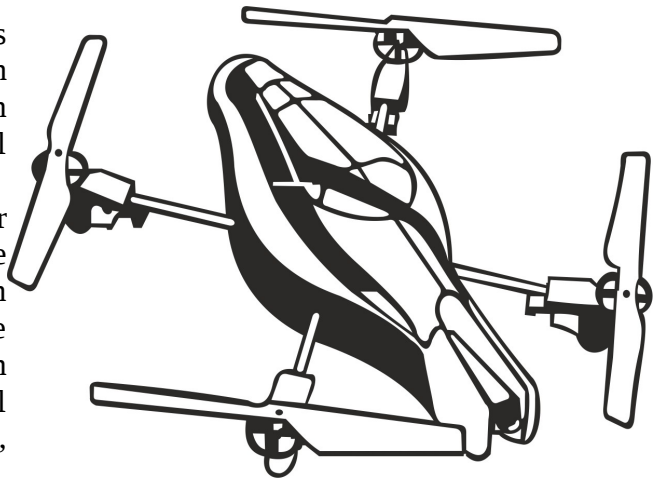
Documentation AHRS

1) Qu'est-ce qu'un AHRS ?



Un ahrs est un module qui récupère les données de trois capteurs : Un magnétomètre, un accéléromètre et un gyromètre, et calcule l'orientation d'un objet (ici le drone) par rapport au référentiel terrestre.

Le gyromètre fournit la vitesse angulaire sur chacun des axes du drone. En intégrant cette donnée on peut ainsi connaître en continue l'orientation angulaire du drone. Si elle était parfaite, cette donnée utilisée seule permettrait de connaître l'orientation exacte du drone à tout moment. Le capteur non idéal contient du bruit et un biais qui, une fois intégrée, produisent une erreur sur l'estimation de l'orientation.

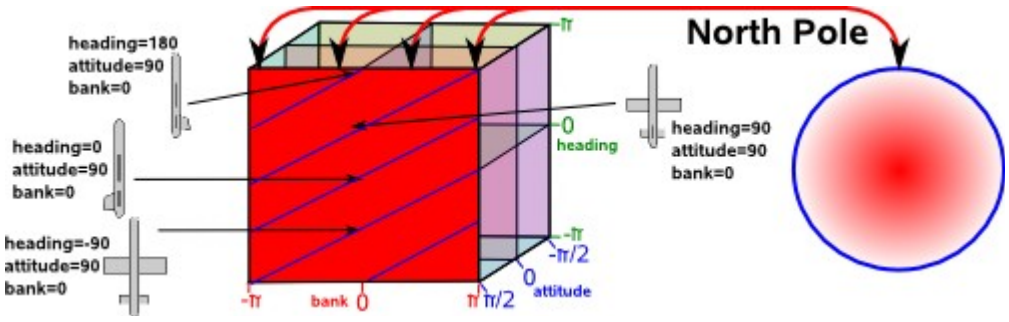
On utilise alors les deux autres capteurs.

L'accéléromètre permet de récupérer la valeur de l'accélération de l'objet sur les différents axes. La gravité est une accélération, l'accéléromètre permet donc de récupérer l'axe « Z » tant que l'objet n'accélère pas. Par la suit on considérera que les accélérations sont un bruit de haute fréquence et donc supprimées par un passe-bas.

Le magnétomètre quant à lui, permet de récupérer l'orientation du champ magnétique terrestre. Ici il permet de récupérer l'axe « X ». Avec ces deux axes

1.1) Angles d'Euler

Les angles d'Euler permettent de connaître l'orientation d'un objet après plusieurs rotations autour de ses axes. Ils sont plus faciles à interpréter pour l'homme, mais sont moins précis que les quaternions ou matrices de rotation. De plus ils nécessitent la prise en compte de singularités et certains cas ou plusieurs combinaisons d'angles d'Euler peuvent amener au même résultat.



