



RAPPORT DE PROJET D'ÉLECTRONIQUE

Centrale inertielle

MOUGIN PAUL
RATTRAPAGE DE 4ETI 2013

Juin 2015

Table des matières

Introduction	2
1 Études préalables	3
1.1 Quid d'une centrale inertielle ?	3
1.2 Considérations théoriques	3
2 Réalisation du projet	6
2.1 Matériel mis à disposition	6
2.2 Implémentation sur la carte STM32F3	6
2.2.1 Récupération des données des capteurs	6
2.2.2 Traitements des données	7
2.2.3 Communication série	8
2.3 Interface utilisateur	9
3 Reste à faire et améliorations	11
3.1 Calcul de l'angle de lacet	11
3.2 Implémentation et comparaison de la méthode de Simpson	11
3.3 Communication Interface vers carte	11
Conclusion	12

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 effectués par le microcontrôleur STM32 sur la carte embarquée. Les informations de positions calculées seront alors envoyées 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 autour de ces trois mêmes axes permet de calculer la position relative de l'objet par rapport à son 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 l'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 :

$$\vec{V} = \int \vec{\gamma} dt \quad (1.1)$$

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

$$x = \int \vec{V} dt + x_0 \quad (1.2)$$

Il existe plusieurs algorithmes permettant de calculer les intégrales successives ; algorithmes plus ou moins complexes et plus ou moins précis qui seront exposés dans la partie *Calcul de l'intégration*.

Ces trois positions (une suivant chaque axe de la centrale) sont calculées 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 faut effectuer un changement de base, mais avant tout, connaître l'orientation de la centrale inertielle par rapport au sol.

Calcul de l'orientation

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

1. définition de Wikipedia

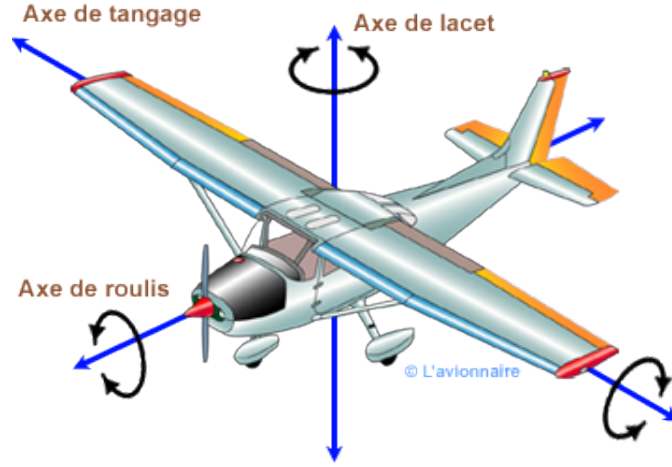


FIGURE 1.1 – Angles de roulis, tangage et lacet

Les angles de roulis et de tangage sont calculables grâce aux composantes du vecteur accélération mesurées par l'accéléromètre de la carte. Nous les noterons $G_{Acc} = (G_x, G_y, G_z)$:

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

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

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 en même temps, 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* :

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

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

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

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

Une autre méthode permet aussi de calculer les angles de notre carte. Il s'agit d'utiliser les mesures du gyroscope qui nous donne la valeur de la vitesse angulaire suivant les trois axes de notre carte. Comme pour la position linéaire de la carte, nous pouvons calculer la position angulaire (α) en intégrant la vitesse angulaire ($\vec{\omega}$) :

$$\alpha = \int \vec{\omega} dt + \alpha_0 \quad (1.8)$$

Calcul du changement de base

Une fois l'orientation de la centrale par rapport au sol connue, il est possible de calculer les vecteurs accélération, vitesse et 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, il est possible de calculer la matrice de rotation R :

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{pmatrix} \cdot \begin{pmatrix} \cos \phi & 0 & \sin \phi \\ 0 & 1 & 0 \\ -\sin \phi & 0 & \cos \phi \end{pmatrix} \cdot \begin{pmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1.9)$$

Ce qui donne la forme général suivante :

$$R = \begin{pmatrix} \cos \psi \cos \phi & -\sin \psi \cos \theta + \cos \psi \sin \phi \sin \theta & \sin \psi \sin \theta + \cos \psi \sin \phi \cos \theta \\ \sin \psi \cos \phi & \cos \psi \cos \theta + \sin \psi \sin \phi \sin \theta & -\cos \psi \sin \theta + \sin \psi \sin \phi \cos \theta \\ -\sin \phi & \cos \phi \sin \theta & \cos \phi \cos \theta \end{pmatrix} \quad (1.10)$$

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 :

$$POS' = POS \cdot R \quad (1.11)$$

Calcul de l'intégration

Comme dit précédemment, 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'équations différentielles du premier ordre. Elle consiste sur un intervalle de temps le plus court possible d'approximer une courbe grâce à ça dérivée.

