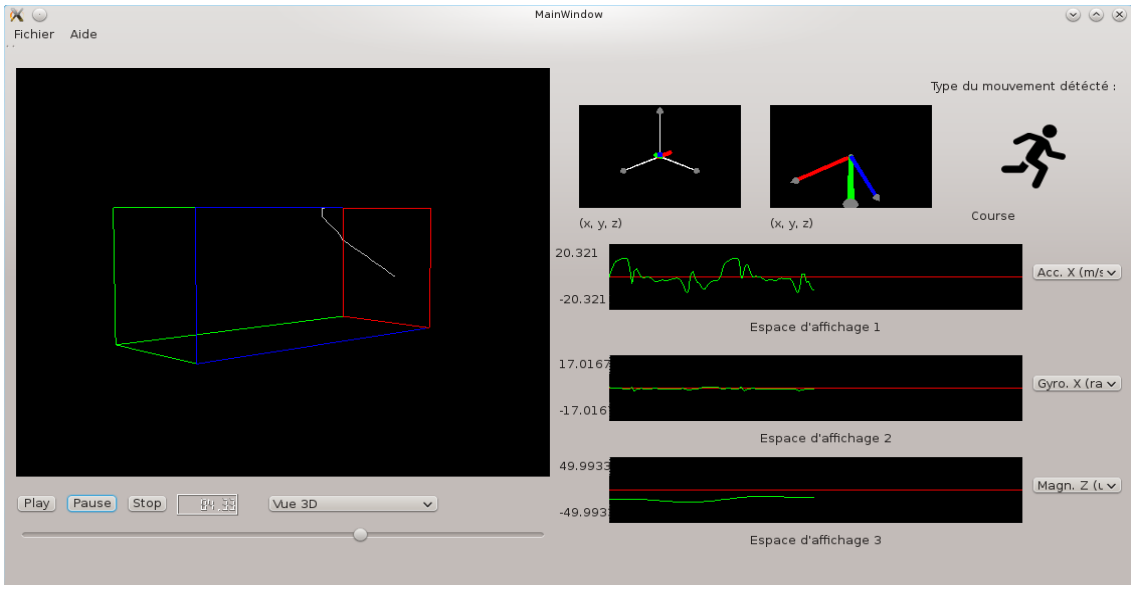
Ferdaousse Chakir – Mohammed Amjad Lasri - Régis Cuissard

Ecole Nationale des Sciences Géographiques
IT 3 – Filière GTSI
Projet OpenGL

E. Bardière

Développement d'un logiciel de visualisation et de qualification d'une centrale inertielle



Sommaire

I. Problématique

De nos jours, il y a de plus en plus d'intérêt pour la conception d'applications informatiques qui reposent sur la nature du mouvement humain. Ces logiciels exploitent souvent les données de centrales inertielles qui équipent désormais la plupart des appareils mobiles (téléphones, tablettes...) .
C'est dans ce cadre que se situe le présent travail. Son objectif est de développer un logiciel permettant d'étudier des données brutes acquises par une centrale inertielle placée sur un opérateur. Cette centrale fournie les données de trois capteurs: Accéloromètre, Gyromètre, Magnétomètre.
Le logiciel est composé d'une interface graphique permettant de visualiser les données et de classifier le mouvement.
Le logiciel a été réalisé en C++ via le logiciel QT et en s'appuyant sur la librairie graphique OpenGL.

II. Méthodologie

Pour atteindre l'objectif, le travail a été découpé en plusieurs tâches. En outre, les compétences acquises en cours de techniques de développement ont pu être mises en oeuvre en :

- Définissant le périmétre du projet, afin que les objectifs soient définis de manière unanime dans l'équipe
- Testant la méthode de projet Scrum via des sprints d'une semaine, suivi de reviews, ce qui a apporté plus de souplesse et de réactivité
- Travaillant avec le logiciel GIT et un dépôt GitHub, afin de faciliter le travail collaboratif

Le sujet étant relativement large, l'équipe a choisi de prendre les hypothèses suivantes :

– la centrale inertielle est fixée sur le pied d'un opérateur
– les différentes classes de mouvement sont : la marche, la course, la montée et la descente d'escaliers
– la visualisation et la classification des données sont réalisées en post-traitement et non en temps réèl