Dans notre cas les angles d'Euler sont calculés par rapport à la matrice de rotation [3] et ne servent pas aux calculs intermédiaires.

1.2) Matrice de Rotation

La première approche pour calculer l'orientation du drone est d'utiliser les matrices de rotation. On utilise alors la méthode décrite dans le document [1]. L'intérêt de la matrice de rotation réside dans le fait qu'elle ne subisse pas de discontinuité sur ses différents éléments et permet de rapidement voir les différentes rotations ayant lieu sur le drone.

$$R_{\mathbf{x}}(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}, \qquad R_{\mathbf{y}}(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \qquad R_{\mathbf{z}}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Dans cette méthode on utilise le gyromètre pour mettre à jour la matrice de rotation :

$$d\Theta_i = w_i dt$$

$$R(t+dt) = R(t) \begin{bmatrix} 1 & -d\Theta_z & d\Theta_y \\ d\Theta_z & 1 & -d\Theta_x \\ -d\Theta_y & d\Theta_x & 1 \end{bmatrix}$$

Comme le gyromètre n'est pas parfait. On corrige la matrice ainsi obtenue à l'aide de l'accéléromètre et du magnétomètre. Pour cela on effectue le produit en croix des différents axes trouvés à l'aide de ces capteurs par rapport à ceux issus du gyromètre. L'erreur ainsi trouvée pour chaque direction est appliquée au gyromètre à l'aide d'un PI.

La plupart des autres calculs servent à normaliser les vecteurs et/ou les rendre orthogonaux entre eux.

Pour passer du référentiel du robot à celui de la terre et vice versa, il suffit de multiplier les coordonnées par la matrice R.
Ainsi $RX=X'$ pour passer X en référentiel du robot à X' en référentiel de la terre.
Et $R^{-1}X'=R^{T}X'=X$ pour effectuer l'inverse.

Lors du travail avec les matrices de rotation, les valeurs de Ki et Kp à utiliser sont respectivement 15 et 2.

Pour le moment, le calcul de la matrice de rotation n'est pas juste si le drone est mis à l'envers. Préferez les quaternions.

1.3) Quaternion

Le quaternion est un autre moyen de représenter les rotations dans l'espace. Il utilise une notation complexe dans l'espace.

$$Q = \begin{bmatrix} qw & qx & qy & qz \end{bmatrix}$$

La méthode de calcul utilisée est récupérée sur un AHRS open source [2] et utilise la méthode de calcul de Madgwick [5]. Les calculs sont similaires à ceux effectués pour la matrice de rotation. On utilise les deux vecteurs de référence pour construire le quaternion qui sert alors à calculer les coefficients à appliquer sur le gyromètre.

Lors du travail avec les quaternions, les valeurs de Ki et Kp à utiliser (trouvées de façon empirique) sont respectivement 5 et 100.

1.4) Filtre de Kalman

Le filtre de Kalman est un filtre passe-bas. Il est utilisé pour filtrer le signal issu du gyromètre. Pour l'accéléromètre et la magnétomètre, une moyenne glissante est utilisée comme filtre.

