Rapport de projet d'électronique

Centrale inertielle

MOUGIN Paul
RATTRAPAGE DE 4ETI 2013

Juin 2015

Table des matières

Introduction

Le but de ce projet est de créer à l'aide d'une carte électronique STM32F3 Discovery une centrale inertielle en utilisant les accéléromètres et les gyroscopes embarqués dans la carte.

Tous les calculs doivent être effectué par le microprocesseur STM32 sur la carte embarquée. Les informations de positions calculées seront alors envoyer à un terminal via une liaison série.

Études préalables

1.1 Quid d'une centrale inertielle ?

> *Une centrale à inertie ou centrale inertielle est un instrument utilisé en navigation, capable d'intégrer les mouvements d'un mobile (accélération et vitesse angulaire) pour estimer son orientation (angles de roulis, de tangage et de cap), sa vitesse linéaire et sa position. L'estimation de position est relative au point de départ ou au dernier point de recalage.* [1]

En effet, une centrale inertielle est un appareil de mesure permettant de connaître la position de l'objet sur lequel elle est fixée sans avoir besoin d'informations extérieures. La seule connaissance de l'accélération linéaire selon trois axes ainsi que les vitesses angulaires autours de ces trois même axes permet de calculer la position relative de l'objet par rapport à sont point de départ.

Cet instrument de mesure est encore utilisé dans tous les avions de ligne pour compenser l'imprécision des systèmes satellitaires comme le GPS.

La précision d'une centrale inertielle varie en fonction de la précision des capteurs (accéléromètres et gyroscopes) utilisés mais aussi en fonction de algorithme de calcul utilisé pour traiter les données.

1.2 Considérations théoriques

Calcul de la position

Les capteurs nous remontent une accélération linéaire ainsi qu'une vitesse angulaire suivant les trois axes (que nous appellerons X, Y et Z) de la centrale inertielle.
Pour déterminer la position, le processeur doit déjà effectuer une intégration de l'accélération pour déterminer les vitesses linéaires par rapport au sol :

$$\vec{V} = \int \vec{\gamma} dt \tag{1.1}$$

Puis pour déterminer la position, il doit encore intégrer la vitesse en prenant en compte la position de départ x_0 :

$$x = \int \vec{V} dt + x_0 \tag{1.2}$$

Il existe plusieurs algorithmes permettant de calculer les intégrales succéssives ; méthodes plus ou moins complexes et plus ou moins précises qui seront explorées dans la partie *Calcul de l'intégration.*

Ces trois positions (une suivant chaque axe de la centrale) sont calculé par rapport au repère de la centrale et non le repère terrestre supposé galiléen. Or nous souhaitons connaître la position de l'objet dans un référentiel absolu.
Pour cela il nous faut faire un changement de base, mais avant tout, connaître l'orientation de la centrale inertielle dans le repère de base.

Calcul de l'orientation

Pour connaître l'orientation de la centrale il nous faut connaître les angles de roulis, de tangage et de lacet (roll, pitch et heading en anglais) représentés sur la figure suivante :

1. définition de Wikipedia

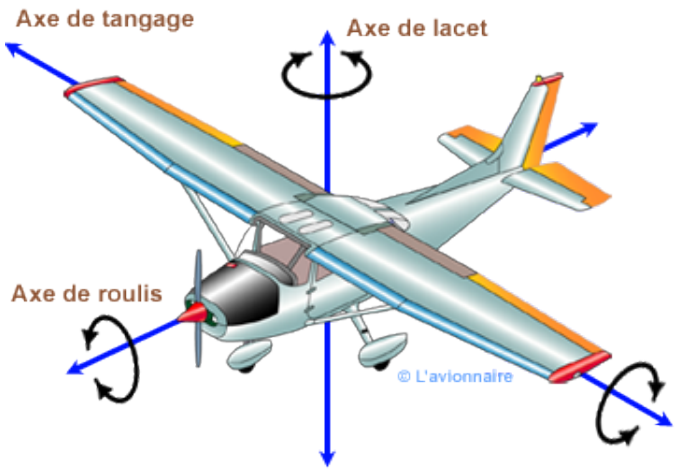

FIGURE 1.1 – Angles de roulis, tangage et lacet

L'angle de roulis et de tangage sont calculables grâce aux composantes du vecteur accélération mesurées par accéléromètre de la carte notée $G_{Acc} = (G_x, G_y, G_z)$:

$$Roll = \arctan\left(\frac{G_y}{G_z}\right) \tag{1.3}$$

$$Pitch = -\arctan\left(\frac{G_x}{\sqrt{G_y^2 + G_z^2}}\right) \tag{1.4}$$

L'angle de roulis (Roll) a une amplitude de $[-180; 180]$ et l'angle de tangage (Pitch) a une amplitude de $[-90; 90]$. Il n'est pas possible d'obtenir une amplitude de $360°$ sur les deux angles, il a donc été décidé de limiter l'angle de tangage sur une plage de $180°$.
Ces deux formules sont tirées de la théorie des angles d'Euler. La démonstration de ces formules est détaillée dans la documentation *Tilt Sensing Using a Three-Axis Accelerometer* de FREESCALE SEMICONDUCTOR dont la référence se trouve en ANNEXE.

Pour calculer l'angle de lacet, il est nécessaire d'utiliser le magnétomètre de la carte car l'accéléromètre seul ne nous permet pas d'aller plus loin. Nous récupérons les données du magnétomètre dans un vecteur $M = (M_x, M_y, M_z)$ ce qui nous permet de calculer les *"'angles d'inclinaisons"'* aussi appelés *tilted angles* :

$$x_{tilted} = M_x \cos(Pitch) + M_z \sin(Pitch) \tag{1.5}$$

$$y_{tilted} = M_x \sin(Roll) \sin(Pitch) + M_y \cos(Roll) - M_z \sin(Roll) \cos(Pitch) \tag{1.6}$$

Ces *tilted angles* nous permettent ensuite de calculer l'angle de lacet (Heading) :

$$Heading = \arctan\left(\frac{y_{tilted}}{x_{tilted}}\right) \tag{1.7}$$

Calcul du changement de base

Une fois l'orientation de la centrale par rapport au sol connue, il est possible de calculer le vecteur accélération, vitesse ou position de la centrale inertielle dans le repère absolu grâce à la théorie des *angles d'Euler* :

Connaissant les angles θ, ϕ, ψ respectivement autour de x, y et z on peut calculer la matrice de rotation R :

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix} \cdot \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.8}$$

Ce qui donne la forme général suivante :

$$R = \begin{pmatrix} \cos\psi\cos\phi & -\sin\psi\cos\theta + \cos\psi\sin\phi\sin\theta & \sin\psi\sin\theta + \cos\psi\sin\phi\cos\theta \\ \sin\psi\cos\phi & \cos\psi\cos\theta + \sin\psi\sin\phi\sin\theta & -\cos\psi\sin\theta + \sin\psi\sin\phi\cos\theta \\ -\sin\phi & \cos\phi\sin\theta & \cos\phi\cos\theta \end{pmatrix} \tag{1.9}$$

Ce qui permet ensuite, connaissant le vecteur position $POS \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ dans le repère de la centrale de calculer le vecteur position $POS' \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix}$ dans le repère absolu :

$$POS' = R \cdot POS \tag{1.10}$$

Calcul de l'intégration

comme dit plus haut, il existe plusieurs méthodes de calcul des intégrales certaines plus complexes et plus précises que d'autres. Nous allons en exposer deux :

La méthode d'Euler

La méthode d'Euler est une méthode numérique élémentaire de résolution d'équation différentielles du premier ordre. Elle consiste sur un intervalle de temps le plus court possible d'approximer une courbe grâce à ça dérivée.