III. Conception

A) Les cas d'utilisation

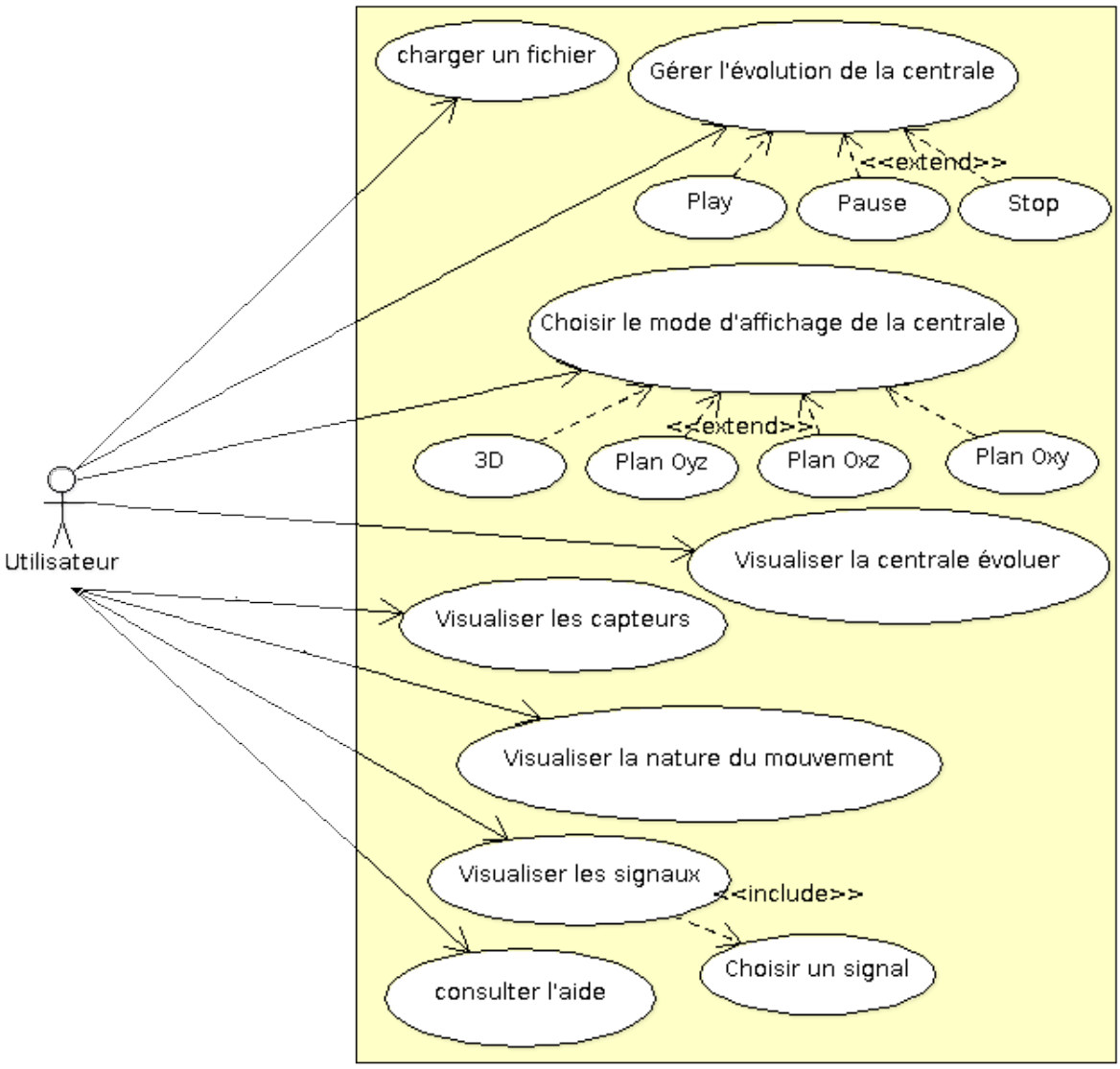


Illustration 1: Diagramme des cas d'utilisation

Le choix des fonctionnalités a été réalisé en essayant d'apporter une plus-value à un utilisateur qui souhaiterait qualifier une centrale inertielle ou d'étudier une possible classification du mouvement.

La fonction d'affichage des signaux lui permet ainsi de choisir la composante qu'il souhaite visualiser.

De même, il est possible de projeter la trajectoire de la centrale sur un plan afin de qualifier la dérive selon les trois axes (tests d'acquisition de parcours en boucle et vérification que point de départ = point d'arrivée par exemple).