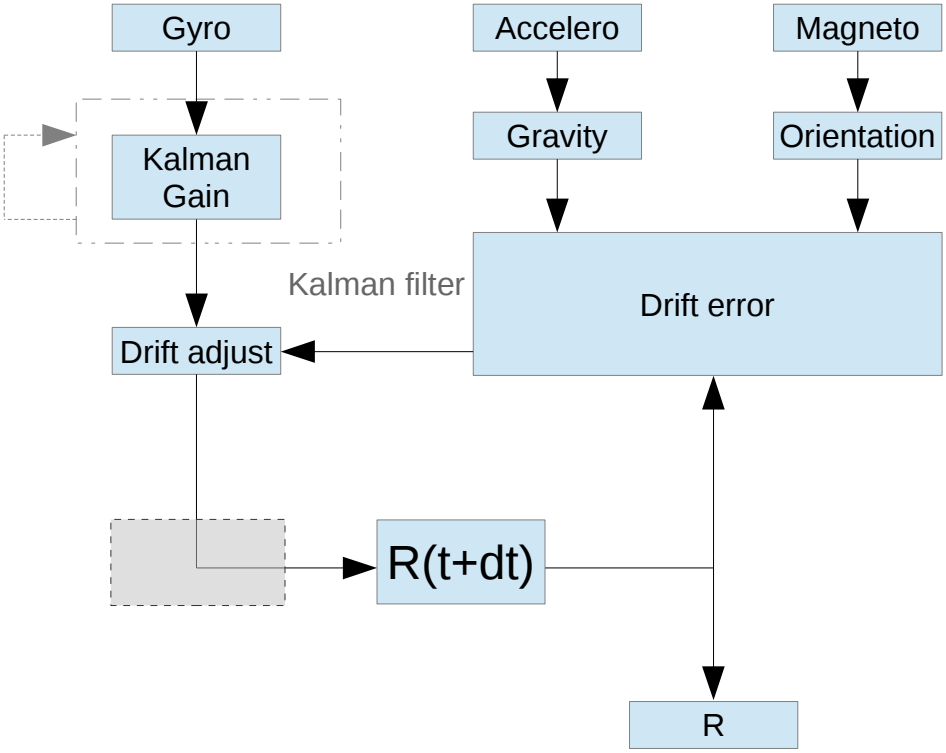


Schéma de l'algorithme utilisé pour le calcul de la matrice de rotation.

Appliqué seulement au gyromètre le filtre de Kalman n'est pas forcément plus utile q'un filtre passe-bas classique. Il n'empêche qu'il réduit grandement le bruit même après la correction de la dérive. En pointillé se trouve l'endroit où le filtre pourrait être appliqué pour une meilleur efficacité.

Ci dessous, le calcul matriciel nécessaire pour appliquer une itération du filtre de Kalman à l'un des capteur du gyromètre. [4]

$$
\begin{aligned}
&Prédiction \\
&\widehat{X}_k^+ = A\widehat{X}_k \\
&P_k^+ = A\widehat{P}_k A^T + Q \\
&Mise\ à\ jour \\
&K_{k+1} = P_k^+ H^T \times (R_{k+1} + H P_k^+ H^T) \\
&P_{k+1}^+ = A\widehat{P}_k A^T + Q \\
&\widehat{X}_{k+1}^+ = \widehat{X}_k^+ + K_{k+1}(y_{k+1} - H\widehat{X}_k^+)
\end{aligned}
$$

$$Y = y_{k+1} - H\widehat{X}_k^+$$

$$X = \begin{bmatrix} dw \\ w \end{bmatrix} \qquad A = \begin{bmatrix} 1 & 0 \\ dt & 1 \end{bmatrix} \qquad H = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} * & 0 \\ 0 & * \end{bmatrix} \qquad P = \begin{bmatrix} * & 0 \\ 0 & * \end{bmatrix}$$

Pour appliquer le filtre de Kalman, il ne reste plus qu'à calculer les coefficients de la diagonale de Q et la valeur de R. Q est la matrice de covariance entre les différents capteurs ou valeur mesurées. Ici on ne dispose pas de capteurs mesurant exactement la même grandeur, on peut donc partir du principe qu'il n'y a pas de covariance entre les diférents capteur. R est la variance du bruit des capteurs. Ici chaque capteur est considéré comme identique, une étude du bruit permettrait de trouver la valeur la plus juste.

P est quand à lui mis à jour par le filtre à chaque nouvelle itération.

2) Mise en place

2.1) Code

Pour tous les tests nous avons utilisés le makefile présent dans « bibrone/tests ». L'utiliser nécessite de modifier le chemin vers le g++ du cross compilateur.

Ex : CPP=/home/user/CodeSourcery/bin/arm-none-linux-gnueabi-g++

Il suffit ensuite de lancer un terminal et de se placer dans le répertoire tests. Puis créer le fichier .elf avec la commande make (ou make -B, si vous avez modifié des fichiers ailleurs).