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

$$\lim_{h \rightarrow 0} \frac{F(x+h) - F(x)}{h} = f(x) \quad (1.12)$$

Aussi, par conséquent, si h est très petit, il est possible de calculer la primitive de $f(x)$ par récurrence :

$$F(x+h) \approx F(x) + f(x)h \quad (1.13)$$

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 élevée lorsque cette méthode est appliquée deux fois à la suite comme c'est le cas pour le calcul de la position depuis l'accélération.

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 mêmes 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 :

$$P(x) = f(a) \frac{(x-m)(x-b)}{(a-m)(a-b)} + f(m) \frac{(x-a)(x-b)}{(m-a)(m-b)} + f(b) \frac{(x-a)(x-m)}{(b-a)(b-m)} \quad (1.14)$$

Un polynôme étant plus facile à intégrer nous pouvons ainsi approximer l'intégrale de f sur l'intervalle $[a, b]$:

$$\int_a^b f(x)dx \approx \int_a^b P(x)dx = \frac{b-a}{6} \left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right] \quad (1.15)$$

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 de ces approximations.

Une méthode plus précise que la méthode d'Euler puisque son erreur est d'ordre 4 soit $o(h^4)$ ce qui, concaténée 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

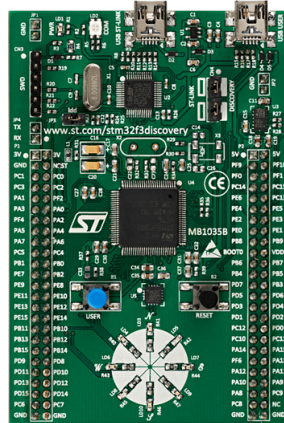


FIGURE 2.1 – Carte *STM32F3-Discovery* de STMicroelectronics

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 différentes couleurs ainsi que de deux boutons poussoirs (USER et RESET).

Au niveau de la communication extérieure, la carte est équipée de deux port mini-USB. L'un, le ST-LINK/V2 permet de programmer et de déboguer la carte à travers un logiciel dédié (nous utiliserons *Keil μVision5* pour cela). L'autre permet d'établir une communication avec d'autres éléments que nous utiliserons pour établir une liaison série entre la carte et un hyper terminal (*Termite*) ou notre interface utilisateur développée 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 fourni par STMicroelectronics auquel ont été retirés les exemples et fonctionnalités qui ne nous intéressaient pas et auquel ont été rajoutées des bibliothèques permettant de réaliser certaines briques du programme comme la bibliothèque VCOM.

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 STMicroelectronics. Nous utilisons ainsi *InitAccAndMag(void)* pour initialiser le capteur LSM303DLHC puis les fonctions *ReadAccelerometer(float* pfData)* et *ReadMagnetometer(float pfData)* pour récupérer les données de l'accéléromètre et du magnétomètre. C'est fonction sont détaillée dans le fichier *imu_devs.c*.

InitAccAndMag(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 aller du mode d'alimentation à l'échelle utilisée tout en passant par la configuration du filtrage des signaux des capteurs.

ReadAccelerometer(float* pData) & ReadMagnetometer(float* pData)

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

Concernant le gyroscope, le principe est exactement le même avec les fonctions *InitGyro(void)* et *ReadGyro(float* pData)*.

2.2.2 Traitements des données

Une fois les données récupérées depuis les capteurs il 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)

L'utilisation d'un filtrage passe bas sur les données permet de lisser celles-ci les unes par rapports aux autres afin d'endiguer toute erreur de mesure du capteur.

Les calculs des angles de roulis et de tangage fonctionne très bien et donnent des résultats fiables et utilisables. Le problème vient du calcul 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 la donnée ne peut plus être utilisée ensuite, l'erreur se répercutant dans les couches supérieures de l'algorithme.

