémentation des deux premiers sera terminée. Les deux premiers use case sont la calibration des magnétomètres et des accéléromètres, le troisième est celle des gyromètres. On pourra consulter les use cases en annexe. Pour chacun on distingue 4 phases décomposées en opérations génériques tant ils sont semblables. On reproduit ici cette décomposition dont la justification détaillée peut être trouvée dans le dossier des besoins.

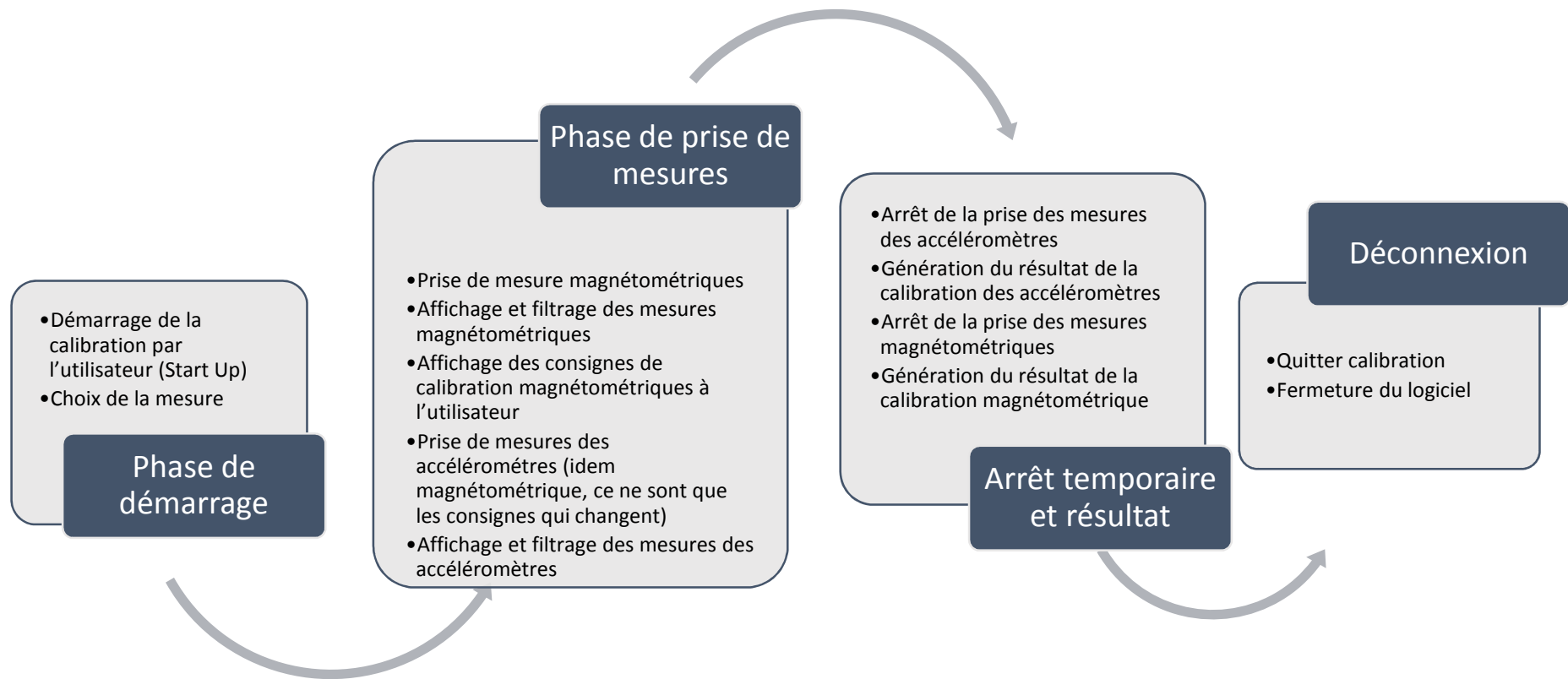


Figure 3 : Décomposition des use cases en opérations

II. Description de l'application par ses Use Cases

Nous allons ici décrire la construction du diagramme de classe de l'application. Pour cela nous allons analyser use case par use case les différentes classes qu'ils impliquent. On a pu le voir dans le dossier des besoins, le nombre de use cases est pour l'instant limité à deux, la calibration magnétométrique et la calibration des accéléromètres. Pour pouvoir pleinement prendre conscience des implications de ces use cases, nous allons réutiliser la décomposition en opérations et en étapes proposée dans le dossier des besoins. A chaque étape nous présenterons le diagramme séquence de l'opération puis le diagramme de classe associé. Certaines tâches font apparaître des variantes que l'on précisera alors. Les noms des classes ont été anglicisés pour faciliter la maintenance future du système. On ne démarrera pas avec la phase de démarrage pour faciliter la compréhension, les choix fait dépendant des autres étapes. On procédera de même pour la fermeture de l'application. On préfère donc commencer par s'intéresser par la phase de prise de mesure.

1. Phase de prise des mesures

a. Stockage des mesures

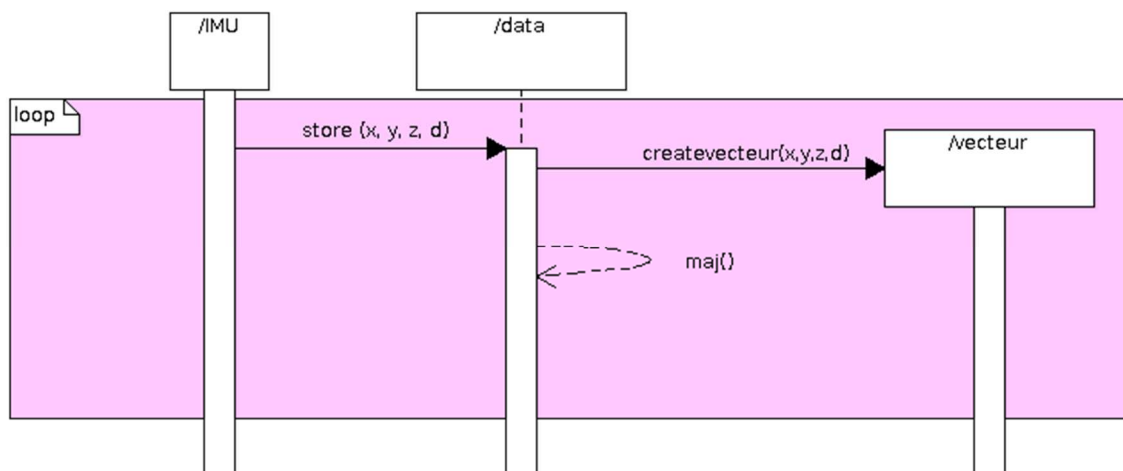


Figure 4 : Diagramme séquence stockage magnétométrique

L'opération d'ajout d'un vecteur dans DATA se fait logiquement suite à un appel d'IMU qui est l'expert en information sur les données des vecteurs (Pattern expert). En vertu du pattern de faible couplage (FaC dans la suite), la création des vecteurs est déportée dans DATA qui assume toutes les tâches de stockage permettant ainsi qu'IMU ne dépende plus du format de stockage. Le diagramme de classe se trouve donc un peu modifié pour prendre la forme suivante :

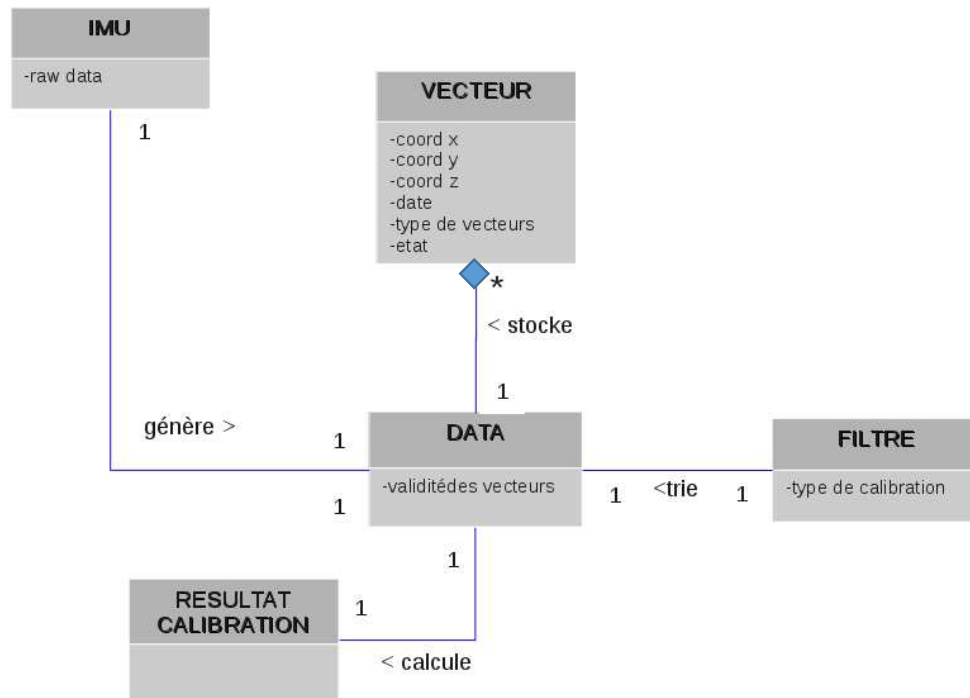


Figure 5 : Diagramme classe stockage magnétométrique

.L'opération de stockage sera la même pour les données accélérométriques, la seule difficulté est que DATA sache quel type de vecteurs elle doit stocker mais ce problème est à résoudre au moment du start-up

b. Affichage et filtrage des mesures

A chaque ajout de vecteur, on veut que les données soient filtrées en accord avec les nouvelles informations que ce dernier apporte. Pour cela, on fait déclencher par DATA un filtrage sur les données qu'il contient.

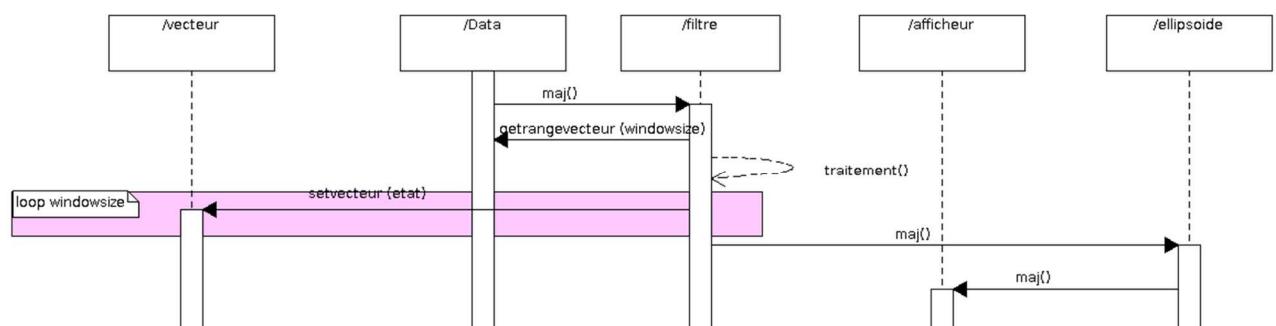


Figure 6 : Diagramme séquence filtrage et affichage

On le voit, la tâche de filtrage déclenchée par DATA se fait via une classe dédiée. Celle-ci présente l'intérêt de garder la forte cohésion (FCo) de DATA. Cette nouvelle classe se trouvera de plus indépendante de la structure utilisée pour le stockage tant que celle-ci implémentera une interface que

l'on pourra nommer filtrable. FILTRE assure le filtrage et déclenche la mise à jour de l'affichage. Cet affichage est distinct de filtrage puisque réalisant des fonctionnalités différentes et théoriquement indépendantes (FaC, FCo). On introduit la classe ELLIPSOIDE qui se chargera de générer le modèle de la vue dont le prototypage peut être consulté en Annexe 2. On voudra de plus que cet affichage se fasse à l'utilisateur. On crée donc une classe qui assure l'affichage sur demande de la classe précédente afin d'être en accord avec le pattern model-view-controller (MVC) dans lequel elle assurera le rôle de vue. Ce pattern étant introduit, on remarque alors qu'IMU tient le rôle de contrôle puisqu'il reçoit ses instructions depuis un élément extérieur à l'application qui est la centrale inertielle du drone.