Connectez-vous ensuite au drone, et utilisez ftp 192.168.1.1 pour placer le fichier sur le drone et telnet 192.168.1.1 pour vous connecter à ce dernier. Lors de la première connexion après un redemarrage, n'oubliez pas de tuer le programme ardrone (ps et kill n°tâche ayant pour nom programme.elf) qui empêche le fonctionnement de l'ahrs. Le fichier .elf placé en ftp se trouve dans le répertoire « data/video ».

Si vous modifiez le fichier ahrs.h, il doit être identique dans « bibrone/src » et « bibrone/include/bibrone ».

2.2) AHRS

Créez un nouvel AHRS en initialisant une variable de la classe ahrs. Si vous le souhaitez vous pouvez redéfinir le mode de fonctionnement (Quaternion, Matrice de rotation, etc.). Lancer ensuite l'AHRS en utilisant la fonction Start(), et arrêtez l'AHRS avec la fonction Stop(). Vous pouvez aussi utiliser l'AHRS « à la main » à l'aide de toutes les autres fonctions, notez qu'il vous faudra alors envoyer les données du magnétomètre et de l'accéléromètre avec les bons signes.

Par défaut l'AHRS utilise les quaternions et kalman.
Kp = 15, Ki = 100, Ka=1, Km=0.2, Kt=1
dt = 0.01
Kal = 1

2.3) Exemple d'utilisation

Voilà une mise en place d'un AHRS mettant à jour les données toutes les 20ms. Utilisant les quaternion mais pas Kalman.

```
ahrs test;
test.Initialize();
test.Start(0.02);
…
test.GetEuler();
```

Un autre exemple, pour utiliser la matrice de rotation pour les calculs :

```
ahrs test;
test.Initialize();
test.SetQuaternion(false);
test.Set(0.01,2,15);//Nécessaire (coef PI diff)
test.Start();
…
```

Attention, il n'est pour le moment pas possible de lancer deux ahrs distinct en utilisant Start(), de même modifier les options de l'ahrs en cours de fonctionnement peut causer des erreurs de calculs s'il est en fonctionnement non threadé, de segmentation s'l avait été lancé avec Start(). Ainsi choisissez les options que vous voulez au début (ou gardez celles par défaut) puis n'utilisez plus que les getters.

4) Méthodes

4.1) Publiques

```
void Start(float dt, bool raw);
void Start(float dt);
void Start();
void Stop();
```

Lance l'AHRS en tâche de fond (autre thread). Il s'éxécute alors toutes les 10ms par défaut, ou si spécifié toutes les dt (en secondes, supérieur à 2ms). Il s'occupe alors de récupérer les données et vous n'avez qu'à utiliser les Getters. Lorsque raw est mis à vrai, un fichier samples.csv est remplis des données brutes (gx,gy,gz,ax,ay,az,mx,my,mz).

```
void Etalonnage();
```

Lance une routine pour étalonner le magnétomètre (et d'autres informations) stockées dans le fichier « config.txt ». Cette routine permet aux informations du magnétomètre d'être étalonnées et est donc obligatoire à chaque changement de zone géographique. L'étalonnage est stable dans le temps à hauteur de quelques mois.

Une fois la routine lancée plusieurs étapes sont demandées : Tenir le drone à l'horizontale et lui faire faire une rotation complète. Le pencher à la verticale et faire de même. Si les barres d'avancement ne sont pas totalement remplies à la fin de l'étallonnage c'est qu'il n'a pas été bien effectué.

```
void Initialize();
```

Permet l'initialisation de l'ahrs et la lecture des informations contenues dans config.txt.

```
void Update(float mx, float my, float mz, float ax, float ay, float az,
float gp, float gq, float gr);
void UpdateMagnetometer(float mx, float my, float mz);
void UpdateAccelerometer(float ax, float ay, float az);
void UpdateGyrometer(float gp, float gq, float gr);
void Update();
```

Ces fonctions permettent de récupérer les informations données par les différents capteurs. Notez qu'elles font automatiquement une moyenne glissante. Alors que Update met aussi à jour l'orientation, les trois fonction UpdateMagnetometer, UpdateAccelerometer et UpdateGyrometer ne mettent pas à jour l'orientation. Cela permet de récupérer les informations des capteurs plus souvent que le calcul de l'orientation.

