# RoBoNav - Régulation de la position

Jean Fruitet



## Positionnement GPS

La conservation d'une position attribuée par un point GPS est très problématique dans ce projet de bouées de régates autonomes.
Si les bouées sont positionnées par pilotage manuel, à vue, il n'est pas très facile de décider si chacune est correctement placée par rapport aux autres bouées du parcours.
On souhaiterait donc plutôt leur assigner des positions GPS calculées au préalables en fonction de la géométrie du plan d'eau et de la direction du vent.

Et c'est là que les difficultés commencent.

1. La précision des points GPS est au mieux d'un mètre, pour les applications grand public. C'est insuffisant pour notre projet. L'utilisation d'un système d'acquisition différentiel (DGNSS) de type RTK est peu réaliste car assurément coûteuse ; elle nécessite des capteurs GSM ad hoc et une connexion GSM (téléphonie mobile) par bouée, peu compatible avec le cahier des charges.
2. La position du signal GPS est soumise à des fluctuations dans la zone de précision qui induisent la bouée en erreur. Elle "croit" se déplacer alors que c'est la position délivrée qui oscille.
   Pour contrecarrer cette dérive une mesure simple serait peut-être de moyenner les *n* dernières mesures pour obtenir la position courante ; on pourrait aussi tester d'autres fonctions statistiques comme la médiane de la distribution.
   Il faut déterminer la fréquence d'échantillonnage pour éviter de saturer le système ; a priori une mesure moyenne par seconde, échantillonnée au 10ème de seconde me paraît suffisante (10 mesures par seconde dont on fait la moyenne) pour une bouée.
3. Une approche complémentaire serait d'instaurer une manière de DGNSS du pauvre en comparant en temps réel la position GPS délivrée par les satellites avec celle d'une position à terre repérée précisément sur Google Maps, par exemple un amer ou le

ponton de mise à l'eau, s'il se voit sur les photos aériennes.
A chaque seconde un vecteur de correction de position, identique pour toutes les bouées (vu les courtes distances en voile RC), peut alors être envoyé par WiFi ce qui les positionne au décimètre près (à vérifier).

Si on arrive à une précision de l'ordre du décimètre dans la détermination de la position d'une bouée, on peut envisager d'implanter les commandes automatiques d'assignation d'une position (puis de conservation de ladite position), malgré la dérive due au vent ou au courant.

En effet, cette dérive peut être calculée en comparant deux positions GPS "redressées" ; cela fournit une distance (en mètre), une direction moyenne (en radian ou en degré) et une vitesse moyenne (en mètre / seconde) pour la dérive par unité de temps (en seconde).

A partir de la dérive il faut actionner les moteurs pour aller (ou revenir) à la position assignée.

# Régulation

La régulation consiste à assigner à chaque instant (chaque seconde) une commande sur les effecteurs (ici les deux moteurs de la bouée) pour minimiser l'erreur entre la position assignée (la consigne) et la position mesurée.
C'est un processus automatique qui ne doit réclamer aucune intervention humaine.
Plusieurs algorithmes sont envisageables, la régulation PID (proportionnelle, intégrale, dérivée) étant probablement la mieux adaptée au cas d'espèce des bouées, qui sont de la même famille que les "rovers" de PX4 et ArduPilot.

On peut envisager d'implanter trois types de régulation :