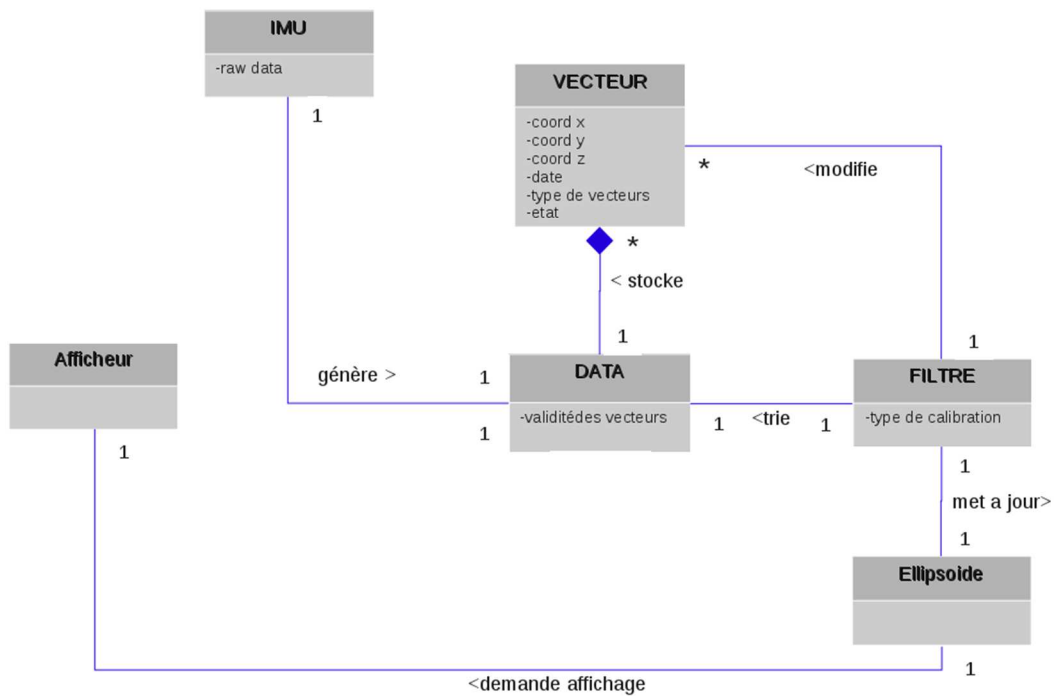


Figure 7 : Diagramme classe filtrage

La visualisation pour les accéléromètres ne sera pas la même. Pour cela on voudra à nouveau filtrer les vecteurs mais de manière différente et générer un affichage différent de celui de la calibration magnétométrique.

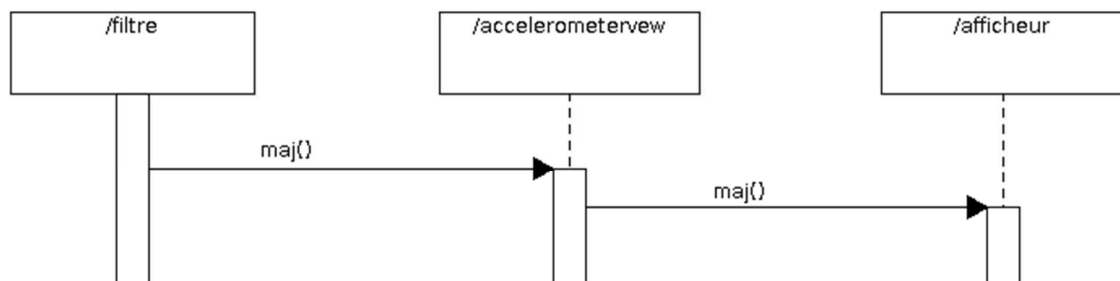


Figure 8 : Diagramme séquence filtrage et affichage de la calibration des accéléromètres

On va donc créer une nouvelle classe qui viendra tenir le rôle d'ELLIPSOIDE dans le cas de la calibration des accéléromètres en invoquant les mêmes patterns que précédemment : FCo et FaC.

Cette classe sera de nouveau reliée à AFFICHEUR pour respecter le pattern MVC. On la nommera ACCELEROMETRE_VIEW.

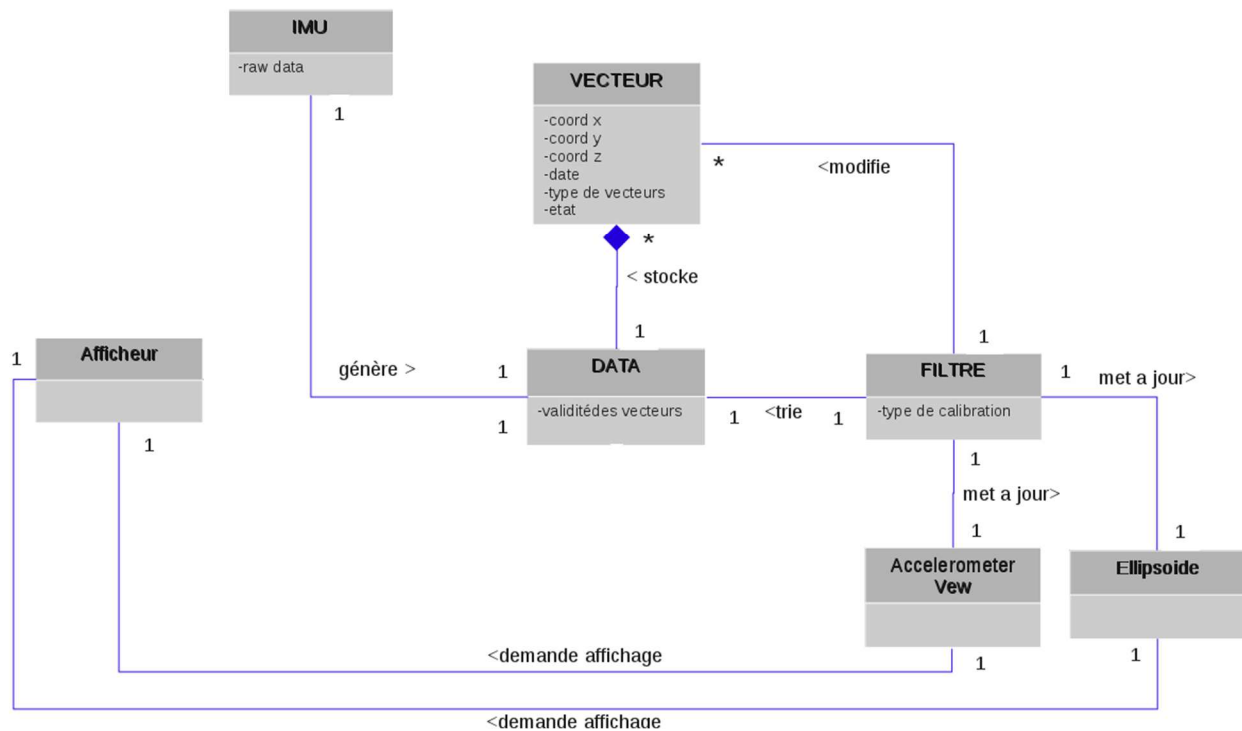


Figure 9 : Diagramme classe filtrage et affichage de la calibration des accéléromètres

c. Guidage utilisateur

Il est aussi demandé d'afficher des consignes afin de guider l'utilisateur. Ces consignes doivent indiquer les manipulations à effectuer sur le drone pour obtenir une calibration correcte. L'utilisateur pourra s'assurer qu'il suit correctement ces consignes grâce au retour graphique généré par l'application. La façon dont celles-ci sont gérées dépend donc du choix quant à la forme du retour de la prise des mesures. L'expert en information n'est donc pas DATA comme on aurait pu s'y attendre mais plutôt ELLIPSOIDE ou ACCELEROMETER_VIEW qui seront donc chargés de ce retour. On a donc deux classes chargées d'effectuer tout ce qui est consignes et retours à l'utilisateur : une dans le cas des magnétomètres et l'autre dans le cas des accéléromètres.

On constate alors que les fonctions de retour et d'affichage visant à préparer la vue se retrouvent dans la même classe (FCo). On a alors deux options qui diffèrent par leur forme, l'un propose un retour immédiat tandis que l'autre un retour différé. Ces deux chaînes d'affichage ont été testées en prototypage logiciel pour finalement opter pour la forme immédiate grâce aux bonnes performances de l'API graphique sélectionnée (voir III.1).

Concernant l'apparence de ces classes, un prototypage papier (voir IV.2) a été effectué et soumis à l'utilisateur (Mr Hattenberger). Ces retours nous ont orientés vers l'interface actuelle du système (voir IV.3)

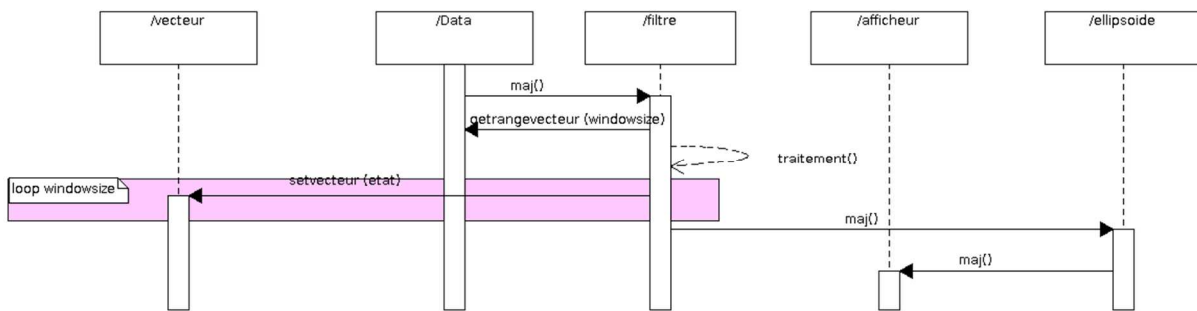


Figure 10 : Diagramme séquence d'affichage des consignes idem précédent

Le diagramme classe est donc identique au précédent.

2. Arrêt temporaire et résultat

L'utilisateur peut alors choisir d'arrêter les mesures pour générer un résultat de la calibration effectuée jusqu'alors ou faire une pause dans la calibration avant de poursuivre.

