Le cœur de l'algorithme est codé dans la classe Etat, qui code l'état instantané du satellite. La méthode actualiser est la plus importante : c'est elle qui lance l'ensemble des calculs permettant de déterminer l'attitude du satellite, et de calculer l'intensité à fournir dans les bobines. Elle réactualise donc l'ensemble des variables d'état et notamment le quaternion q.

L'algorithme de détermination de l'attitude et de contrôle est décrit dans le rapport des Mines (rapport-ADCS-mines-actuel.pdf). Il est basé sur une équation différentielle appliquée au quaternion représentant l'attitude du satellite et au biais du gyromètre (un vecteur 3d). Il se résume par les formules suivantes :

$$\begin{cases} \dot{\hat{q}} = \frac{1}{2} * \hat{q} * \left(\omega_{mes} - \hat{b} + k_p \Omega\right) + k(1 - \| \hat{q} \|^2)\hat{q} \\ \dot{\hat{b}} = -k_i \Omega \end{cases}$$

où $k, k_i, k_p > 0, \omega_{mes}$ est le résultat de la mesure du gyroscope et

$$\Omega = \frac{k_B}{||\vec{B}_i^{table}|| \cdot ||\vec{B}_b^{mes}||} \vec{B}_b^{mes} \times (\hat{q}^{-1} \vec{B}_i^{table} \hat{q}) + \frac{k_S}{||\vec{S}_i|| \cdot ||\vec{S}_b^{mes}||} \vec{S}_b^{mes} \times (\hat{q}^{-1} \vec{S}_i \hat{q})$$

avec $\vec{B}_b^{mes}$ (resp. $\vec{S}_b^{mes}$) le résultat de la mesure du magnétomètre (resp. capteurs solaires) et $\vec{B}_i^{table}$ est la valeur du champ magnétique dans $\mathcal{F}_i$ lue dans les tables embarquées.

Ce qui précède est un filtre non linéaire. Il permet d'estimer le quaternion q à chaque instant en fonction non seulement des signaux des capteurs, mais aussi de la valeur calculée précédemment. Cela permet de lisser le signal, c'est-à-dire d'éliminer les variations brusques. Plus précisément, q désigne le quaternion représentant l'attitude du satellite, b le biais du gyromètre, ω_{mes} la valeur de vecteur rotati on mesurée par le gyro, les variables en k sont des gains statiques, et enfin les valeurs en B et S représentent les vecteurs champ magnétique et rayon solaire dans différents référentiels. On implémente le tout par un schéma d'Euler explicite (ie. $q = q + \dot{q} * dt$).
Enfin la partie contrôle est réalisée par la formule ;

$$\vec{m} = -\frac{\vec{B}_b}{||\vec{B}_b||^2} \times \left(k_D\ \epsilon(\omega) + k_P\ \epsilon(q)\right)$$

ou m est le moment magnétique à génerer, $\varepsilon(\omega)$ est $\omega_{mes} - \omega_{ref} - b$ et $\varepsilon(q)$ est la partie vectorielle du quaternion q

Ce sont ces calculs qui sont implémentés dans etat.cpp, à la fin du fichier, dans la méthode actualiser.

Description de la méthode actualiser :
1ère phase : récupération de données.
Il faut coder les lignes permettant de faire le lien entre les capteurs et l'algo : les fonctions getLat() et getLong() pour le GPS, getOmega() pour le gyro (en rad/s, j'en suis quasi sûr mais pas à 100%), getTLE() pour mettre a jour les données sur l'orbite des TLE (si j'ai bien compris, seule l'anomalie moyenne « nu » doit changer, les autres paramètres ne dépendant que de l'orbite. Mais celle-ci change peut-être au cours du temps ? ; les paramètres sont renvoyés en degrés comme sur les véritables TLE), getChampM() pour le magnétomètre, getVecSol() pour

les capteurs solaires (il reste à initialiser le vecteur vmes avec les valeurs des 6 capteurs solaires).

2ème phase : on applique le filtre décrit par les premières équations. Un paramètre important est le paramètre dt qui désigne le temps écoulé entre chaque étape de calcul (c'est la méthode des éléments finis appliquée à l'équation différentielle). Avec 1s on est bien, mais dans l'idéal il faudrait que ce soit le vrai temps écoulé depuis le précédent calcul !

Explications sur les quaternions

Un quaternion représente une rotation 3D, au même titre que 3 angles d'Euler ou qu'une matrice de rotation. Il se présente sous la forme suivante dans nos programmes : (sin(α/2)x, sin(α/2)y, sin(α/2)z, cos(α/2)) et représente alors la rotation d'angle α autour de l'axe (x,y,z). On peut le voir comme un vecteur 4D de norme 1. On peut les multiplier entre eux (non commutatifs) et les conjuguer à un vecteur 3D (opération q.conjug(v)) ce qui revient à multiplier des matrices de rotation entre elles et à conjuguer une matrice de rotation à un vecteur. Pour plus d'explications cf Wikipédia « Quaternions et rotation dans l'espace ». Attention à l'ordre : la partie réelle est la quatrième composante du quaternion dans ce programme.

Paramètres orbitaux