On sait par la théorie de la dérivation que pour n'importe quelle fonction $f(x)$ dont la primitive est $F(x)$ sur un intervalle défini on a :

$$\lim_{h \to 0} \frac{F(x+h) - F(x)}{h} = f(x) \tag{1.11}$$

aussi par conséquent si h est très petit il est possible de calculer la Primitive de $f(x)$ par récurrence :

$$F(x+h) \approx F(x) + f(x)h \tag{1.12}$$

Cette méthode très simpliste permet d'approximer l'intégration de manière assez grossière. L'erreur de cette méthode est d'ordre 2 soit $o(h^2)$ ce qui est assez élevé. Cette erreur est d'autant plus élevé lorsque l'on pratique cette méthode deux fois à la suite comme c'est le cas pour la centrale inertielle.

Méthode de Simpson

La méthode de Simpson utilise une approximation d'ordre 2 de f grâce au polynôme quadratique P. Le polynôme représente une parabole entre a et b qui prend les même valeurs que f aux points a, b et $m = \frac{(a+b)}{2}$. On connaît alors l'expression de cette parabole grâce à l'interpolation de Lagrange :

$$P(x) = f(a)\frac{(x-m)(x-b)}{(a-m)(a-b)} + f(m)\frac{(x-a)x-b)}{(m-a)(m-b)} + f(b)\frac{(x-a)(x-m)}{(b-a)(b-m)} \tag{1.13}$$

Le polynôme étant plus facile a intégrer on peut ainsi approximer l'intégrale de f sur l'intervalle $[a, b]$:

$$\int_a^b f(x)dx \approx \int_a^b P(x)dx = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] \tag{1.14}$$

Ainsi il est possible d'approximer l'intégrale de l'accélération pour en déduire la vitesse, en rendant l'intervalle $[a, b]$ le plus petit possible et en effectuant la somme successive des approximations.

Une méthode plus précise que la méthode d'Euler puisque sont erreur est d'ordre 4 soit $o(h^4)$ ce qui, concaténé à elle même laisse une erreur d'ordre 2, ce qui peut être suffisant pour notre centrale.

Réalisation du projet

2.1 Matériel mis à disposition

Pour la réalisation de cette centrale inertielle, le seul élément imposé était la carte de développement : une *STM32F3-Discovery* de chez STMicroelectronics

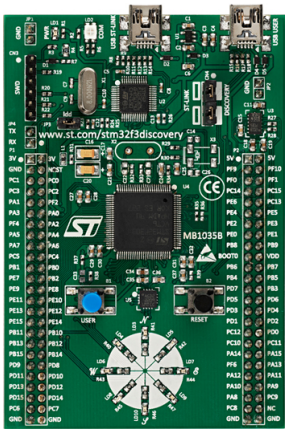

Figure 2.1 – Carte *STM32F3-Discovery* de STMicroelectronics

Cette carte d'évaluation est équipée d'un microprocesseur ARM Cortex-M4, 256 KB de mémoire Flash et 46 KB de mémoire RAM. La carte embarque un accéléromètre LSM303DLHC trois axes et un gyroscope L3GD20 trois axes. La carte est également équipée de LED de différente couleurs ainsi que de deux boutons poussoirs.

Au niveau de la communication extérieur, la carte est équipée de deux port mini-USB. L'un, le ST-LINK/V2 permet de programmer et de débugger la carte à travers un logiciel dédié (nous utiliserons *Keil µVision5* pour cela).
L'autre permet d'établir une communication avec d'autres éléments. nous l'utiliserons pour établir une liaison série entre la carte et un hyperterminal (*Termite*) ou notre interface utilisateur développé en *Qt*.

2.2 Implémentation sur la carte STM32F3

Les travaux réalisés sur la carte sont divisés en plusieurs points indépendants : la récupération des données des capteurs, leur traitement et la communication via la liaison série USB.
Le projet est à la base le projet de démonstration fournit par STMicroelectronics auquel on été retirés les exemples et fonctionnalité qui ne nous intéressaient pas et auquel ont été rajoutées des librairies permettant de réaliser certaines briques du programme.