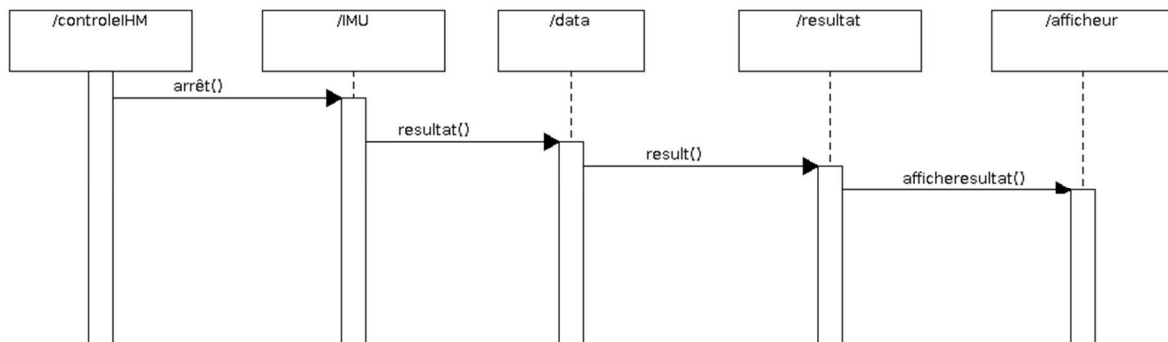


Figure 11 : Arrêt temporaire de la calibration

Cela impose donc que l'utilisateur puisse agir sur le logiciel via une autre interface que l'IMU afin de pouvoir signaler qu'il veut faire une pause. Via le pattern MVC, on constate que cela nécessiterait qu'IMU puisse détecter des inputs issu d'une interface homme machine(IHM) distincte de l'IMU via le clavier ou la souris. En raison des patterns de forte de cohésion et de faible couplage, on préfère créer une nouvelle classe dédiée au recueil de telles interactions : **CONTROLEUR_IHM**. La classe **RESULTAT** doit pouvoir demander l'affichage du résultat, on fait donc apparaître son lien avec **AFFICHEUR**. On en déduit le diagramme de classe suivant :

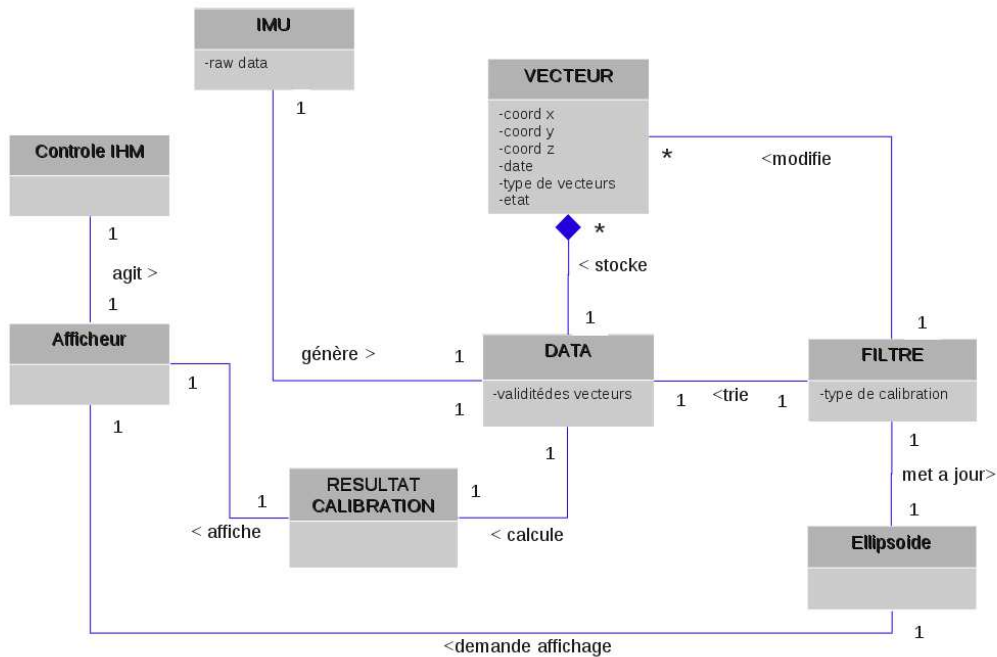


Figure 12 : Diagramme classe pause dans la calibration magnétométrique

3. Continuer ou quitter

Une fois le résultat intermédiaire généré, deux possibilités sont encore possibles pour l'utilisateur, d'une part reprendre la calibration et d'autres part, l'arrêter. Dans le premier cas, il suffit de relancer IMU :

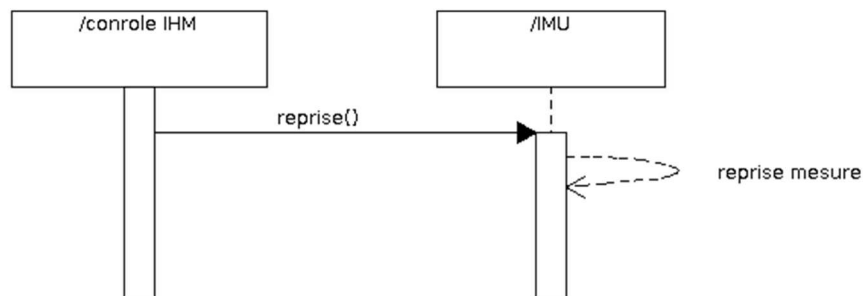


Figure 13 : Diagramme séquence reprise de la calibration magnétométrique

Le diagramme de classe reste inchangé, toutes les relations nécessaires à la reprise sont les mêmes que celles de l'arrêt, aucune classe n'ayant été détruite lors de l'arrêt temporaire.

Il peut aussi décider d'arrêter la calibration en se satisfaisant ainsi des résultats obtenus. Le diagramme séquence associé est alors le suivant :

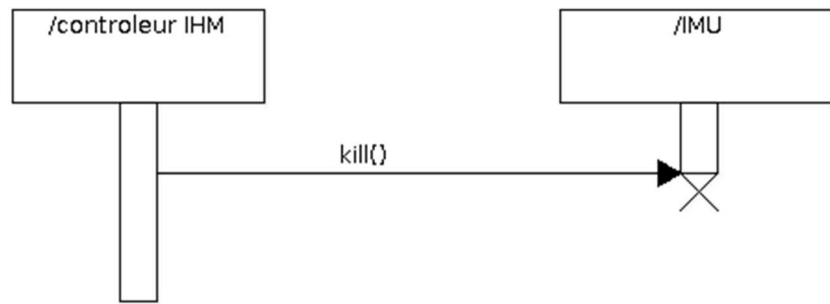


Figure 14 : Diagramme séquence arrêt calibration

L'arrêt se fait via une action de l'utilisateur. Puisque pour le moment seul IMU a une poignée sur le reste de l'application, on peut se contenter de détruire IMU. Le garbage collector gère ensuite la destruction des autres classes associées.

Le diagramme de conception obtenu suite à cette première phase de dessin UML est le suivant :

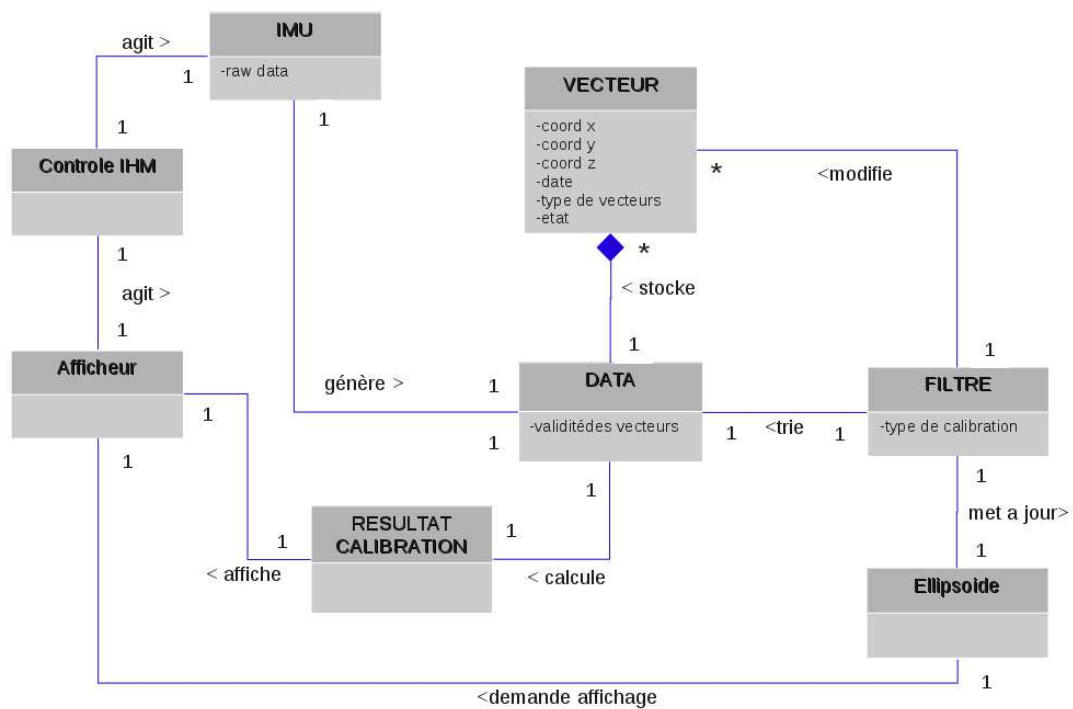


Figure 15 : Diagramme classe arrêt calibration

4. Start-Up et arrêt

On l'a vu aussi bien dans les scénarios de travail que dans les use cases le démarrage d'une calibration se déroule en deux temps, le lancement de l'application permettant de choisir la calibration que ce soit une console ou une interface dédiée puis le démarrage de la calibration elle-même. Il convient donc de distinguer convenablement ces deux temps de démarrage. Pour cela on va s'appuyer sur le pattern MVC qui a guidé en grande partie la conception jusqu'alors.

a. Démarrage de l'application

Après le démarrage de l'application, l'utilisateur doit pouvoir choisir la calibration qu'il souhaite effectuer. Pour cela, nous avons besoin de deux classes, une informant l'utilisateur qu'il dispose d'un choix et une seconde qui récupère le choix de l'utilisateur. Par rapport à la conception effectuée jusqu'alors, il semble logique d'utiliser l'AFFICHEUR en accord avec le pattern FCo puisque cette classe se charge déjà d'afficher toutes les informations à l'utilisateur. C'est CONTROLE_IHM qui ordonnera à AFFICHEUR d'effectuer cet affichage puisque c'est lui qui est chargé de récupérer les ordres de l'utilisateur en provenance du clavier et de la souris. Ainsi en vertu du pattern créateur, c'est CONTROLE_IHM qui va se charger de lancer AFFICHEUR. Il ne semble donc pas nécessaire d'avoir, pour le moment, une classe dédiée au démarrage de l'application.