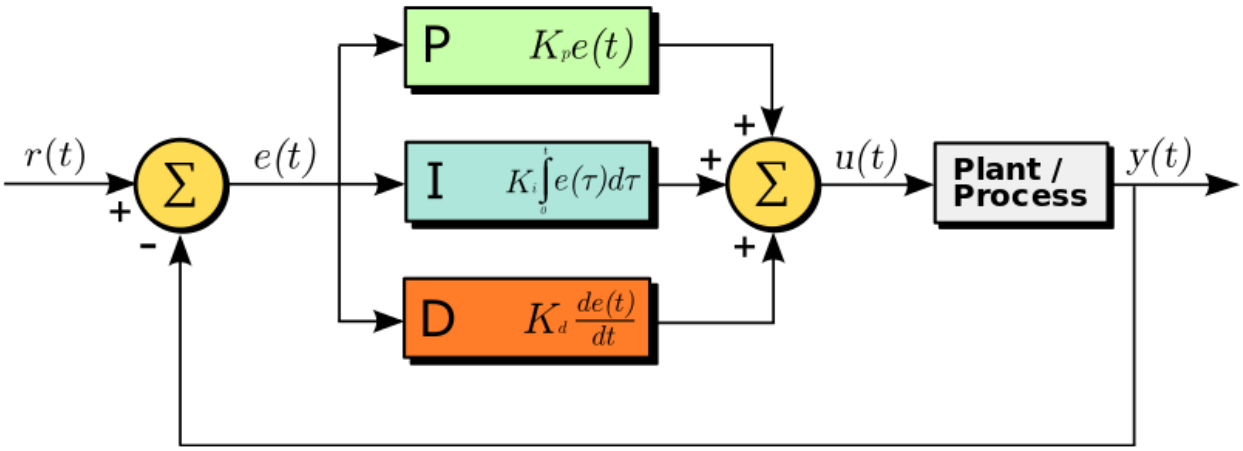
- Dans la phase de déplacement
    - une régulation de la direction de déplacement. En effet, si les deux moteurs ne tournent pas à la même vitesse pour la même impulsion PWM (ce qui assez fréquent), la bouée tourne sur elle-même au lieu d'aller en ligne droite. La régulation porte sur les variations du compas par rapport au cap assigné ; elle se traduit par des signaux différents pour chaque moteur.
    - une régulation de l'approche de la position GPS assignée. La bouée doit accélérer en phase de déplacement tant qu'elle est loin de l'objectif, puis ralentir et s'immobiliser sans dépasser celui-ci. La mesure porte sur la distance au but.
- Dans la phase d'immobilisation la régulation porte sur la conservation du cap face au vent en prenant en compte le compas de bord et vise à annuler les micros déplacements mesurés par les accéléromètres électroniques placés sur le circuit intégré du contrôleur de navigation ;
  A vrai dire **si on fabrique correctement la bouée et que la direction du vent est stable**, la bouée devrait se tenir naturellement face au vent ce qui rend l'utilisation du compas inutile. Dans la pratique, vu sa forme cylindrique elle a tendance à pivoter sur elle-même et le compas joue un rôle décisif.
  On peut déduire la direction de déplacement de la moyenne des valeurs fournies par le

compas sur une ou deux minutes, puis chercher à se rapprocher au mieux de cette direction.
Concernant les accéléromètres, ils sont extrêmement précis puisqu'ils permettent de positionner un drone volant en vol stationnaire. A priori nous n'avons pas à tenir compte des accélérations verticales dues aux oscillations de la houle, mais ce sera à vérifier car les moteurs aussi oscillent avec la bouée ; peut-être peut-on contrecarrer ces oscillations en augmentant l'inertie de la bouée en la lestant.
Quoi qu'il en soit, ce sont les accéléromètres qui permettent de conserver une position très précise dans l'espace, une fois que la bouée a atteint sa position assignée.

En conclusion l'implantation de différentes formes de régulation est un enjeu majeur du projet.