B) La conception objet

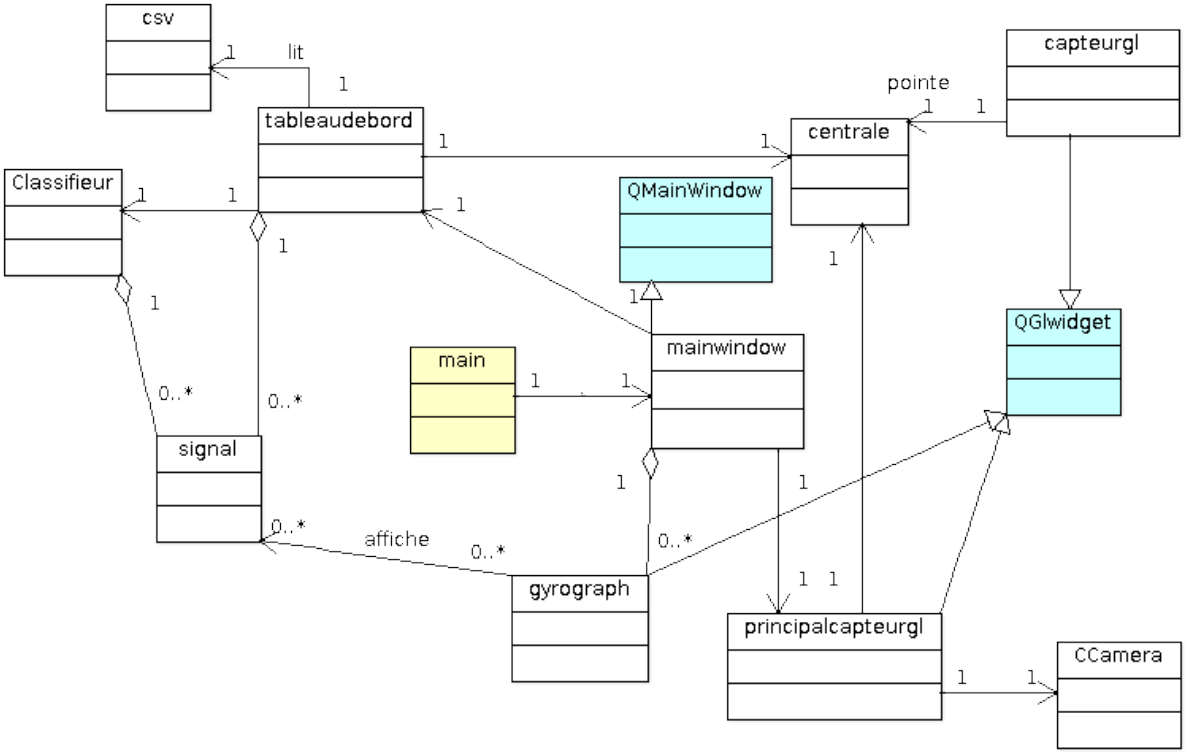


Illustration 2: diagramme de classes

Mainwindow représente la classe principale d'affichage.

TableauDeBord est la classe principale qui gère les traitements (métier). C'est par son intermédiaire que toutes les données sont créées, notamment les signaux. C'est également elle qui fera évoluer virtuellement la centrale dans le temps.

CSV est la classe de lecture du fichier de données.

Classifieur comme son nom l'indique, permet de classifier la nature du mouvement de l'opérateur.

Signal permet de représenter un signal, d'effectuer des traitements et des statistiques dessus.

Principalcapteurgl et capteurgl pointent sur la même centrale que TableauDeBord. Ils permettent respectivement de visualiser son évolution et celle de ses capteurs. PrincipalCapteurGL dispose d'une caméra qui permet de gérer un affichage plus avancé.

Gyrograph permet la visualisation d'un signal.

C) Fonctionnement global du logiciel

Ci-dessous est représenté le fonctionnement global du logiciel.