2.2.1 Récupération des données des capteurs

La première chose à faire avant de vouloir traiter une information est de la mesurer via un capteur puis de la récupérer pour l'utiliser plus tard dans les calculs.

Les fonctions utilisées pour récupérer les données des capteurs sont déjà implémentées dans les exemples par ST-Microelectronics. Nous utilisons ainsi *Demo_CompassConfig(void)* pour initialiser le capteur LSM303DLHC puis les fonctions *Demo_CompassReadAcc(float* pfData)* et *Demo_CompassReadMag (float pfData)* pour récupérer les données de l'accéléromètre et du magnétomètre.

Demo_CompassConfig(void)

C'est la fonction qui est appelée pendant l'initialisation du programme et qui permet de configurer correctement les différents capteurs utilisés plus tard dans le programme.
Les paramètres peuvent allez du mode d'alimentation à l'échelle utiliser tout en passant par la configuration des filtres utilisés pour traiter les signaux des capteurs.

Demo_CompassReadAcc(float* pfData) & Demo_CompassReadMag (float* pfData)

Ce sont deux fonctions permettent de récupérer les informations des capteurs et de les stocker dans un pointer mis en paramètre. Ces fonctions permettent aussi la calibration des données en fonction du mode choisi à l'initialisation.

2.2.2 Traitements des données

Une fois les données récupérer depuis les capteurs ils faut les traiter afin de calculer ce que nous cherchons : l'orientation de la centrale inertielle grâce à ses angles de roulis, de tangage et de lacet ainsi que sa position relative par rapport au référentiel terrestre.

ReadOrientation(float *pHeading, float *pRoll, float *pPitch)

C'est la fonction qui permet de retourner dans les pointeurs en paramètres les angles d'orientation en effectuant les calculs détaillés dans la partie théorique :

```
174  void ReadOrientation(float *pHeading, float *pRoll, float *pPitch)
175  {
176    uint8_t i = 0;
177    float MagBuffer[3] = {0.0f}, AccBuffer[3] = {0.0f};
178    float sinRoll, cosRoll, sinPitch, cosPitch = 0.0f, RollAng = 0.0f, PitchAng = 0.0f;
179    float fTiltedX, fTiltedY = 0.0f;
180    float HeadingValue = 0.0f;
181
182    ReadMagnetometer(MagBuffer);
183    ReadAccelerometer(AccBuffer);
184
185    //Low Pass Filter
186      //AccBuffer
187    for (i=0; i<3; i++)
188      fAccBuffer[i] = AccBuffer[i] * alpha + (fAccBuffer[i] * (1.0 - alpha));
189      //MagBuffer
190    for (i=0; i<3; i++)
191      fMagBuffer[i] = MagBuffer[i] * alpha + (fMagBuffer[i] * (1.0 - alpha));
192
193    RollAng = atan2f(fAccBuffer[1], fAccBuffer[2]);
194    PitchAng = -atan2f(fAccBuffer[0], sqrt(fAccBuffer[1]*fAccBuffer[1] + fAccBuffer[2]*fAccBuffer[2]));
195
196    sinRoll = sinf(RollAng);
197    cosRoll = cosf(RollAng);
198    sinPitch = sinf(PitchAng);
199    cosPitch = cosf(PitchAng);
200
201    fTiltedX = fMagBuffer[0]*cosRoll+fMagBuffer[2]*sinPitch;
202    fTiltedY = fMagBuffer[0]*sinRoll*sinPitch + fMagBuffer[1]*cosRoll - fMagBuffer[2]*sinRoll*cosPitch;
203
204    HeadingValue = atan2f(fTiltedY,fTiltedX);
205
206     // We cannot correct for tilt over 40 degrees with this alg, if the board is tilted as such, return 0.
207    if(RollAng > 0.78f || RollAng < -0.78f || PitchAng > 0.78f || PitchAng < -0.78f)
208    {
209      HeadingValue = 0;
210    }
211
212    *pHeading = HeadingValue*180/PI;
213    *pRoll = RollAng*180/PI;
214    *pPitch = PitchAng*180/PI;
215  }
```