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 nulles (vitesse et position).

```
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 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 disposant 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 messages 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, il est possible envoyer des messages à travers la liaison série grâce à la fonction `VCP_PutStr(char* buffer)` où le buffer est une chaîne de caractères contenant le message que l'on souhaite envoyer. Dans notre cas, le protocole est le suivant :

$$X : \%11.3f \quad Y : \%11.3f \quad Z : \%11.3f \backslash n \quad (2.1)$$

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.

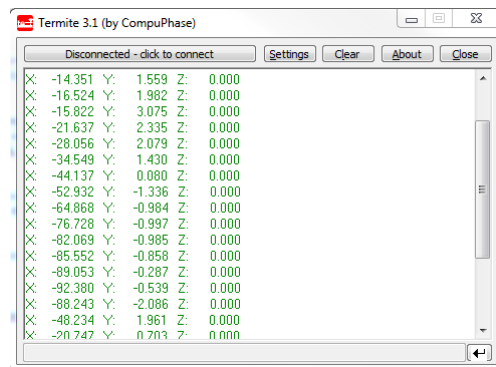


FIGURE 2.5 – Hyper terminal affichant les valeurs des angles calculés par la carte

2.3 Interface utilisateur

L'interface utilisateur a été réalisée en C++ à l'aide du framework *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 différents paramètres définis par l'utilisateur lui-même dans l'écran *Settings*.

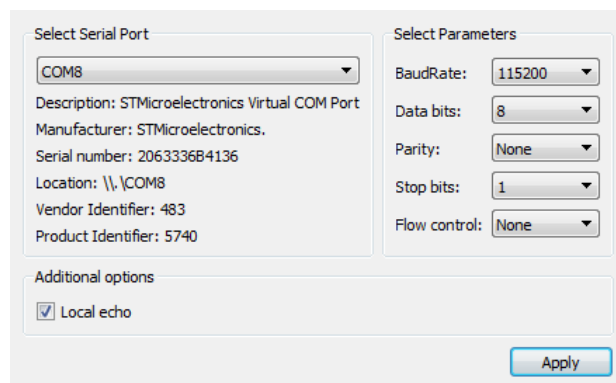


FIGURE 2.6 – Écran *Settings*

Si les paramètres rentrés par l'utilisateur sont correct, il est alors possible de se connecter à la liaison série. Il faut toutefois que la liaison série ne soit pas déjà connectée 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 envoyés par la carte sur la liaison série et concatène les données pour les réinjecter dans les éléments de l'interface graphique.

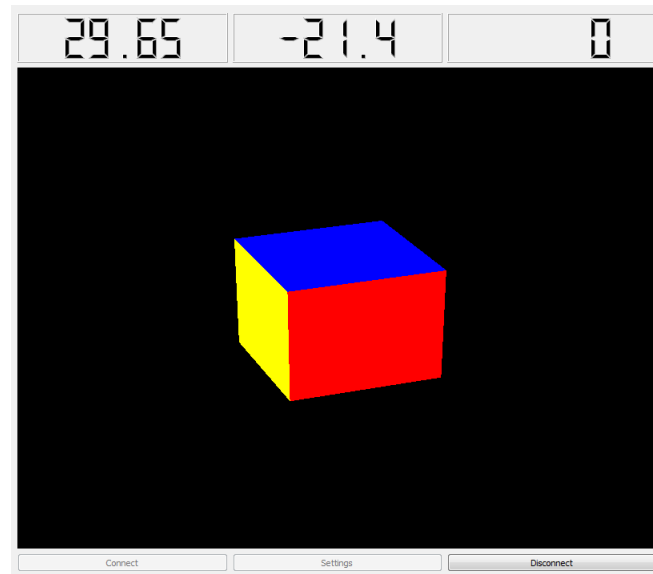


FIGURE 2.7 – Interface graphique

Les angles de roulis, tangage et lacet sont indiqués par les afficheurs 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 réalisée à l'aide des bibliothèques *QtOpenGL* et *QGLWidget*.

L'ensemble des codes sources est disponible sur un *repository GitHub* dont le lien est donné en ANNEXE.

Reste à faire et améliorations

3.1 Calcul de l'angle de lacet

Il serait bien de réussir à 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.

Toutefois 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 comparer avec la méthode d'Euler pour prouver qu'elle est plus performante.

Ceci dit, la méthode de Simpson nécessite la connaissance de 3 points de mesure pour faire l'approximation, ce qui double donc le temps de cycle de la méthode comparée à celle d'Euler.

D'autres méthodes pourraient être aussi implémentées, comme le filtrage de Kalman ou le filtrage de Madgwick qui sont des méthodes notamment utilisées dans les centrales inertielles industrielles. Leur algorithme fait entrer en jeu des quaternions et malgré quelques lectures et revues de code sur ce sujet, il semblait très compliqué d'expliquer vraiment en quoi consistaient ces méthodes.

3.3 Communication Interface vers carte

il pourrait être possible d'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'une autre ou modifier le temps de cycle des calculs. La librairie implémentée dans le projet permet cette amélioration, elle n'a juste pas été tentée.

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 chemin 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.