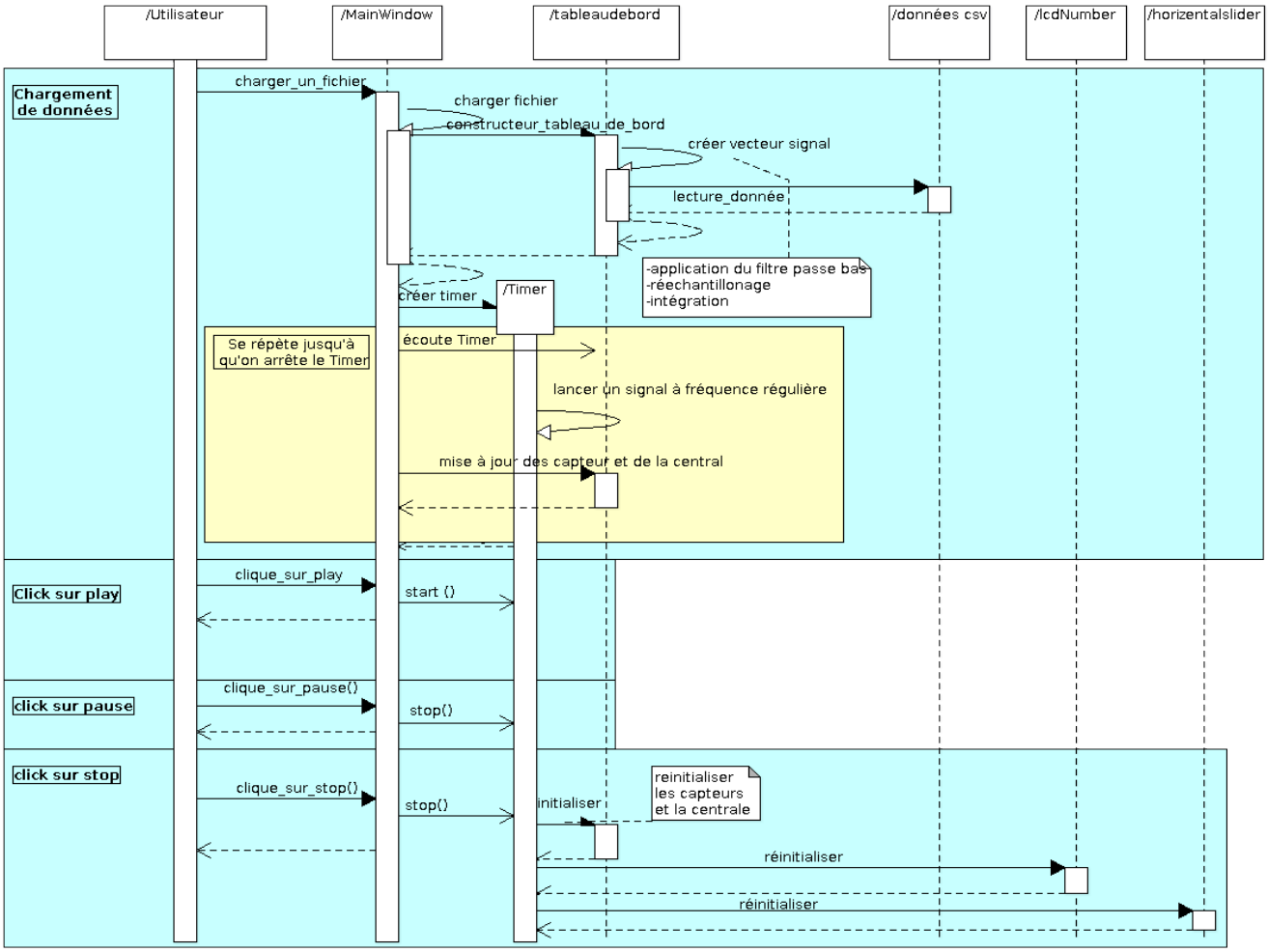


Illustration 3: Fonctionnement global du logiciel

Lorsque l'utilisateur choisit un fichier, le logiciel réalise les pré-traitements nécessaires pour l'affichage de la centrale et de la classification du mouvement de l'opérateur.

Une fois ces traitements réalisés, MainWindow crée un timer qui va permettre de lancer à une fréquence régulière :

- l'évolution de la centrale dans le temps
- la mise à jour de la fenêtre d'évolution de la centrale
- la mise à jour des fenêtres des capteurs
- la mise à jour des fenêtres de visualisation des signaux
- la mise à jour de l'affichage de la classe de mouvement
- la mise à jour de l'écran LCD (affichant le temps), barre de défilement

A noter que le rafraîchissement des fenêtres d'affichage n'apparait pas sur le diagramme, mais les

méthodes updateGL des classes d'affichage (capteurgl, principalcapteurgl et gyrograph) sont appelées selon le même principe que majCentrale.

Le tout tourne tant que l'utilisateur ne clique pas sur pause ou stop.

IV. Les étapes du traitement

A) Acquisition des données

L'acquisition des données a été réalisé à partir d'un téléphone Samsung Note 1 sous Android équipé de l'application IMU+GPS Stream.
L'application envoie les données en wifi par protocole UDP. Celles-ci sont récupérées via un programme réalisé en Python et permettant d'ouvrir le socket depuis un PC portable, dont le code source est ci-dessous.

```
# ------------------------------------------------------
import socket, traceback

# Insérer l'IP et le port sur lequel le téléphone envoie le flux UDP
host = ' 0.0.0.0 '
port = 5555

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
s.bind((host, port))

while 1:
try:
message, address = s.recvfrom(8192)
print message
except (KeyboardInterrupt, SystemExit):
raise
except:
traceback.print_exc()
# ------------------------------------------------------
```

Les données sont au format texte, séparateur virgule décomposées en 13 colonnes :

- une clonne présentant le temps en seconde (s)
- un code capteur
- trois colonnes présentant l'accéloromètre en (m/s-2)
- un code capteur
- trois colonnes présentant le gyroscope en (rad/s-1)
- un code capteur
- trois colonnes présentant le magnétomètre en (ts)

L'acquisition des données n'étant pas le cœur du projet, un protocole relativement simple a été privilégié. L'opérateur piéton insère le téléphone dans sa chaussette pour se déplacer. Plusieurs jeux de données ont été créés.

Tout d'abord, quatre jeux distincts ont été acquis pour deux opérateurs : marche, course, montée d'escaliers. Le but de ces jeux de données est l'identification d'indicateurs permettant la classification du mouvement.

Un autre jeu de données sur un troisième opérateur a été réalisé. Ce jeu est caractérisé par un enchaînement des quatres classes de mouvements citées ci-dessus.

Ci-dessous sont représentés les trois axes de la centrale sur le téléphone.

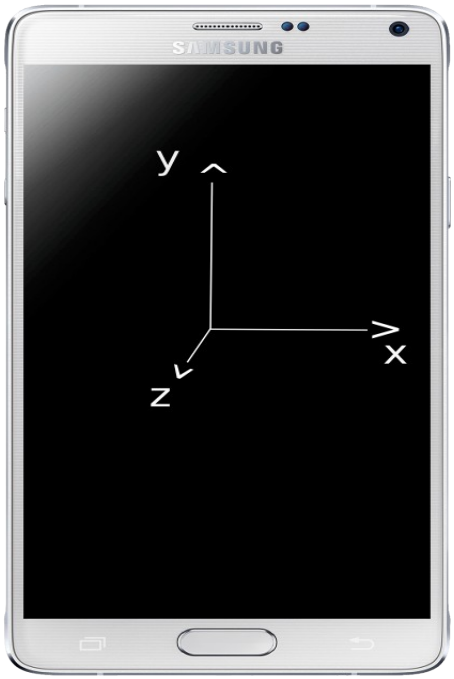


Illustration 4: Les trois axes de la centrale du téléphone

B) Filtrage des données

Une première visualisation sous Octave a permis d'observer que les signaux issus de la centrale étaient bruités.