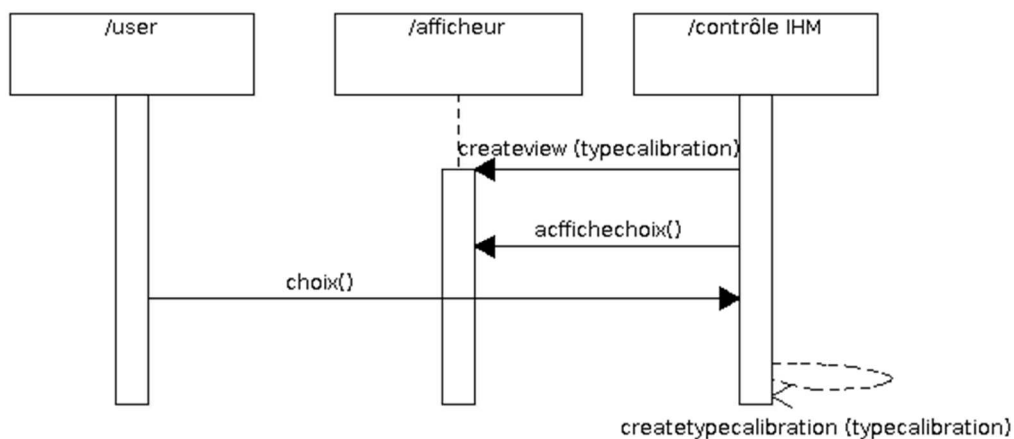


Figure 16 : Diagramme séquence start-up

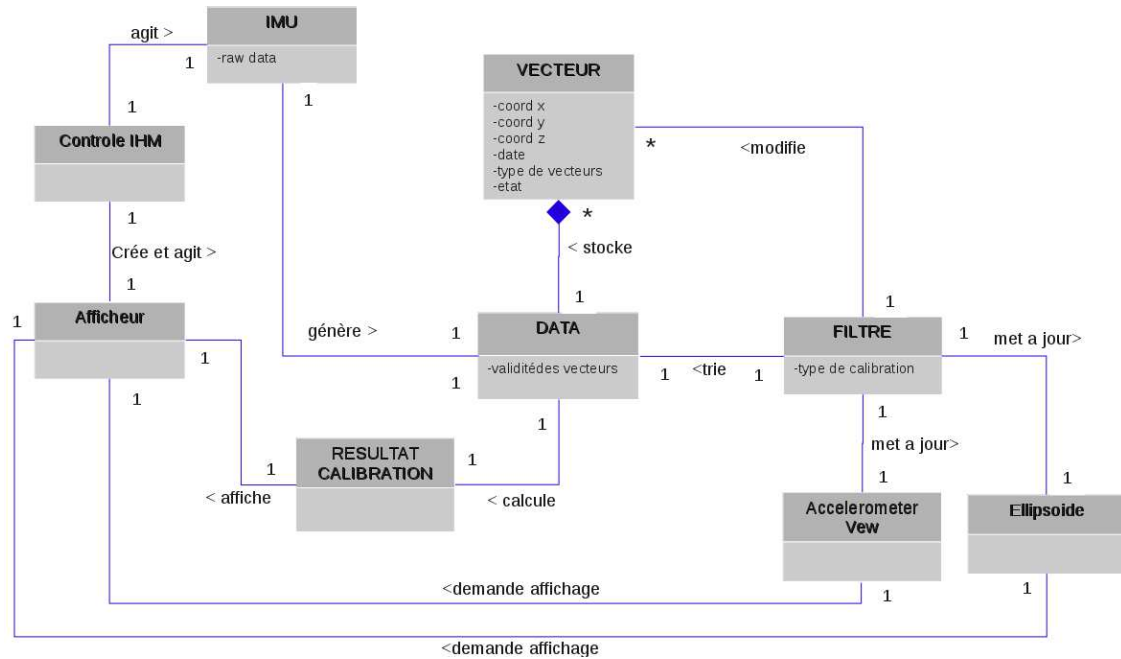


Figure 17 : Diagramme classe start-up

b. Démarrage de la calibration

Après le choix de l'utilisateur doit apparaître la fenêtre de calibration prête à recevoir les données sur l'input approprié de l'utilisateur. Pour cela il convient de créer les classes nécessaires soit toutes les classes du diagramme dans le mode de calibration approprié. De plus, on ne doit créer qu'ELLIPSOIDE ou ACCELEROMETRE_VIEW pour faire l'affichage adéquat. C'est clairement CONTROLE_IHM l'expert en information sur le type de calibration choisi, cependant lui faire créer toutes les classes qui dépendent de cette information est peu pertinent au vu du diagramme de classe. De plus faire une création en cascade forcerait toutes les classes à connaître l'afficheur utilisé puisque les classes qui apparaissent au bout des branches telles RESULTAT et ELLIPSOIDE doivent être rattachées à AFFICHEUR comme on peut le voir sur le diagramme suivant :

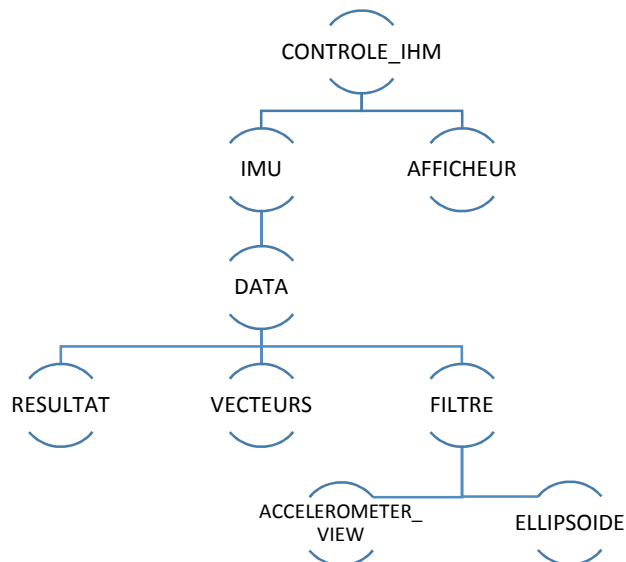


Figure 18 : Création en cascade

On préférera donc une troisième option consistant à créer temporairement une nouvelle classe permettant de créer l'ensemble du modèle avec les liens avant de les laisser interagir seuls.

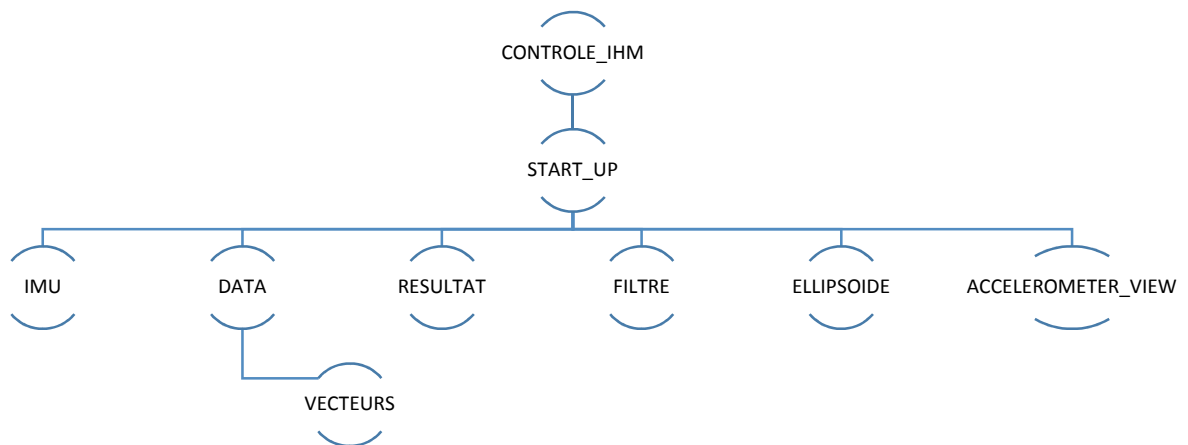


Figure 19 : Création avec classe intermédiaire

Cette dernière méthode permet, comme on le voit sur le graphique ci-dessus de créer facilement les liens avec pour seul coût supplémentaire la création d'une nouvelle classe. Un objet de ce type n'aura plus de raison d'exister après cette phase d'initialisation et pourra donc être détruit. On obtient alors le diagramme d'initialisation suivant :

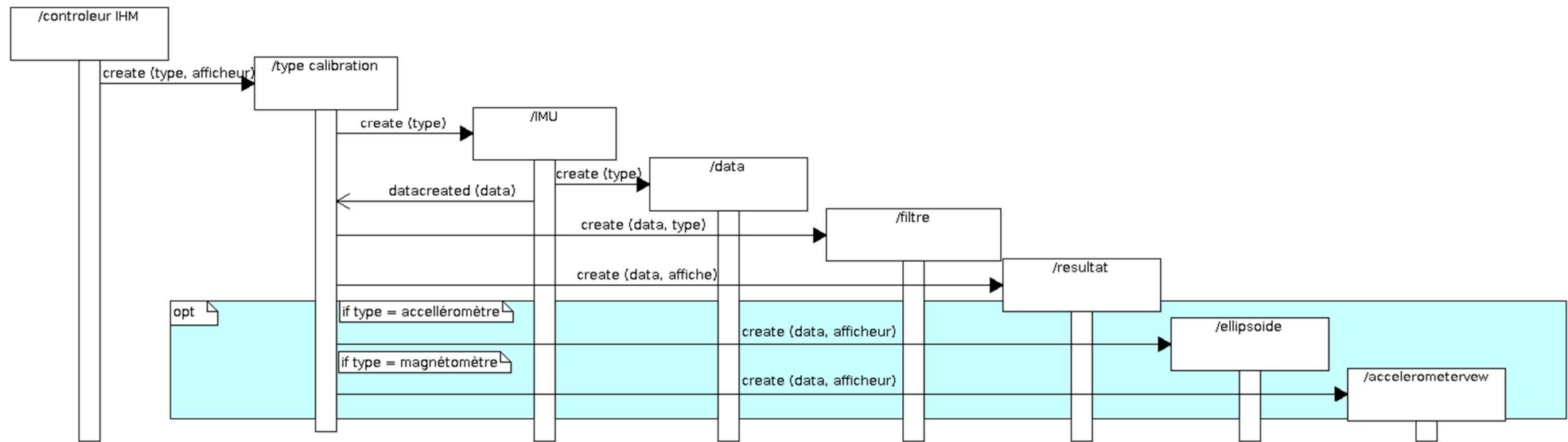


Figure 20 : Diagramme séquence de démarrage de la calibration

Le diagramme classe s'en trouve modifié de la façon suivante :

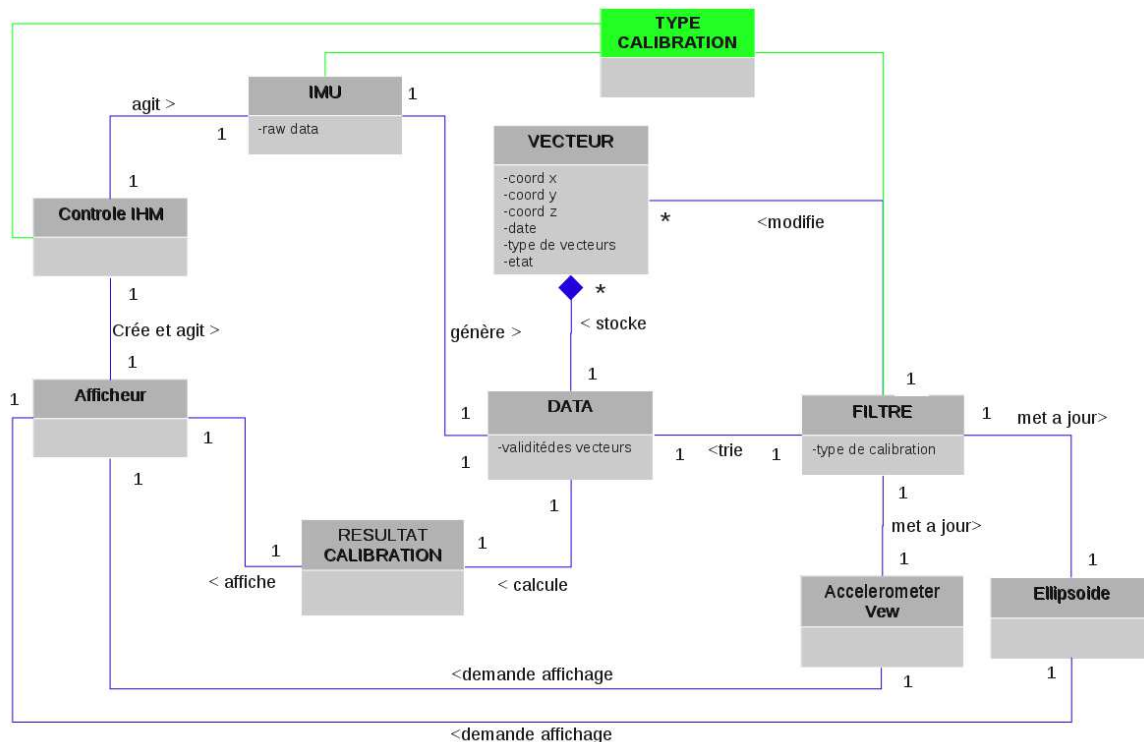


Figure 21 : Diagramme classe de démarrage de la calibration

Il ne reste plus qu'à gérer l'arrêt de l'application quand l'utilisateur veut quitter l'application. Sur action de l'utilisateur, on ramène l'affichage à l'écran de choix de calibration où il pourra choisir de quitter l'application.