## Algorithme PID

Soit un dispositif à réguler.
On a une mesure physique en entrée (input), ***r(t)***, par exemple la distance à parcourir pour atteindre la position assignée.
On cherche à obtenir une commande à appliquer au système, soit ***u(t)***, qui peut être un signal de commande des moteurs.
Après application de cette commande, on récupère une nouvelle valeur de la grandeur à réguler, soit ***y(t)***, qui est réinjectée dans le système




https://www.teachmemicro.com/arduino-pid-control-tutorial/

Il y a de nombreuses formulations de la régulation PID. La plus classique est

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt}$$

avec ***u(t)*** (output), commande à appliquer,
***e(t)***, erreur entre la consigne assignée et la réalisation mesurée à l'instant ***t***

***Kp***, ***Ki***, ***Kd***, coefficients respectivement de proportionnalité, d'intégration et de différenciation, à appliquer aux différents termes de l'équation pour que le système converge suffisamment rapidement et sans oscillations intempestives.
Les valeurs respectives de ces coefficients sont assez délicates à déterminer, car elles dépendent de la nature physique du système à réguler et de sa temporalité.

Pour les applications qui nous intéressent, la commande est appliquée aux moteurs, l'erreur est soit une erreur de position, soit de cap, soit d'accélération, ou n'importe quelle combinaison des trois !
La temporalité est de l'ordre de la seconde pour la position, du 1/10ème de seconde pour le cap et du 1/100ème de seconde pour l'accélération.

En pratique l'algorithme de PID va consister en une suite répétée d'opérations arithmétiques :

```
Comd_int = 0   // Somme des erreurs sur la période considérée

Tous les t, faire :  // en centièmes de secondes

     erreur = consigne - mesure;  // Nombre décimal positif ou
négatif

     Comd_int = Comd_int + Ki*erreur;  // Terme intégral de
l'équation

     variation_erreur = erreur - erreur_précédente; // Différence

     commande = Kp * erreur + Comd_int + Kd * variation_erreur;

     erreur_précédente = erreur
```

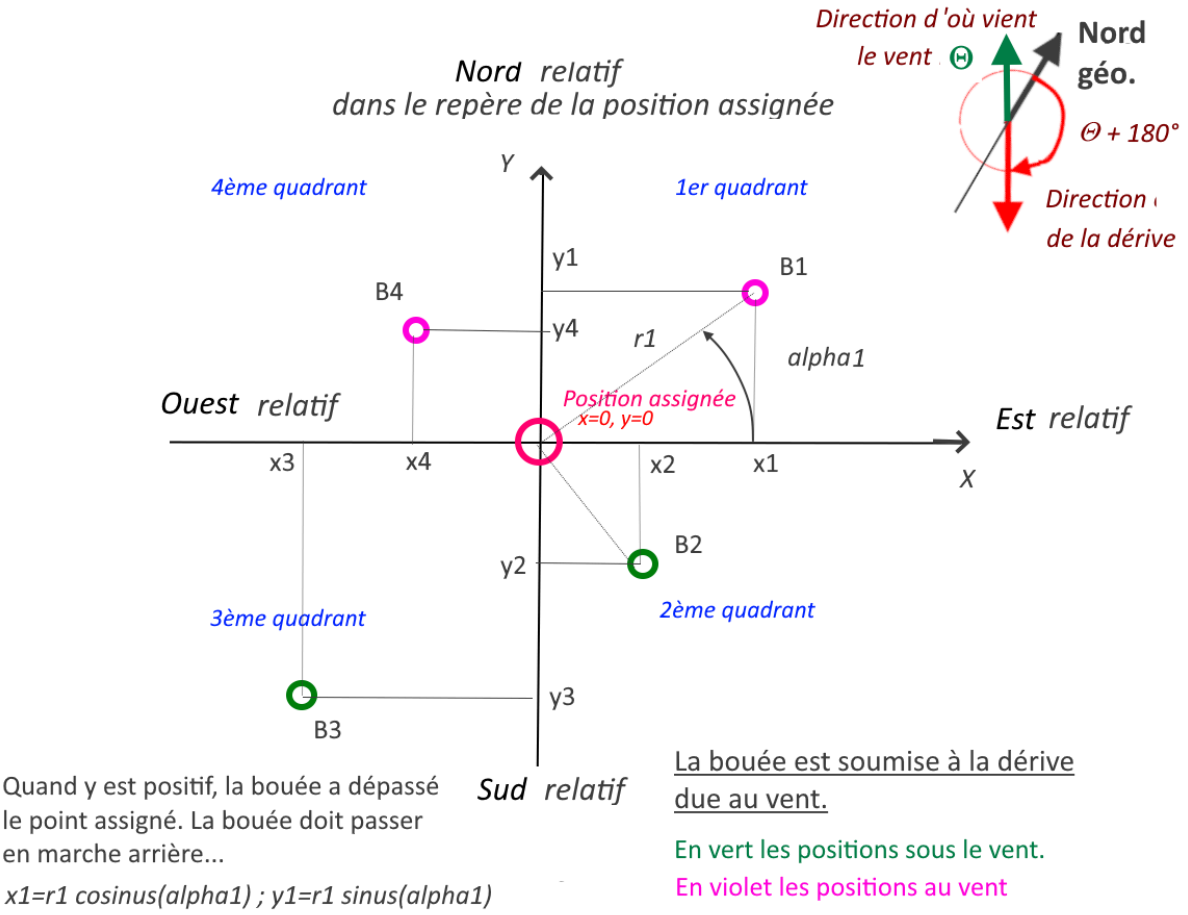
La grosse difficulté est de déterminer les valeurs des coefficients ***K***.