Pour effectuer une correction du signal, il fallait d'abord déterminer l'origine et la nature du bruit. Une recherche bibliographique a permis d'identifier :

- que la mesure de l'accélération donnée par le capteur comprenait également l'effet de la gravité
- qu'une dérive apparaissait au bout d'un certain temps sur le gyroscope si celui-ci ne faisait pas l'objet d'un mouvement brusque

Afin de corriger ces effets indésirables, il a été choisi d'appliquer un filtre passe-bas pour identifier la composante gravité et la soustraire au signal brut.

Etant donné le placement du gyroscope, il a été supposé que le mouvement serait suffisament important pour pouvoir annuler une éventuelle dérive.

Le filtre passe-bas a été réalisé dans le domaine spectral en utilisant une transfurmée de Fourier. Celle a été calculée en s'appuyant sur la librairie Aquila, présentant l'avantage d'être OpenSource et relativement légère.

C) Classification du mouvement

La recherche documentaire a permis d'identifier deux types de méthodes de classification de mouvements à partir de données inertielles :

– des méthodes basiques basées sur un parcours des signaux de centrale : identification de seuils, étude de la corrélation entre les signaux

– des méthodes plus robustes basées sur des méthodes comme la classification bayésienne ou les champs cachés de Markov

Au vu du délai, il a été choisi d'opter pour une méthode simple basé sur des statistiques locales. Celles-ci sont obtenues en faisant glisser une fenêtre sur les signal.

Le diagramme ci-dessous décrit le processus de classification.

–

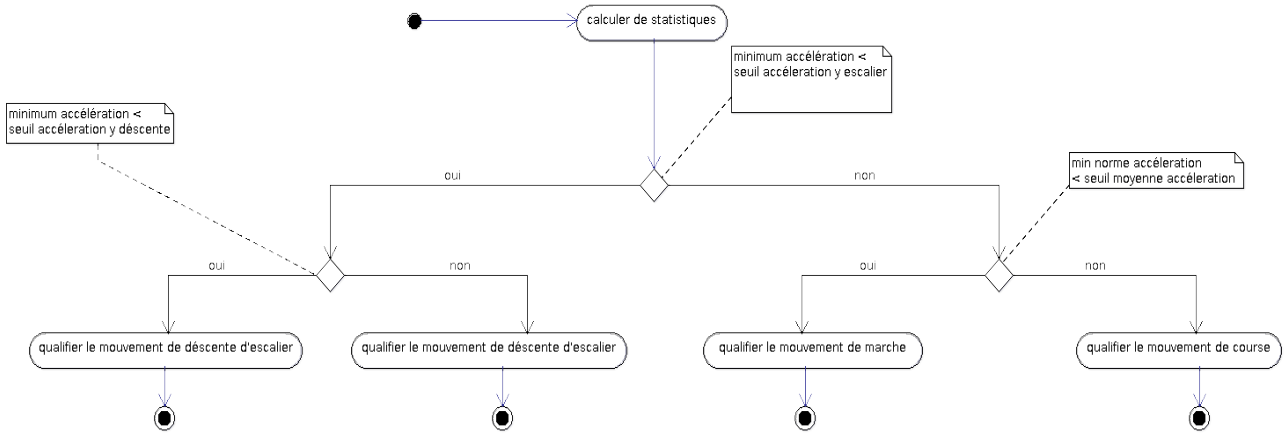


Illustration 5: Processus de classification du mouvement

Les seuils utilisés ont été déterminés de manière empirique à la suite d'une étude approfondie de différents indicateurs locaux : coefficient de corrélation, moyenne, minimum, maximum, écart-type, amplitude.

D) Intégration des données

Les données inertielles que le logiciel exploite doivent être intégrées afin de pouvoir visualiser l'évolution de la centrale. Les données de l'accéléromètre sont intégrées deux fois, celle du gyroscope une fois.
Les résultats issus d'un processus d'intégration d'un signal sont très sensibles au bruit (d'autant plus si le processus est répété).
Il a été choisi d'intégrer les signaux par calcul d'aires sucessif représenté par le signal à intégrer et l'axe des abscisses car c'est la méthode qui permet d'éviter toute approximation.

soit deltaX = unTemps[i]-unTemps[i-1]
y1 = unSignal[i-1]
y2 = unSignal[i]
yMinAbs : la plus petite valeur de la valeur absolue de y1 et y2,
deltaY = abs(y2-y1)
deltaIntegre : la valeur de l'intégral entre t et t+1.

Lors de l'intégration, trois cas sont distingués.