FIGURE 2.2 – Fonction void ReadOrientation(float *pHeading, float *pRoll, float *pPitch)

Les calculs de les angles de roulis et de tangage fonctionne très bien et donnent des résultats fiables et utilisables. Le problème vient du calcule de l'angle de lacet qui n'est précis qu'à l'horizontal. Une fois la carte dans une orientation quelconque, la valeur calculée de l'angle n'est plus correcte et en peut alors plus être utilisée ensuite, l'erreur ce répercutant dans les couches supérieures.

EulerMethode(float *AccBuffer, float *PosBuffer, int delay)

C'est la fonction qui permet de retourner le vecteur position de la carte en fonction du vecteur accélération en utilisant la méthode d'Euler. Pour ce faire il faut aussi mettre en paramètre le temps de délais entre deux mesures et faire l'hypothèse que les conditions initiales au début de la mesure sont nulle (position et vitesse).

```
227  void EulerCalcul(float *primitive, float *function, int delay)
228 {
229    int i;
230    for(i=0; i<3; i++)
231      primitive[i] += primitive[i] + function[i]*delay;
232  }
233
234  void EulerMethode(float *AccBuffer, float *PosBuffer)
235 {
236    float SpeedBuffer[3] = {0.0f};
237    float TMPBuffer[3] = {0.0f};
238    EulerCalcul(SpeedBuffer, AccBuffer, DELAY);
239    EulerCalcul(TMPBuffer, SpeedBuffer, DELAY);
240
241    PosBuffer = TMPBuffer;
242  }
243
```

FIGURE 2.3 – Fonction void EulerMethode(float *AccBuffer, float *PosBuffer, int delay)

Cette fonction n'a pas été testée.

SimpsonMethode

C'est la fonction qui permet de retourner dans les pointeurs en paramètre les positions relatives de la carte en utilisant la méthode de Simpson.
Cette méthode n'a pas encore été implémentée.

Basechangement(float *posBoard, float *posAbsolute, float roll, float pitch, float yaw)

Cette fonction permet d'effectuer le changement de repère du vecteur position de la carte et ainsi calculer la position de la carte dans le repère absolu connaissant son déplacement dans son repère propre et les angles de roulis, tangage et lacet de la carte.

```
279  void BaseChangement(float *posBoard, float *posAbsolute, float roll, float pitch, float yaw)
280 {
281    float X, Y, Z;
282    X = cos(yaw)*cos(pitch)*posBoard[0]
283      + (-sin(yaw)*cos(roll)+cos(yaw)*sin(pitch)*sin(roll))*posBoard[1]
284      + (sin(yaw)*sin(roll)+cos(yaw)*sin(pitch)*cos(roll))*posBoard[2];
285
286    Y = sin(yaw)*cos(pitch)*posBoard[0]
287      + (cos(yaw)*cos(roll)+sin(yaw)*sin(pitch)*sin(roll))*posBoard[1]
288      + (-cos(yaw)*sin(roll)+sin(yaw)*sin(pitch)*cos(roll))*posBoard[2];
289
290    Z = -sin(pitch)*posBoard[0]
291      + cos(pitch)*sin(roll)*posBoard[1]
292      + cos(pitch)*cos(roll)*posBoard[2];
293
294    float TMPBuffer[3] = {X,Y,Z};
295    posAbsolute = TMPBuffer;
296 }
```

FIGURE 2.4 – Fonction void Basechangement(float *posBoard, float *posAbsolute, float roll, float pitch, float yaw)

2.2.3 Communication série

Une fois toutes les données calculées, il faut les afficher à l'utilisateur. La carte ne déposant d'aucun affichage intégré, mis à part ses LED, ce qui rend l'affichage pas vraiment évident, il faut communiquer ces données à un autre appareil - comme un PC - qui pourra les afficher.