Il faudra aussi modifier l'implantation si le délai ***t*** entre chaque itération n'est pas régulier.

## Nature de l'erreur

S'il s'agit d'une erreur de position, c'est par rapport à un point GPS dont la précision est assurément sujette à caution.
On cherche à l'améliorer en implantant un **algorithme statistique** sur la période de temps considérée. Mais il serait illusoire de chercher à obtenir une précision meilleure que le décimètre.
En prenant en compte la direction du point à atteindre (fournie par comparaison entre la position courante et le point à atteindre) on peut transformer l'erreur de position en une distance + une orientation relative (le compas de bord indique comment la bouée est orientée par rapport à sa direction de déplacement).

Dans un repère centré sur la position à atteindre, quand la position passe du second ou du troisième quadrant au premier ou au quatrième quadrant (modulo une composante de quelques degrés à déterminer par essai-erreur pour limiter les oscillations), la bouée a dépassé son objectif et l'erreur de position est négative, sinon elle est positive.



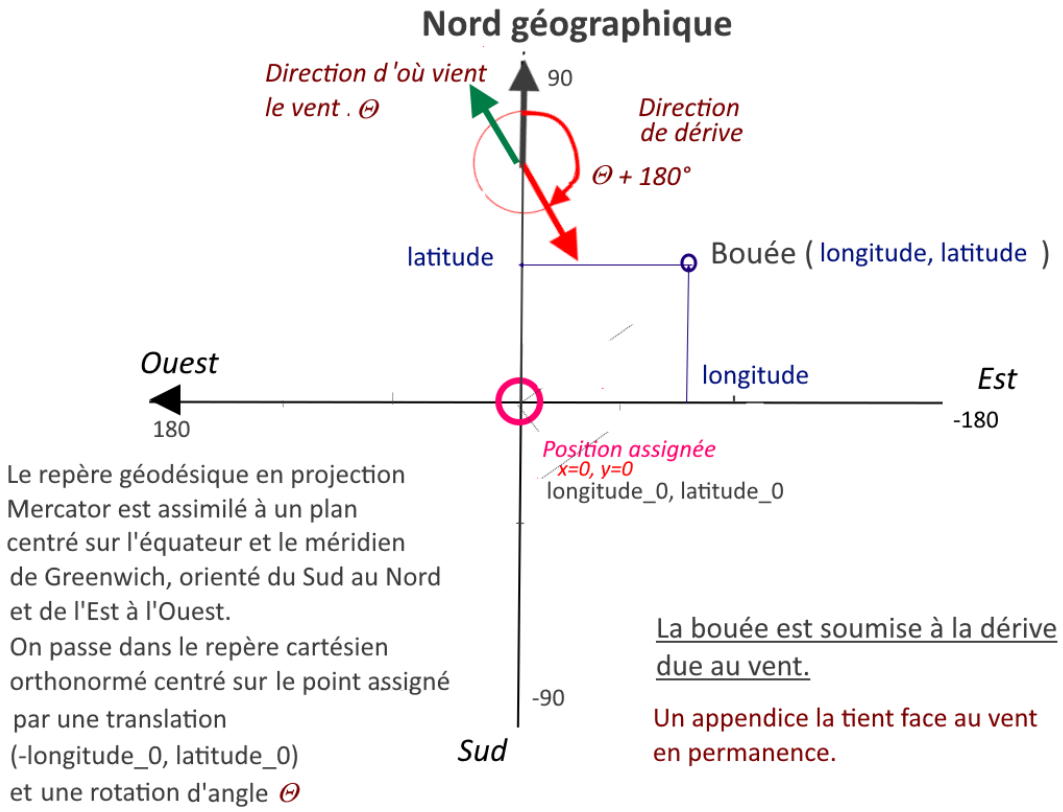

Dans ce repère relatif
$r^2 = (x\_bouee - x\_assignee)^2 + (y\_bouee - y\_assignee)^2$
$\alpha = Arctangente((y\_bouee - y\_assignee) / (x\_bouee - x\_assignee))$

avec $x\_assignee=0$ et $y\_assignee=0$

Dans une zone de quelques centimètres autour de la position assignée, la position de la bouée est indéterminée.

## Changement de repère

On passe du repère géographique absolu des points GPS au repère relatif à la direction du vent centré sur le point à atteindre par une rotation et une translation élémentaires.