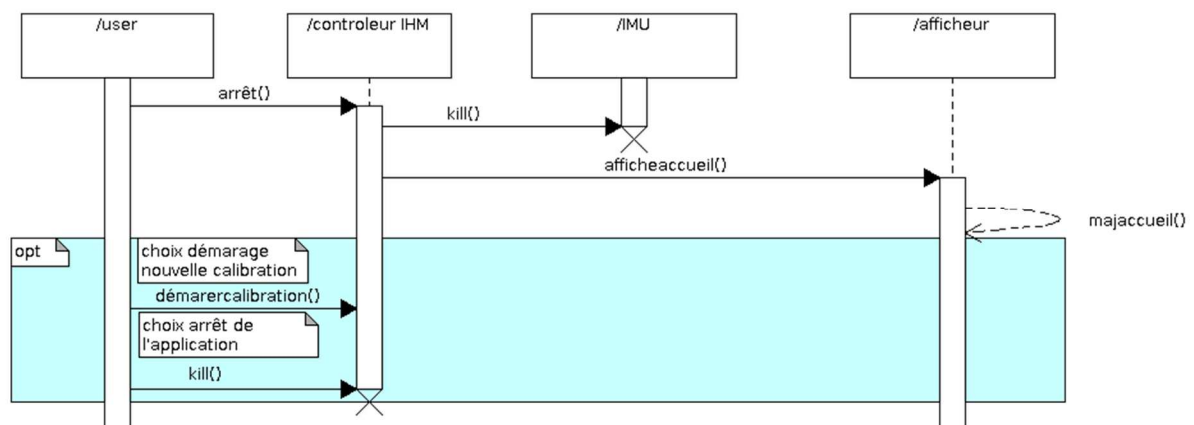


Figure 22 : Diagramme séquence d'arrêt

A nouveau, en détruisant IMU, on provoque une destruction en cascade des autres classes du domaine hors CONTROLEUR-IHM et AFFICHEUR.

Aucune modification n'est à apporter au diagramme précédent par rapport à cette nouvelle opération ; on obtient donc le diagramme suivant en fin de conception :

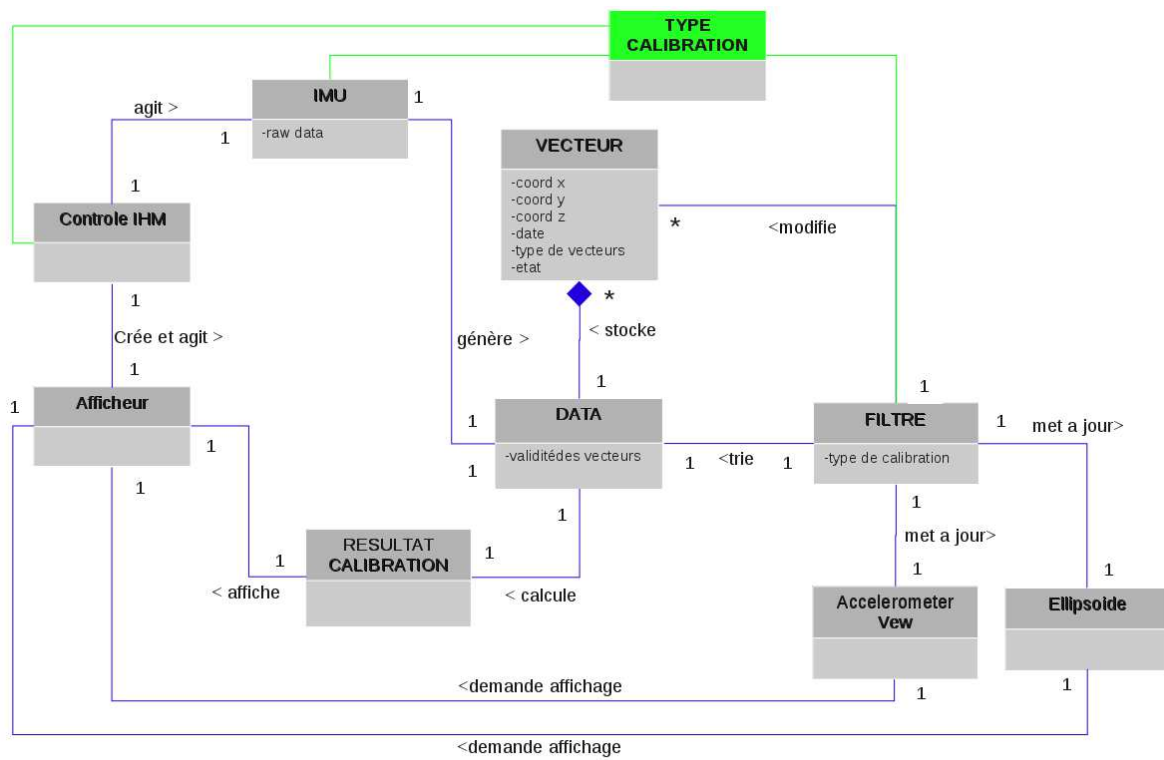


Figure 23 : Diagramme de conception final

III. Choix techniques issus de cette description

1. Choix de conception d'interface graphique

Pour les choix de conception d'interface graphique, on l'a vu quatre classes sont importantes, CONTROLEUR_IHM, AFFICHEUR, ELLIPSOIDE et ACCELEROMETER_VIEW. Si les deux premières sont assez génériques puisque communes à quasiment toutes les applications exploitant une interface graphique, les secondes nécessitent des développements spécifiques. Il convient que leur conception soit très clairement adaptée du tableau final du dossier des besoins qui détaille les différents points requis pour ces interfaces.

Concernant la démarche adoptée, elle reste similaire à celle proposée dans le dossier des besoins. Un prototypage papier permet d'écrêter les idées, un prototypage logiciel probablement en Java avec la librairie graphique finale afin de la prendre en main, permet de raffiner ces idées avant de ne sélectionner que la meilleure et de l'adapter à l'architecture spécifique du logiciel. Il convient donc de déterminer quelques choix de librairies potentielles qui peuvent permettre une mise en place de ces interfaces

Il nous faut choisir une librairie graphique pour la base de notre interface (les widgets ...) et là beaucoup de choix s'offrent à nous. Pour pouvoir faire un choix constructif nous allons faire un récapitulatif des avantages et inconvénients de trois librairies qui ont retenu notre attention : SWT, SWING, QtJambi.

	SWT	SWING	QtJambi
Portabilité	-	++	+
Rapidité	++	-	-
Consommation en ressources machine	+	-	-
Esthétisme	+	-	+
Simplicité	-	++	+

La rapidité et la consommation en ressources machine sont des éléments primordiaux pour la conception de notre interface, c'est pourquoi notre premier choix s'est porté sur la librairie graphique SWT.

S'en sont suivis de nombreuses difficultés d'implémentation du à notre méconnaissance de cette API. Suit à une consultation de Mr Jestin, nous nous sommes réorientés sur Swing qui présente de plus grandes facilités.

Viennent ensuite la réalisation de l'interface graphique. On a identifié dans la description de l'application 4 écrans : l'accueil, la phase de calibration pour les magnétomètres, la phase de calibration pour les accéléromètres, la phase d'affichage des résultats. Si les phases de calibration ont été fixées par le prototypage papier, reste l'accueil et le résultat.

Pour l'accueil, on voudra simplifier la tâche à l'utilisateur. On lui propose donc de réaliser toutes les étapes préliminaires à la calibration dans cette fenêtre et de lancer la suite des étapes. La connexion avec le drone et la GCS nous permet de récupérer beaucoup d'information à partir de l'identifiant du drone. De plus il nous faut permettre à l'utilisateur de sélectionner le bon mode pour la calibration (Raw_sensors le plus souvent mais cela peut varier). A cette fin, un écran d'accueil en trois temps à été conçu : d'abord la saisie de l'identifiant puis la sélection du mode (les différents modes possibles

étant récupérés sur le bus) et enfin la sélection du mode de calibration souhaité une fois que des données de type Raw, données nécessaire à la calibration, ont été détectées sur le bus.

On obtient donc les étapes graphiques suivantes :