Avant de parler d'une quelconque interface Homme-Machine, il est donc important de transmettre correctement ces données via une liaison série de type Virtual COM entre le port USB-USER de la carte et un port USB d'un ordinateur. La librairie permettant d'envoyer et de recevoir des message à travers une liaison série est aussi une librairie créée et mise à disposition par STMICROELECTRONICS dans son projet de démonstration.

En connectant le port USB-USER de la carte à un PC, ce dernier doit installer le bon périphérique et ainsi reconnaître un port de communication VCP. Pour ce connecter à ce port, il faut que la liaison soit effective entre la carte et le PC, que la liaison soit initialisé du côté de la carte et que du côté du PC, on se connecte au port de communication alloué avec les bons paramètres, à savoir :

- BaudRate : 115200
- Bits de donnée : 8
- Parité : Non
- Bit de stop : 1
- Contrôle de flux : Non

Une fois la connexion établie entre les deux entités, on peut envoyer à travers la liaison série grâce à la fonction *VCP_PutStr(char* buffer)* où le buffer est une chaine de caractères contenant le message que l'on souhaite envoyer. Dans notre cas, le protocole est le suivant : $"X : \%11.3f\ Y : \%11.3f\ Z : \%11.3f \backslash n"$ où *%11.3f* est un nombre flottant avec 3 chiffres avant et après la virgule. Ce protocole permet l'affichage des informations directement dans un hyper terminal comme *Termite* mais aussi la concaténation des informations par un autre logiciel connaissant ce protocole, comme c'est le cas pour l'interface utilisateur.

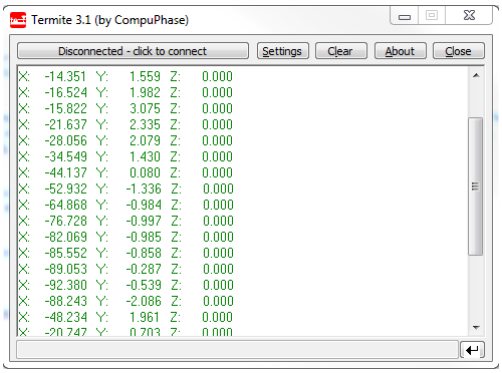

FIGURE 2.5 – Hyper terminal affichant les valeurs des angles calculés par la carte

2.3 Interface utilisateur

L'interface utilisateur a été réalisée en *C++* à l'aide du farmework *Qt* qui permet une utilisation multiplateforme et une grande liberté de programmation tout en ayant une communauté très active.

L'interface utilisateur permet la connexion à la liaison série initialisée par la carte. Cette connexion est soumise à différent paramètres défini par l'utilisateur lui même dans l'écran *Settings*.

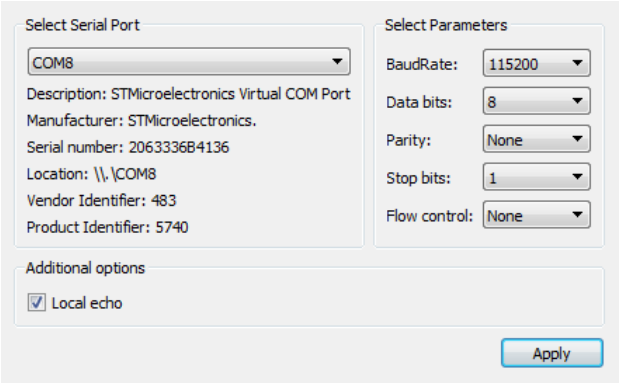

FIGURE 2.6 – Écran *Settings*

Si les paramètres rentré par l'utilisateur sont correct, il est alors possible de se connecter à la liaison série. Il faut tout fois que la liaison série ne soit pas déjà connecté à un autre programme, comme un hyper terminal, puisque la liaison série ne permet pas le multicast.