Vu les dimensions très réduites des plans d'eau où se pratique la voile radiocommandée, on peut se contenter de coordonnées cartésiennes dans un repère euclidien, sans faire appel aux coordonnées sphériques des repères géodésiques.




Dans le repère géodésique les coordonnées d'un point s'expriment en latitude et longitude. Les latitudes vont de -90° (Sud) à 90° (Nord) ; les longitudes de -180° (Est) à 180° (Ouest).

Soient ***lat_0*** la latitude et ***lon_0*** la longitude du point à atteindre (point assigné) et ***lat*** la latitude et ***lon*** la longitude de la bouée à l'instant *t*.

**Translation pour centrer le repère sur la position assignée** :
**T[-(*lon* - *lon_0)* , (*lat* - *lat_0)*]**

Un point de coordonnées (lat, lon) a par translation les coordonnées
$A_x$ = *lon_0 - lon,*
$A_y$= *lat - lat_0*

**Rotation pour se ramener face à la direction du vent**[1] :

L'opposé de la direction de dérive est calculé en moyennant la dérive sur quelques minutes.
Soit $\Theta$ la direction opposée à la dérive (c'est grosso modo la direction d'où vient le vent), en degré de 0° à 360° [0, 2$\pi$) dans le sens horaire.

[1] En présence de courant, la force du courant et la force du vent s'additionnent vectoriellement, ce qui produit une dérive à laquelle le système bouée / moteurs doit s'opposer. En pratique les régates radiocommandées se pratiquent en l'absence de courant…

0° = Nord, 90° = Est, 180° = Sud, 270° = Ouest.

Matrice de rotation d'angle $\Theta$ dans un repère orthonormé :

$$\begin{pmatrix} A'_x \\ A'_y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} A_x \\ A_y \end{pmatrix}$$

avec *Ax = lat - lat_0*, *Ay = lon_0 - lon*, (en raison de l'orientation Est - Ouest du repère GPS), on obtient

*A'x = (lat - lat_0) * cos(Θ) - (lon_0 - lon) * sin(Θ)*
*A'y = (lat - lat_0) * sin(Θ) + (lon_0 - lon) * cos(Θ)*

Si *A'y > 0* il faut reculer, sinon il faut avancer.