Un utilisation rapide ne requière que le premier appel, une utilisation plus propre est d'appeler à des périodes différentes la dernière et les trois précédentes. La méthode Start() permet de le faire automatiquement.

```
void Set(float dt, float ki, float kp, float ka, float km, float kt);
void Set(float dt, float ki, float kp);
void Set(float dt);
```

Ces fonctions permettent de définir les différents facteurs utilisés pour le calcul de l'orientation.

- Dt : Temps entre chaque mise à jour (entre chaque appel à Udpate()) en secondes,
- Ki : Coefficient de l'intégrateur pour la correction de l'erreur statique du gyromètre et orientation,
- Kp : Coefficient du proportionnel pour la correction de l'orientation,
- Ka /Km : Fiabilité accordée à l'accéléromètre et magnétomètre entre eux, par défaut 1 et 0.2,
- Kt : Coefficient appliqué à la correction finale.

```
void SetKalman(float kal);
void SetKalman(bool);
```
Permet de spécifier l'utilisation ou non de kalman pour lisser l'orientation. La première contraint le système à l'utiliser et permet de spécifier à quelle proportion la valeur trouvée par ce filtre est appliqué sur la mesure. A 0 la valeur est identique à celle de la mesure, à 1 seule la valeur de kalman est prise en compte. Par défaut 1 et vrai.

```
void SetQuaternion(bool);
```
Permet de spécifier l'utilisation de quaternion plutôt que de matrice de rotation. Par défaut vrai.

```
std::vector< std::vector<float> > GetRotationMatrix();
```
Retourne la matrice de rotation courante.

```
std::vector<float> GetQuaternion();
```
Retourne le quaternion courant.

```
std::vector<float> GetEuler();
```
Retourne les angles d'euler dans l'ordre : Rotation autour de X, Z et Y ;

4.2) Privées

```
std::vector<float> cross_product(std::vector<float>, std::vector<float>);
```
Effectue le produit en croix entre deux vecteurs de taille 3.

```
std::vector< std::vector<float> > multiply(std::vector< std::vector<float>
>,std::vector< std::vector<float> >);
```
Multiplie deux matrices de taille 3.

```
float pow_sum(std::vector<float>);
```
Calcul la somme des carré de chaque élément d'un vecteur de taille 3.

```
std::vector<float> kalman_update(std::vector<float>);
```
Met à jour le filtre de Kalman.

```
std::vector<float> MatToQua(std::vector< std::vector<float> >);
```
Transforme une matrice en Quaternion.

Sources :

[1] William Premerlani and Paul Bizard. "Direction Cosine Matrix IMU: Theory." *Gentlenav*. 17 May 2009. Web. <http://gentlenav.googlecode.com/files/DCMDraft2.pdf>.

[2] "Open Source AHRS with X-IMU | X-io Technologies." *Xio Technologies*. Web. <http://www.x-io.co.uk/open-source-ahrs-with-x-imu/>.

[3] Baker, Martin J. "Maths - Rotation Conversions." *Rotations Conversions*. Web. <http://www.euclideanspace.com/maths/geometry/rotations/conversions/index.htm>.

[4] HoshiKata "Kalman Filtering Demo" Wen. Web. 2 Aug 2012. <http://www.codeproject.com/Articles/326657/KalmanDemo>.

[5] Madgwick, Sebastian O.H. "Estimation of IMU and MARG Orientation Using a Gradient Descent Algorithm." *IEEE Xplore*. 01 July 2011. Web. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5975346&tag=1>.