Une fois connectée, l'interface récupère les messages envoyé par la carte sur la liaison série et concatène les données pour les réinjecter dans les éléments de l'interface graphique.

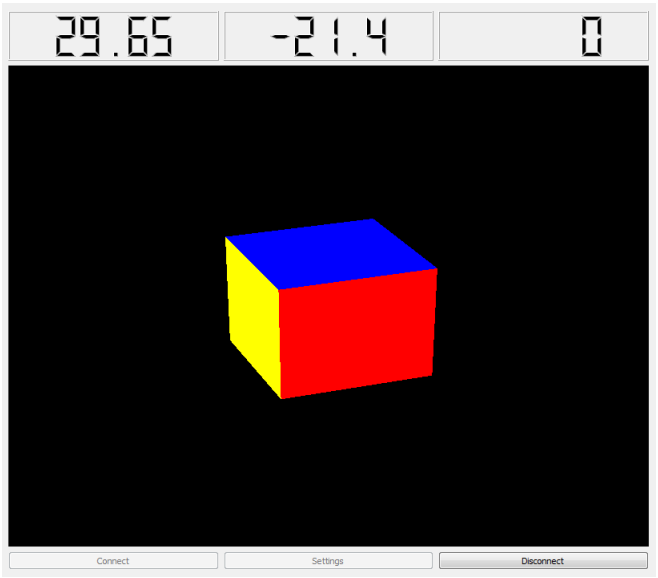

FIGURE 2.7 – Interface graphique

Les angles de roulis, tangage et lacet sont indiqués par les afficheur LCD en haut de l'interface alors que la partie centrale permet quand à elle d'orienter un parallélépipède rectangle en fonction de l'orientation de la carte. Toute la partie 3D est réalisé à l'aide des librairies *QtOpenGL* et *QGLWidget*.

Reste à faire et améliorations

3.1 Calcul de l'angle de lacet

Il serait bien de réussir a corriger la mesure de l'angle de lacet. Cela peut se faire en intégrant les données du gyroscope pour retrouver la position angulaire autour de l'axe z.

Toute fois cette méthode n'est pas non plus parfaite puisqu'elle est amputée d'une erreur de *dérive* que le gyroscope ne peut mesurer. Il faudrait alors mettre en abîme les deux méthodes de calcul afin que l'une compense l'erreur de l'autre et vice et versa.

3.2 Implémentation et comparaison de la méthode de Simpson

La méthode de Simpson n'a pas été implémentée par soucis de temps. Il s'agirait ici de l'implémenter pour ensuite la comparée avec la méthode d'Euler pour prouver qu'elle est plus performante.
Ceci dit, la mesure de l'angle de lacet et l'implémentation de la matrice de passage, cette comparaison est assez compromise.

D'autre méthode pourraient être aussi implémentées, comme la méthode du filtre de Kalman ou le filtre de Madgwick qui sont des méthode notamment utilisées dans les centrales inertielles industrielles.

3.3 Communication Interface vers carte

On pourrait utiliser la liaison série dans le sens inverse. C'est-à-dire au lieu de faire du Broadcast pur et simple de l'information, l'interface utilisateur pourrait aussi écrire sur la liaison série et communiquer avec la carte pour par exemple lui demander d'utiliser une méthode plutôt qu'un autre ou modifier le temps de cycle de la carte.

Conclusion

Le projet permet actuellement de pouvoir visualiser l'orientation de la carte *STM32F3* grâce à ces accéléromètres sur une interface graphique qui récupère les données via une liaison série USB.

Cela amorce les prémisses d'une centrale inertielle, mais comme nous l'avons vu, beaucoup de chemins et d'améliorations restent encore à faire, principalement en matière d'algorithme.
La carte en elle même est capable de supporter la charge de travail d'une centrale inertielle complète, même si les temps de cycle sont parfois problématiques pour des méthodes trop peu précises.