Le carré de la distance à franchir est *A'x * A'x + A'y * A'y*

On repasse dans les coordonnées géographiques par la rotation inverse et la translation opposée.

## Implantation avec Arduino

On peut soit utiliser un circuit électronique dédié, soit développer son propre code en s'appuyant sur la bibliothèque de Brett Beauregard.

## Conclusion

La régulation est au cœur du projet RoBoNav.

Les deux premières années ont été consacrées au choix des composants, à la fabrication de la bouée et du circuit embarqué de navigation, au pilotage manuel et aux tests.

Les prochaines réalisations devront porter sur la stabilisation en position et la gestion des erreurs (failsafe).

## Liens

Vulgarisation de la régulation PID
https://energieplus-lesite.be/techniques/chauffage10/principes-de-regulation-p-pi-pid/#La_regulation_Proportionnelle_-_Integrale_PI

Exemple de réalisation avec carte Arduino :
https://www.arrow.com/en/research-and-events/articles/pid-controller-basics-and-tutorial-pid-implementation-in-arduino

Tutoriel PID pour carte Arduino (en anglais) :
https://www.teachmemicro.com/arduino-pid-control-tutorial/

Une implantation C++ de l'algorithme PID pour Arduino avec des exemples :
https://playground.arduino.cc/Code/PIDLibrary/
https://github.com/br3ttb/Arduino-PID-Library

Une implantation utilisant des accéléromètres, dont on peut fortement s'inspirer :
https://hackaday.io/project/21920-ambot
https://github.com/rppelayo/arduino-self-balancing-robot

Une palanquée de math. pour 15$ :
https://www.eolss.net/ebooks/Sample%20Chapters/C18/E6-43-03-03.pdf

Forum de Ferdinand Piette avec des remarques concernant la mise en oeuvre :
http://www.ferdinandpiette.com/blog/2011/08/implementer-un-pid-sans-faire-de-calculs/

Nicolas dit :
3 septembre 2018 à 23:37

(...) Je voudrais ajouter une remarque.
L'implémentation

```
Tous les x millisecondes, faire :
      erreur = consigne - mesure;
      somme_erreurs += erreur;
      variation_erreur = erreur - erreur_précédente;
      commande = Kp * erreur + Ki * somme_erreurs + Kd *
variation_erreur;
      erreur_précédente = erreur
```

pose le problème suivant :
Une fois que le terme intégral s'est chargé d'annuler l'erreur et que l'on est en régime établi, on a :

```
      erreur = 0 et  variation_erreur = 0
```

Donc

```
      commande = Ki*somme_erreurs
```

Si la consigne de vitesse du moteur est non-nulle, la commande sera non nulle également.

Si l'on est en phase de réglage des gains du correcteur, on aura une mauvaise surprise...

Par exemple, si en régime établi, la commande vaut 5 V et que l'on passe `Ki` de 1 à 1.5, la commande va s'envoler à 7.5V. Alors que le moteur tournait à la bonne vitesse, il va se mettre à accélérer !

Pour résoudre cela il faut implémenter :

```
Commande_intégrale = 0
Tous les x millisecondes, faire :
     erreur = consigne - mesure;
     Commande_intégrale = Commande_intégrale + Ki*erreur;
     variation_erreur = erreur - erreur_précédente;
     commande = Kp * erreur + Commande_intégrale + Kd *
variation_erreur;

     erreur_précédente = erreur
```

Si l'on reprend l'exemple précédent, le moteur tourne à la vitesse de consigne (`erreur = 0`, `variation_erreur = 0`) et `commande = 5V = Commande_intégrale`.

On est en train d'ajuster `Ki` et on le fait passer de 1 à 1.5

Alors `Commande_intégrale = Commande_intégrale + Ki*erreur = Commande_intégrale = 5V`

Le moteur continue de tourner à la valeur de consigne et on procède alors tranquillement à un changement de consigne pour observer l'effet du nouveau réglage...