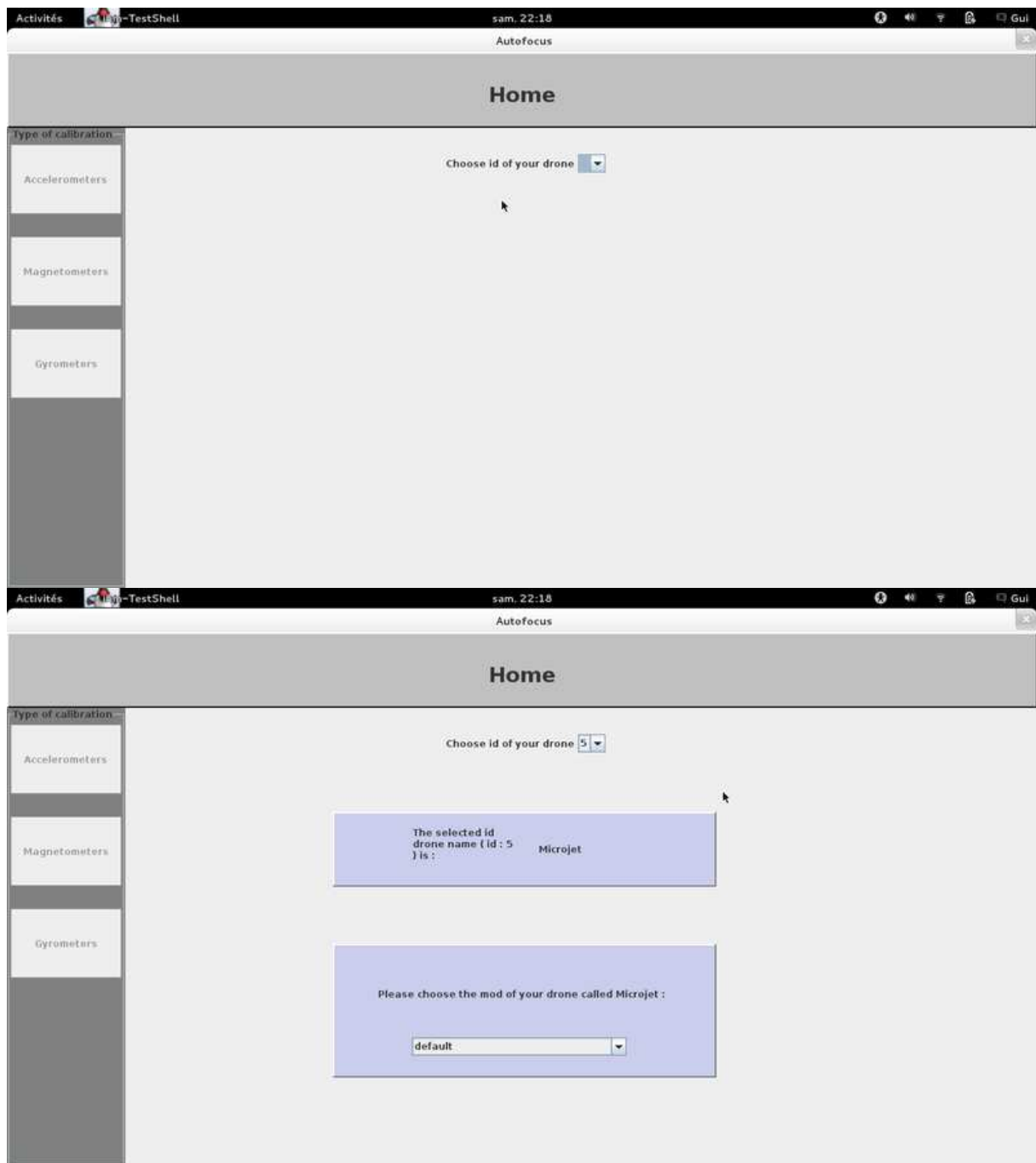


Figure 24 : Ecran d'accueil

Pour l'écran d'affichage des résultats, on choisit une simple zone qui permette un copier-coller des résultats que l'on affichera. En cas de calcul prolongé elle affichera une indication. Après test sur l'utilisateur, ces deux solutions semblent convenir.

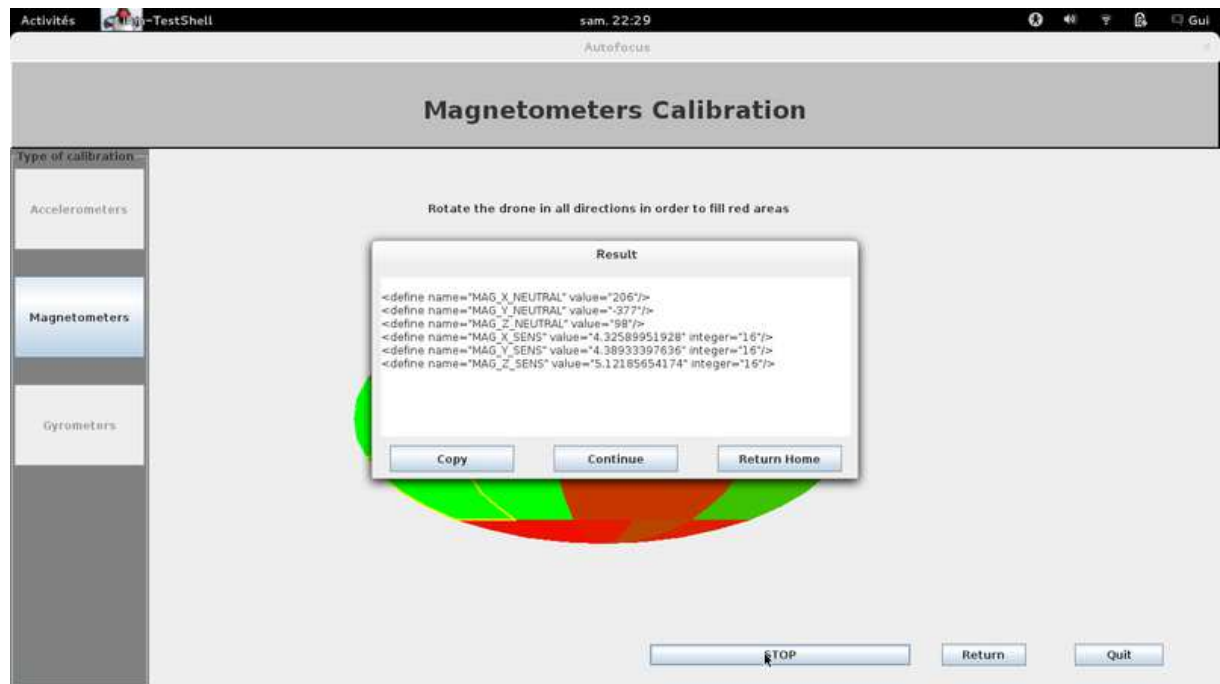


Figure 25 : Ecran de resultat

Une première implémentation de l'écran d'accueil, moins « intelligente » car ne détectant pas les noms et les modes d'émission de données automatiquement a été testée en prototypage. Après retour de l'utilisateur, elle a été revue pour aboutir au design de conception actuel.

2. Choix de conception algorithmique

Concernant les choix algorithmique, il apparaît par rapport au plan de projet qu'une certaine partie des algorithmes va devoir être recodée. En effet, la partie filtrage ne peut plus s'appuyer sur l'ensemble des valeurs comme elle le fait actuellement. Si l'algorithme de calibration peut-être conservé, il va falloir trouver un système permettant d'exploiter le code déjà écrit dans un autre langage

Pour la calibration finale des mesures, nous allons regarder les possibilités d'utiliser le script de calibration existant. Le logiciel existant est un script en langage Python qui prend toutes les mesures dans un fichier log, les filtre, calcule les paramètres de calibration, et génère une des vues des mesures. Notre interface devra aussi calculer les paramètres de calibration ainsi que générer des vues des mesures donc nous avons regardé si nous pouvions utiliser le script Python ou une partie du script pour notre interface. Il existe cependant quelques différences:

	Script de calibration existant	Interface du projet
Langage	Python	Java
Taches	Filtrage, Calibration et vue effectué après la prise de mesure	Filtrage et vue effectué en temps réel et la calibration après la mesure.
Données	Données utilisées pour le filtrage sont issues d'un fichier log	Données utilisées pour la vue et le filtrage sont acquises en temps réel. Calibration?

Nous avons donc trois options pour utiliser le script Python dans notre logiciel de calibration.

1. Recoder le script (ou une partie du script) Python en langage Java compatible avec le reste du logiciel.
2. Utiliser une librairie qui permet d'effectuer un binding java – Python (Jython) qui est capable de lire le script Python à partir de classe Java.

Pour déterminer quelle méthode on utilisera nous avons pesé les avantages et les inconvénients des différentes méthodes dans un tableau.

	1	2
Charge de Travail (codage)	--	+
Charge cpu (rapidité)	++	-
Intégration au system (facilité à intégrer avec les classes existante)	++	--
Utilisation client (téléchargement d'une librairie ou d'un programme de plus)	++	-
Maintenance, (complexité du programme)	+	-

Le filtrage et la vue doivent être effectués en temps réel et c'est un processus interactif, le facteur le plus important est donc la rapidité du logiciel. Il est donc judicieux de recoder cette partie en langage Java, même si cela engendre plus de travail pour nous.

La calibration après la prise de mesure n'a pas de contrainte de rapidité. Nous voulons donc privilégier l'intégration et la maintenance de cette partie du logiciel, ainsi que la facilité d'usage pour

le client.

En fonction des avantages et des inconvénients identifiés ci-dessus, nous avons décidé de partir sur un prototypage d'algorithme codé en java. Après plusieurs tests, nous avons décidé de faire un appel système car la tâche nous semblait trop complexe. On créera donc un fichier de log sur lequel on exécutera ensuite le script python.

On cherchera aussi au vu des besoins à rendre la classe **FILTRE** aussi générale que possible de façon à pouvoir la réutiliser pour les deux types de filtrage. Après des recherches concernant une librairie spécifique un compromis a été adopté consistant à réaliser un filtrage variable par variable en utilisant une classe déjà créée.

3. Algorithmique de l'affichage

Ainsi deux chaînes d'affichage ont été momentanément maintenues en parallèle afin de juger celle qui donne les meilleurs résultats une avec rafraîchissement à chaque ajout de point et modification du fond par groupe de points, le nombre de points nécessaires restant à définir et une autre où ces deux opérations s'effectuent en simultané. Les bons résultats de l'API graphique permettent de mettre en place le rafraîchissement point par point. Le lien rouge ci-dessous est donc maintenu pour sa mise en place.

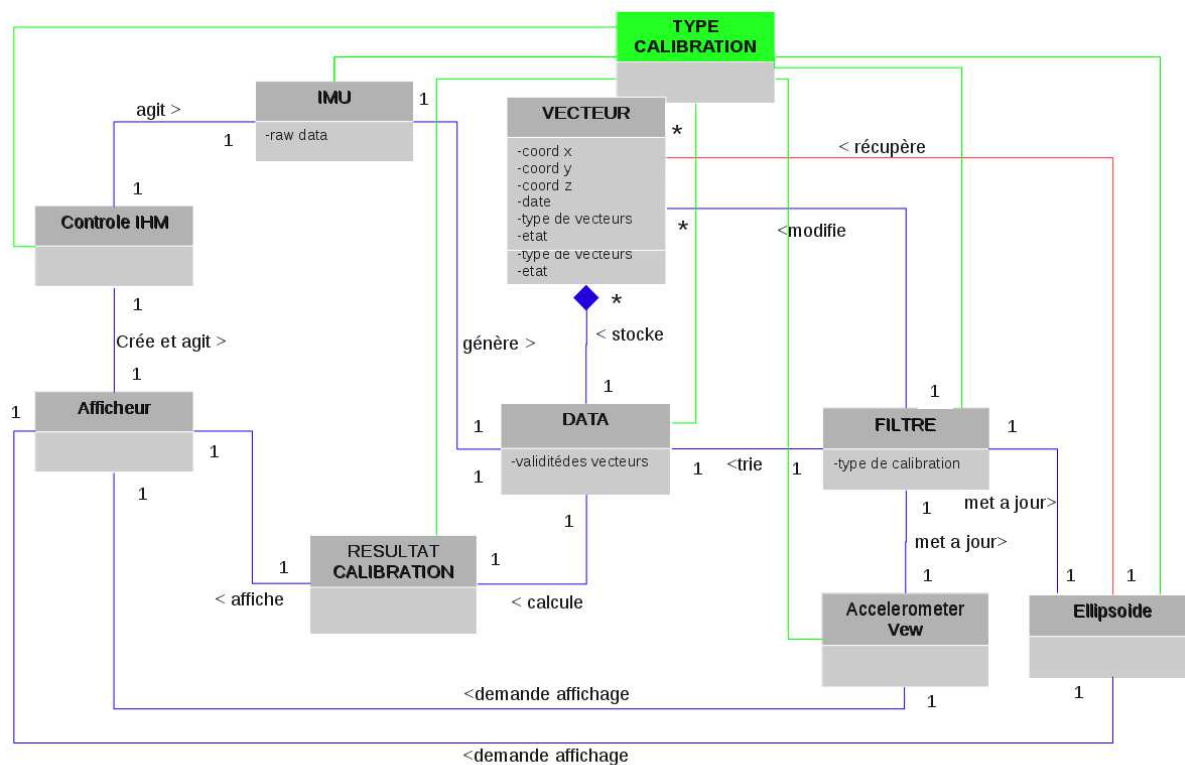


Figure 26 : Diagramme de classe choix de chaîne d'affichage

4. Matrice conception-besoins

Cette matrice vise à montrer quelle classe est responsable de la réponse à un besoin. Pour cela, on s'appuie sur le tableau des besoins défini dans le dossier des besoins. On renseigne alors la dernière colonne qui contient la classe associée au besoin. Cela est rendu possible par la façon dont ont été déterminées les classes. Les use cases faisant apparaître les réponses aux besoins, il est normal que les classes déduites des use cases apportent une réponse aux besoins. On obtient donc la matrice suivante.