- Le cas ou les deux valeurs du signal sont positives :

Dans ce cas, calculer l'integrale revient à calculer la surface d'un triangle et d'un rectangle et faire la somme des deux de la façon suivante :

deltaIntegre= deltaY*deltaX/2.0+deltaX*yMinAbs

- le cas ou les deux valeurs du signal sont négatives :

La même formule avec un signe moins .

- le cas ou les deux valeur du signal sont de signe différent :

Le calcul se base sur Thalès pour déterminer les deux côtés a et b mentionnés sur la figure ci-dessous.

```
si y2 !=0  a=  deltaX*abs(y1/y2)
  sinon
            a=deltaX

si y1 !=0  b= deltaX*abs(y2/y1)

  sinon
            b= deltaX
```

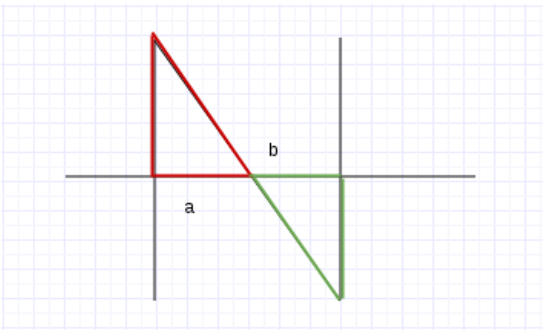


Illustration 6: Intégration du signal, cas 3

E) Changement de repère des données

Pour faire évoluer la centrale dans l'espace, un changement de repère est nécéssaire pour reprojeter les valeurs de la position (X, Y, Z) qui sont exprimées initialement dans le repère du téléphone dans le repère bâtiment.
Pour effectuer ce passage entre les deux repères, on applique la formule suivante:

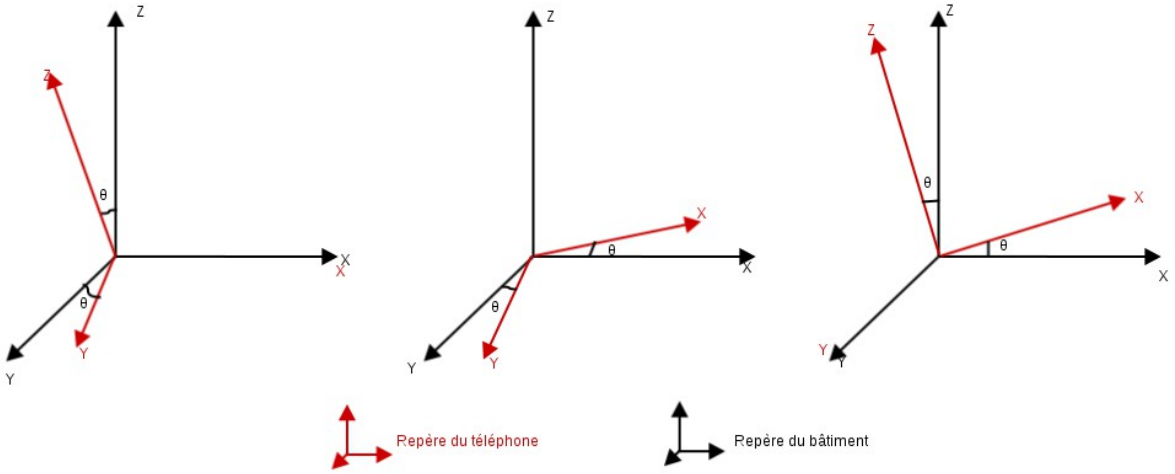


Illustration 7: Changement de repère téléphone/bâtiment

$$R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \quad R_Y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad R_Z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{passage} = R_X R_Y R_Z$$

$$V_{gyro} = M_{Passage} . V_{accelero}$$

On note ici, qu'une amélioration peut être portée à cette formule de passage en ajoutant la translation entre les deux capteurs. Généralement le centre des repères des deux capteurs ne sont jamais confondus. Pour quantifier cette translation il faudrait mesurer au préalable la distance entre les centres des deux capteurs, cette quantité varie d'une centrale à une autre.

F) Evolution de la centrale au fil du temps

L'évolution de la centrale au fil du temps est gérée grâce au design pattern Observateur implémenté sous Qt avec les signaux et les slots.

En écoutant un timer émettant à fréquence régulière, les méthodes suivantes sont appelées :

– tdb::majCentrale : fait évoluer la position de la centrale en parcourant le signal représentant sa position

– mise à jour des QWidgetGL permettant les différents affichages

Afin de parcourir le signal en respectant l'évolution du temps courant, la date courante est sauvegardée à chaque itération. Si d'une itération à l'autre, la différence de date est supérieure à la période d'échantillonage du signal alors on parcourt plus rapidement le signal.

G) Interface

L'interface a été créée à l'aide de Qt Designer.