Identifiant	Besoin	Classe réponse
B1	Fonctions et scenarios opérationnelles	
B1.1	Le système permet la réalisation du use case : « calibration magnétométrique »	Totalité du système
B1.2	Le système permet la réalisation du use case : « calibration des accéléromètres »	Totalité du système
B1.3	Le système permet la réalisation du use case : « calibration des gyromètres »	Totalité du système
B1.4	Le système permet de choisir entre différents types de calibration	Afficheur et Contrôle
B1.5	Le système permet de générer des résultats de calibration	Resultat_calibration
B1.6	Le système permet d'enregistrer les données collectées pendant la calibration	Data
B2	Performances attendues	
B2.1	Le système a une interface graphique permettant de guider l'utilisateur	Ellipsoide Accelerometer_view et
B2.2	Le système a une interface graphique permettant de juger de la qualité des résultats	Ellipsoide Accelerometer_view et
B2.3	Le système permet l'arrêt et la reprise d'une calibration	Contrôle_IHM
B2.4	Le système permet de générer des résultats à partir de résultats partiels de la calibration magnétométrique et des accéléromètres	Résultat_calibration
B2.5	Le système permet de choisir entre une calibration des accéléromètres ou des magnétomètres	Contrôle_IHM et Afficheur
B2.6	Le système est indépendant des autres éléments de PAPARAZZI et du système d'exploitation	Totalité du système
B2.7	Le système doit permettre une collecte facile des résultats	Résultat_calibration
B2.8	Le système doit indiquer l'ensemble des manipulations à effectuer	Afficheur
B3	Interfaces	

B3.1	Le système doit proposer une interface conviviale	Afficheur
B3.2	Le système donne un feedback temps réel (pas de temps de latence visible sous charge normale du système global) à l'utilisateur	Ellipsoide et Accelerometer_View
B4	Contraintes	
B4.1	Le système fonctionne de manière nominale sous un environnement et une charge de travail normale et une configuration de travail classique	Totalité du système
B4.2	Le système garantit que la qualité des résultats sera au moins la même qu'avec le système précédent	Résultat Calibration et filtre
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	Totalité du système
B4.4	Le système doit informer en temps réels l'utilisateur de son état	Afficheur

IV. Choix techniques issus de l'implémentation.

Peu de choix de conception ont été modifiés par l'implémentation des tests. Cependant, il s'avère que certaines modifications ont été nécessaires.

1. Réalisation de l'IHM

Le paradigme IHM utilisé ne permet pas une séparation propre du contrôleur et de la vue, ainsi les classes afficheur et contrôleIHM ont été fusionnées sous la classe shellV2.

2. Réalisation de Accelerometer_view et Ellipsoide

La réalisation de ces deux vues impose une dépendance entre les classes. De plus les données prises en entrée n'étant pas les mêmes, la classe filtre a dû être spécialisée pour prendre ces modifications en compte. La totalité de ces modifications peut être consulté sur diagClass.gif qui représente la globalité des classes.

V. Annexes

1. Cas d'utilisation

Comme on peut le voir dans les scénarios de travail, les use cases du système sont peu nombreux. On en dénombre 3. La calibration magnétométrique, celle des accéléromètres et enfin celle des gyromètres. Puisque ce dernier cas est proposé uniquement en bonus, nous ne le traiterons qu'une fois le dossier terminé.

Chaque procédure de calibration se découpe en 4 phases :

- **La phase de démarrage**

Cette phase consiste à lancer l'appliquet puis à choisir un type de calibration soit magnétométrique, soit accélérométrique.

- **La prise de mesures**

Cette phase consiste à récupérer des données qui vont être utilisées pour la calibration. Pour avoir une bonne calibration, ces données obtenues doivent couvrir une grande variété de positions. Le logiciel aidera l'utilisateur en montrant dans quel sens tourner le drone pour couvrir toutes les positions.

- **La calibration**

Quand la prise de mesure est terminée l'utilisateur peut décider de refaire une prise de mesure, ou de procéder à la calibration des données obtenues. La calibration renvoie un fichier XML contenant les paramètres de calibration.

- **La déconnection**

L'utilisateur peut décider de refaire une calibration ou d'arrêter de travailler avec le logiciel.

Etant donné le nombre limité de use cases, il convient de les séparer en plus petites entités afin de descendre en granularité et de pouvoir comprendre correctement les implications de chacun. Pour cela on les décompose en opérations selon le schéma suivant :

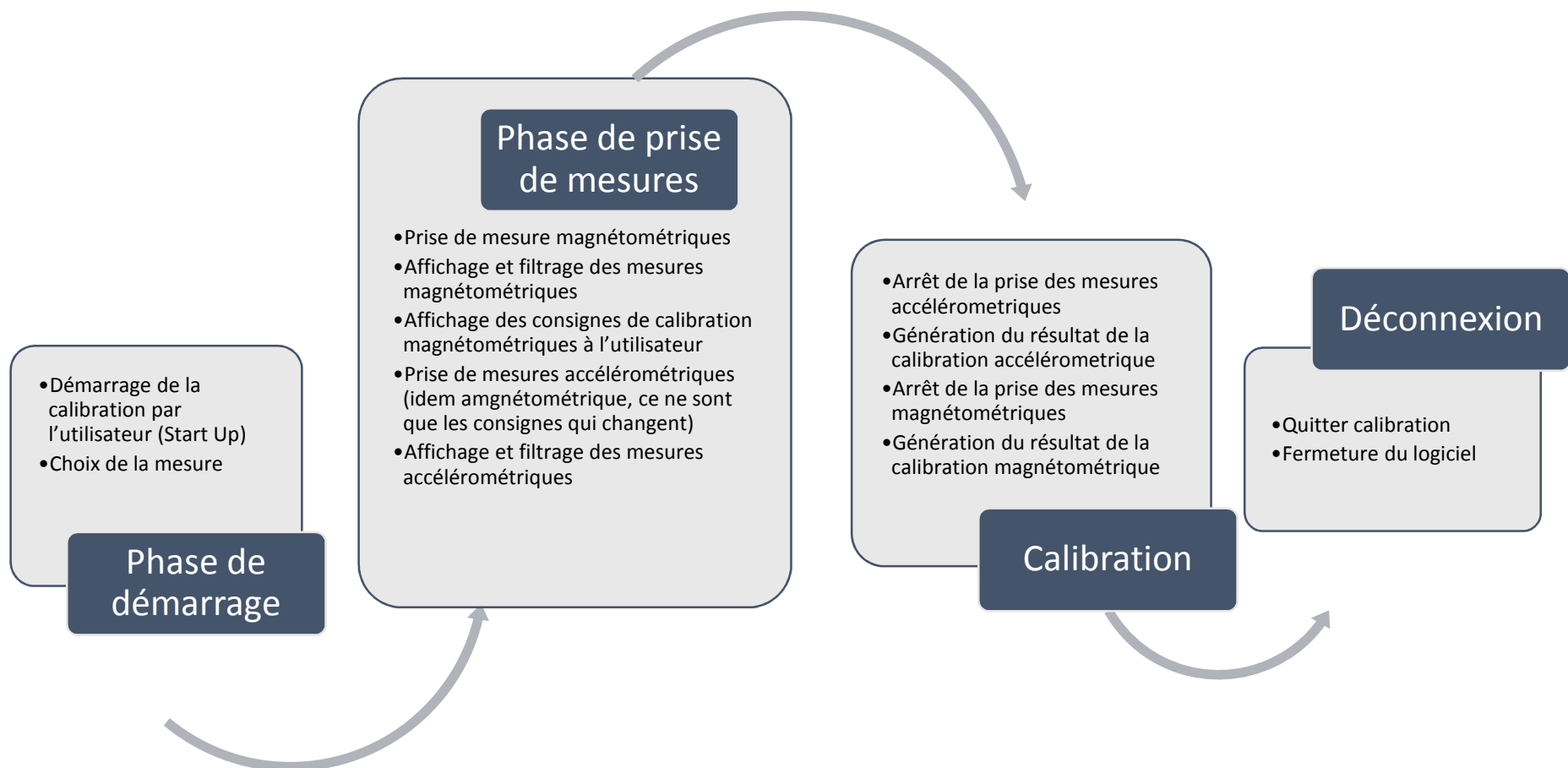


Figure 27 : Décomposition des use cases en phase

a. Phase de démarrage

Use case	Phase de démarrage
Description	Le démarrage du logiciel et l'ensemble des étapes jusqu'à la prise de mesure
Acteurs	L'utilisateur, le logiciel, le drone/IMU
Assomptions	Une connexion du drone avec le bus Ivy est nécessaire. Des accéléromètres et magnétomètres compatibles. Un ordi pour le logiciel. Un plan de travail stable pour effectuer les mesures.
Etapes	<ol style="list-style-type: none">1. L'utilisateur démarre le logiciel2. Le logiciel vérifie la présence du drone3. L'utilisateur sélectionne l'option « calibration de l'accéléromètre » dans la fenêtre du logiciel4. Le système affiche une vue et un indicateur de progression de la mesure en cours ainsi que de la calibration totale

b. Use case prise de mesure

Use Case	Prise de mesures magnétométriques
Description	L'utilisateur est guidé par le logiciel durant la phase de prise de mesure magnétométrique jusqu'à la phase de calibration
Acteurs	Le client, le logiciel, le drone/IMU
Assomptions	Une connexion du drone avec le bus Ivy est nécessaire. Un magnétomètre compatible. Un ordinateur pour le logiciel. Le use-case démarrage du logiciel a été effectué.
Etapes	<ol style="list-style-type: none"> 1. Le système enregistre les mesures magnétométriques 2. Le système affiche les consignes de calibration à l'utilisateur 3. L'utilisateur manipule le drone en accord avec les consignes. Dans le cas des magnétomètres cela consiste à faire tourner le drone autour de son centre de gravité de manière lente et continue 4. Le système filtre et affiche les mesures magnétométriques en temps réel sur une fenêtre graphique 5. L'utilisateur arrête les mesures en choisissant l'option dans la fenêtre du programme

Use Case	Prise de mesures
Description	L L'utilisateur est guidé par le logiciel durant la phase de prise de mesure accélérométrique jusqu'à la phase de calibration
Acteurs	Le client, le logiciel, le drone/IMU
Assomptions	Une connexion du drone avec le bus Ivy est nécessaire. Un magnétomètre compatible. Un ordinateur pour le logiciel. Le use-case démarrage du logiciel a été effectué.
Etapes	<ol style="list-style-type: none"> 1. Le système enregistre les mesures magnétométriques 2. Le système affiche les consignes de calibration à l'utilisateur 3. L'utilisateur manipule le drone en accord avec les consignes. Dans le cas des accéléromètres cela consiste à maintenir le drone plusieurs secondes dans une position stable et à réitérer cette étape sur un minimum de six positions 4. Le système filtre et affiche les mesures magnétométriques en temps réel sur une fenêtre graphique 5. L'utilisateur arrête les mesures en choisissant l'option dans la fenêtre du programme

c. Use case calibration

Use Case	Prise de mesures
Description	L'affichage des résultats de la calibration à l'utilisateur et le guidage dans leur utilisation.
Acteurs	Le client, le logiciel, le drone/IMU
Assomptions	Une connexion du drone avec le bus Ivy est nécessaire. Un magnétomètre compatible. Un ordinateur pour le logiciel. La phase de mesure dans un des modes accéléromètres ou magnétomètres a déjà été effectué.
Etapas	<ol style="list-style-type: none"> 1. Le système enregistre les mesures magnétométriques 2. Le système affiche les consignes de calibration à l'utilisateur 3. L'utilisateur manipule le drone en accord avec les consignes. Dans le cas des accéléromètres cela consiste à maintenir le drone plusieurs secondes dans une position stable et à réitérer cette étape sur un minimum de six positions 4. Le système filtre et affiche les mesures magnétométriques en temps réel sur une fenêtre graphique 5. L'utilisateur arrête les mesures en choisissant l'option dans la fenêtre du programme

d. Use case fermeture du logiciel

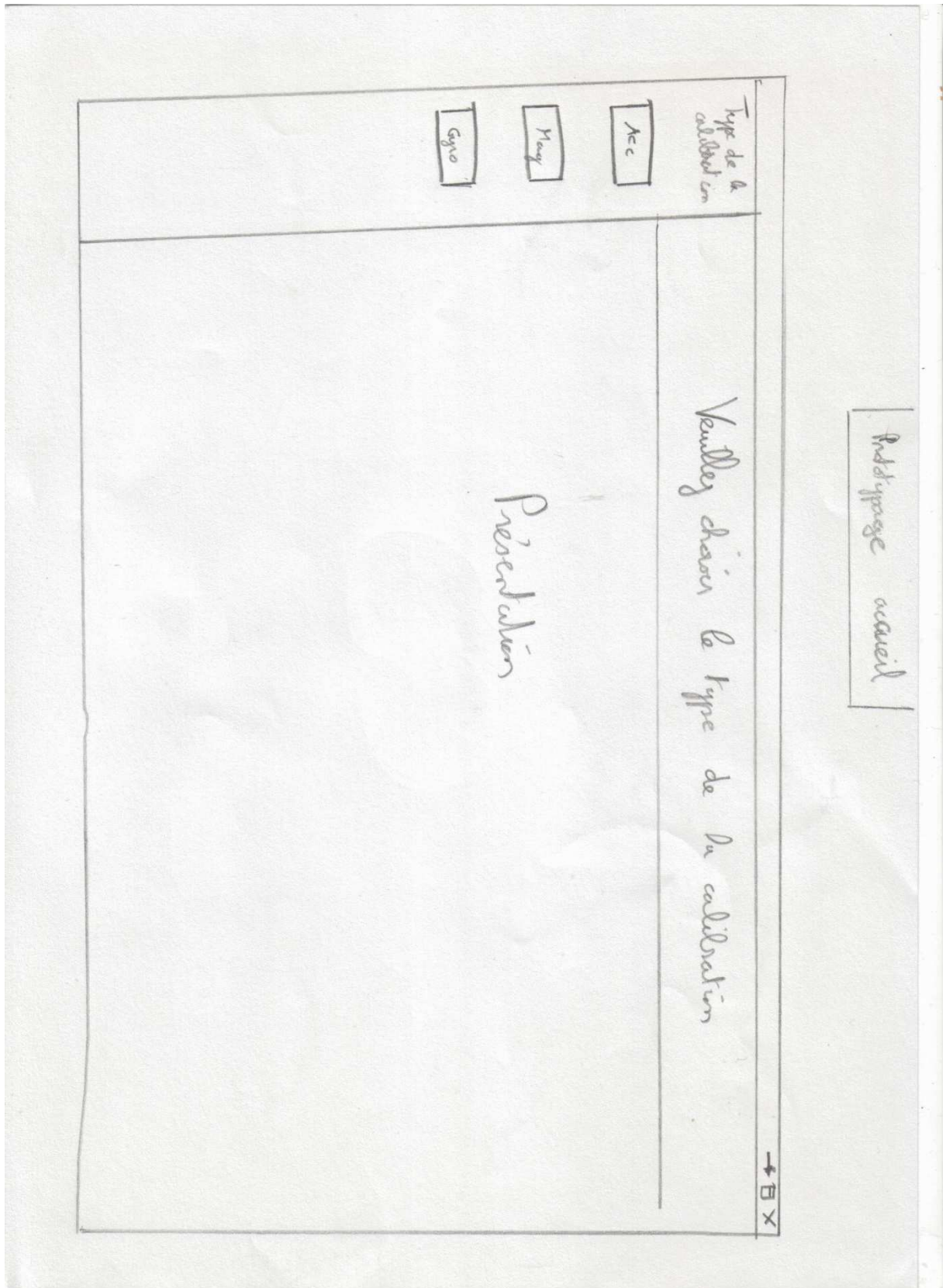
Use Case	Prise de mesures
Description	Fermeture du logiciel
Acteurs	Le client, le logiciel, le drone/IMU
Assomptions	Une connexion du drone avec le bus Ivy est nécessaire. Un magnétomètre compatible. Un ordinateur pour le logiciel. La phase de calibration a été effectué, l'utilisateur vient de collecter les résultats, il veut quitter le logiciel
Etapes	<ol style="list-style-type: none">1. L'utilisateur quitte la fenêtre de résultat2. Il peut choisir de continuer la calibration ou de quitter la calibration en cours3. S'il décide de quitter la calibration en cours il peut alors sélectionner un autre drone4. Sinon il clique sur la croix

e. Conclusion

Les use cases sont, comme on l'a vu assez similaires, seule change l'interface associée à chaque prise de mesure. Pour se donner une idée des différences, on peut trouver en Annexe 3 les propositions d'IHM.

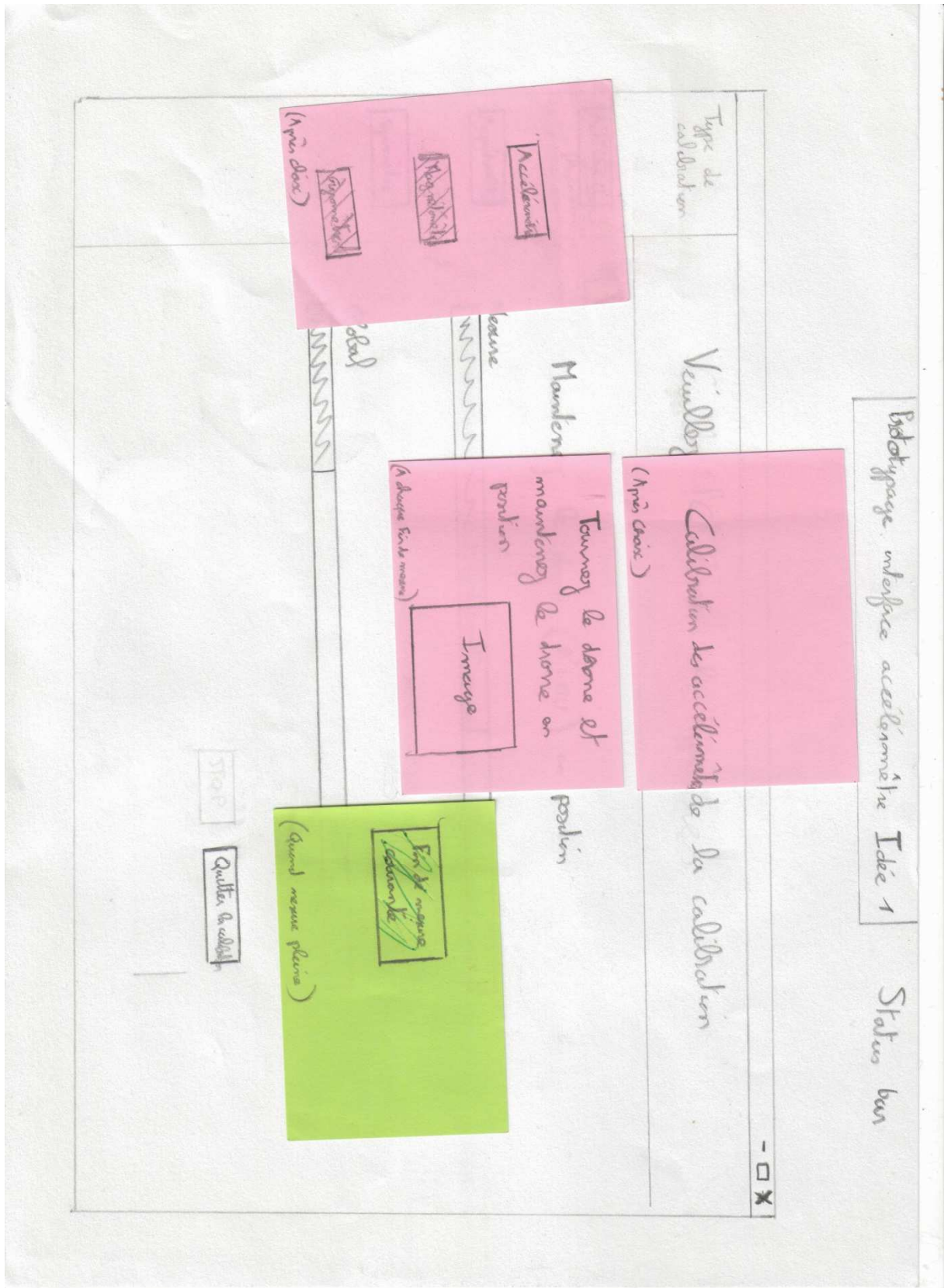
2. Prototypage papier d'interface

a. Prototypage accueil



b. Prototypage interface accéléromètre

Status Bar



Prototypage interface magnétométique Idée 8

Remarque

Veulley (Ap. l'air)

Calibration des magnétomètres de la calibration

Typologie

Me

Mag

gto


(Ap. l'air)

Tourner le drapeau d'orientation dans les deux sens


des différents puits

STOP

Quit



Magnétomètre de l'orientation des puits dans les deux sens de l'orientation des puits, la zone de puits "bien" placée dans la zone issue en fait des puits opposés

Prototypage interface magnéto-linéaire idée 3		Planographie V2	
Type de			
	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <div style="border: 1px solid black; padding: 2px; display: inline-block;">Nec</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">Mag</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">Type</div> </div>		
	<p>Veuillez</p> <p>Tourner la tête autour dans tous les sens pour remplir les différents zones</p>	<div style="border: 1px solid black; padding: 10px; text-align: center;"> <p>Calibration des magnéto-linéaires</p> <p>(No droit)</p> </div>	<p>La calibration</p>
	<div style="border: 1px solid black; padding: 10px; text-align: center;">  <p>(No 441 mod)</p> </div>	<div style="border: 1px solid black; padding: 10px;"> <p>Item 2 mais la parastrophie est liée, la transformation de l'espace se faisant en regardant les bords vers</p> <p>(Expliquer)</p> </div>	<p>- □ x</p>
	<div style="border: 1px solid black; padding: 5px; display: inline-block;">STOP</div> <div style="border: 1px solid black; padding: 5px; display: inline-block;">Quel</div>		

c. Prototypage interface magnétomètre

3D Graphics