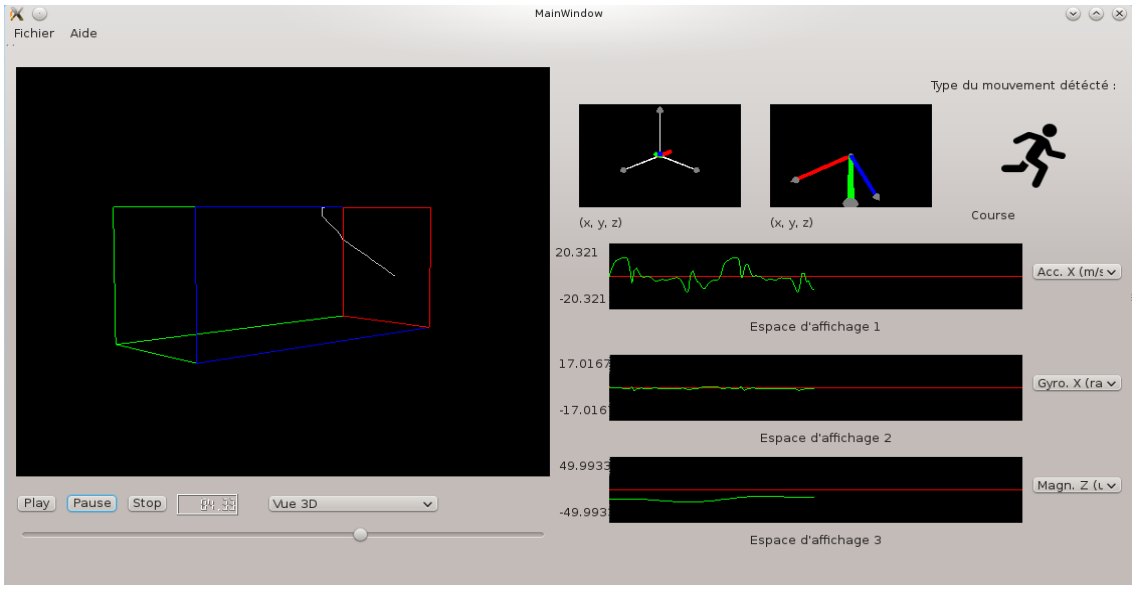


Illustration 8: capture d'écran de l'interface du logiciel

La partie de gauche permet de visualiser l'évolution de la centrale en 3D ou en mode projeté sur les trois plans (classe PrincipalCapteurGL). L'utilsiateur peut également activer le mode suivi en appuyant sur la touche 's' du clavier. La partie supérieure représente les données de l'accéléromètre et du gyroscope (classe CapteurGL). L'image en haut à droite représente le type courant du mouvement. Il est possible d'afficher trois signaux différents via les listes déroulantes (classe gyrograph).
Le fichier se lit avec les boutons classiques d'un lecteur multimédia.

V. Résultats et perspectives

Ce projet a été une réelle chance pour appliquer les connaissances acquises principalement dans l'élément "Méthodes et Techniques de développement". Ces connaissances simplifient considérablement la tâche du développeur, spécialement lorsque l'application commence à prendre des proportions volumineuses. Nous avons appliqué ces méthodes (Versioning avec Git, déboggage, tests unitaires, Documentation du code avec doxygen,...) en essayant de rendre notre

travail le plus simple à reprendre et à comprendre par les développeurs qui veulent y contribuer dans l'avenir.

Dans cette première version, la solution qu'on propose permet de traiter, exploiter et aussi visualiser plusieurs aspects d'une centrale inertielle. On propose alors, dans tout ce qui suit, des pistes d'améliorations possibles.

- **Améliorer le traitement du bruit de la centrale:**

Comme nous avons évoqué précedemment, pour traiter le bruit d'une centrale il faut éliminer l'effet de la gravité sur l'accéléromètre et la dérive du gyroscope. Nous avons implémenté le filtre passe bas pour isoler la gravité et ne garder que les accélérations linéaires qui nous intéressent. Un traitement de la dérive du gyroscope est alors à prévoir dans les versions à venir. La méthode la plus connue dans ce contexte est d'intégrer des contraintes extérieures avec les données de la centrale inertielle en appliquant un filtre de Kalman pour s'affranchir du bruit. On note ici que l'API Android SDK permet d'isoler la gravité du signal de l'accéléromètre à travers le capteur TYPE_GRAVITY de la classe Sensor.

- **Méthodes de classification du mouvement:**

Pour classifier le mouvement de l'opérateur nous nous sommes basés sur une méthode empirique. On note ici qu'il existe une documentation riche et des approches variées du sujet (empiriques, probabilistes, ...). On juge ainsi qu'une étude plus profonde mérite d'être menée pour tester les performances de chaque algorithme et d'implémenter le plus approprié.

- **Afficher les indicateurs statistiques:**

Les espaces d'affichage des signaux en fonctions du temps ont pour vocation de simplifier à l'utilisateur l'analyse visuelle des signaux. Ces espaces peuvent alors être utilisés pour afficher les indicateurs statistiques (variance, corrélation, ...). Ces indicateurs peuvent apporter une dimension en plus dans l'analyse du signal.

- **Utiliser les rendus optimisés DL, VBO, VA:**

Pour l'instant l'application gère les espaces d'affichage sans détérioration dans la qualité d'affichage. Néanmoins si une suite est donnée à ce projet et que d'autres espaces d'affichage sont ajoutés, la vitesse de raffraichissement de la fenêtre pourrait être affectée. C'est pour cela qu'on propose d'utiliser des méthodes de rendu 3D comme DL, VBO, ...

- **Proposer un mode temps réel:**

L'application dans sa première version supporte uniquement les traitements en temps différé (on enregistre le flux des données issus de la centrale inertielle dans un fichier, et l'application lit le fichier CSV, le parse et effectue les traitements nécésaires). Une option pourrait être proposée à l'utilisateur : écouter directement le flux des données en temps réel à travers des sockets en utilisant le protocol UDP par exemple.

- **Optimiser le code:**

Une étape d'optmisation du code a été abordée avant la sortie de la première version, cette étape nous a permis de réduire considérablement les dépendances entre les classes et de corriger quelques fuites mémoire. Néanmoins, on juge que des améliorations peuvent être portées à la classe MainWindow. En effet, les opérations d'initialisation de l'interface graphique peuvent être déportées dans la classe TableauDeBord, qui est la classe centrale s'occupant des instantiations et initialisations nécéssaires pour le bon fonctionnement de l'application (partie métier).

- **Effectuer les tests:**

Lors de la phase développement, nous avons effectué des tests sur les principales fonctionnalités de l'application: traitement du bruit, intégration, chargement du fichier. Ces tests ont permis

d'évaluer le code à mi-projet. Nous avons aussi testé l'interface graphique à la fin de la phase du développement. Néanmoins plusieurs autres fonctionnalités n'ont pas étaient testées comme l'algorithme de classification. Les développeurs qui reprendront le projet pourront implémenter les tests dans la classe dédiée.

VI. Annexe 1 : intégration de la librairie Aquila au projet

1) Compilation de la librairie aquila :
cmake ./ -DCMAKE_INSTALL_PREFIX="chemin_vers_les_sources_aquila/lib"
make
make install

2) Copie à la main de la librairie générée dans librairie libOoura_fft.a dans
../../traitement_du_signal/traitement_du_signal/lib
=> cette librairie correspond à du code écrit en C et appelée par les deux autres via méthodes extern
(liaison statique)

3) Pour compatibilité ISO C++ 2011
Ajout de l'option dans le parametrage de qmake

-r QMAKE_CXXFLAGS+=-std=gnu++11

4) Ajout des trois bibliothèques dans qt en mode bibliothèque externe

VII. Annexe 2 : installation de QT3dD

Pour créer une interface graphique qui supporte des widgets en OpenGL, il faut passer par un plugin de QTQUICK qui s'appelle QT3D.

Pour l'installer soit:

Vous êtes chanceux:

Vous avez Qt et QT-Creator en denière version avec le qmake aussi en dernière version pour cela il faut simplement:
* dans un dossier faire: git clone https://gitorious.org/qt/qt3d.git qt3d
* Ouvrir QTCreator
* Ouvrir le fichier .pro du répértoire que vous avez pullé
* le compiler en mode RELEASE (attention pas en mode debug)
* aller au dossier ou vous avez buildé (cd build_qt3d...)
* écrire: make install
* relancer QTCreator -> créer un nouveau projet -> dans le fichier .pro ajouter QT += 3dquick -> compiler le projet vide et
ça ne doit pas renvoyer d'erreurs -> dans ce cas bravo vous avez QT3D

Vous n'êtes pas chanceux (comme moi):
===
Il faut suivre ces étapes pas à pas:

Installer la dernière version de qmake:
==
* Ajouter une nouvelle ligne (nouveau dépot) dans votre sources.list (vous ne savez pas comment ajouter un nouveau dépot: google.fr):
deb http://ftp.fr.debian.org/debian wheezy-backports main
* en ligne de commande : sudo apt-get install qt5-qmake

Installer la dernière version de QTCreator:
==
* Avant d'installer la dernière version, supprimez l'ancienne version de qt-sdk: apt-get remove qt-sdk qt-creator
* Télécharger la dernière version stable depuis: http://download.qt-project.org/official_releases/online_installers/qt-opensource-linux-x64-1.6.0-5-online.run
* double cliquez sur le fichier que vous avez télécharger (avant faites un chmod +x lenomdufichier.run)
* Dans la fenêtre qui s'affiche cliquez sur paramètres pour modifier les paramètres du proxy de l'installateur
* Cliquez sur suivant pour lancer l'installation
* à la fin de l'installation -> lancer QtCreator
* Ouvrez QT Creator -> OUtils -> Options.. -> Compiler et Executer -> vous devez voir QT 5.3 GCC dans autodetectée et QT 4 dans manuel
supprimer la manuel et ne laisser que la version autodetectée .. ou encore mieux mêe dans la version manuelle pointé sur LE QMAKE de Qt5.3
* Maintenant que vous avez les dernières versions recquises : reprener la partie : VOUS ETES CHANCEUX

Si vous avez des problèmes venez vers moi ou mail: mohamed-amjad.lasri@ensg.eu

Se former à Qt3D : http://qt-project.org/wiki/Introduction-to-Qt3D