

ODOMETRIE par l'équipe RCVA

L'odométrie, définie au sens large, consiste à évaluer la position d'un mobile par divers moyens internes ou externes. Les moyens internes permettent un repérage autonome sans aucune liaison avec l'extérieur à la différence des moyens externes qui consistent à communiquer avec l'extérieur (balises au sol, satellites ...). En ce limitant au cas d'un robot Eurobot et en supposant que la liaison avec le sol ne constitue pas une aide externe, on peut classer quelques procédés odométriques, la liste n'étant pas exhaustive.

Odométrie interne :

-Utilisation de 2 roues codeuses.

Elles mesurent les déplacements gauche et droite du robot.

Qualité :

Simple d'emploi, non brouillable.

Défaut :

Ne détecte pas les déplacements latéraux donc inutilisable pour les robots à déplacements holonomes.

Existence d'une dérive sur le positionnement liée au fait que l'algorithme procède par intégration, donc accumulation des erreurs.

-Utilisation d'une souris.

Qualité :

Simple et utilisable pour les mouvements holonomes.

Défaut :

Manque de précision si on se réfère aux quelques expériences exposées sur le forum, car grande dépendance à la nature et l'état du terrain.

-Navigation inertielle :

Utilisation d'un système de capteurs 3 axes constitué de 3 accéléromètres et de 3 gyromètres. Cette technique utilisée en aéronautique pourrait devenir abordable (financièrement) depuis l'apparition de la nouvelle technologie **MEMS** (Système Micro Electro Mécanique).

Qualité :

Le système idéal à autonomie totale donc non brouillable et qui prend en compte tout type de déplacement et toute forme de terrain. (Bosselé, pentu ...)

Défaut :

La complexité des calculs qui demande un processeur musclé.

Les dérives des capteurs qui conduisent à une incertitude sur le positionnement dont l'ordre de grandeur reste à évaluer.

(Quelques liens utiles sont donnés en fin d'article pour ceux qui seraient tentés par l'aventure)

Odométrie externe :

Utilisation de balises disposées en bordure de terrain.

Qualité :

Séduisant à priori car prend en compte les déplacements holonomes.

Pas de dérive dans le temps car pas d'intégration dans le principe.

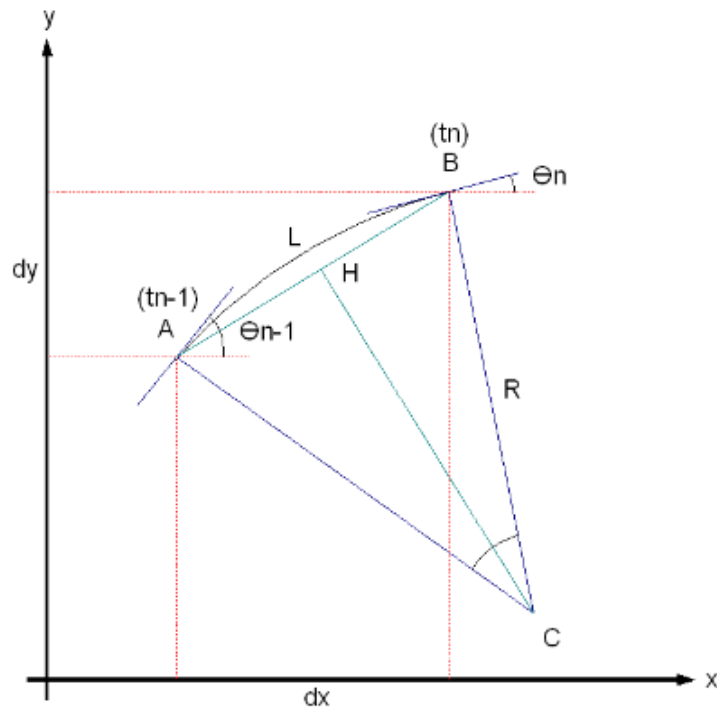
Défaut :

Brouillable comme tout système externe.

Les temps de mesure souvent excessifs et incompatibles avec des déplacements grande vitesse. (Un lien est donné en fin d'article sur le nouveau système de balises Microb-Technologie).

On peut aussi imaginer une odométrie qui associe les moyens externes et internes afin de profiter des qualités de chacun sans oublier des recalages très simples de mise en œuvre sur bordures ou obstacles à position fixe et connue.

L'article qui suit traite de l'odométrie utilisant 2 roues codeuses.



Préalable :

Le robot est sensé utiliser 2 roues codeuses lui permettant de mesurer les déplacements gauche et droite du robot.

GAUCHE : variable comptant en nombre de pas le déplacement à gauche

DROITE : variable comptant en nombre de pas le déplacement à droite

A partir de ces 2 variables sont calculés le déplacement et l'orientation du robot.

D_ROBOT représente en nombre de pas le déplacement du centre de l'essieu.

theta_ROBOT représente en nombre de pas l'orientation de l'axe longitudinal du robot par rapport à l'axe des x.

$$D_ROBOT = \frac{1}{2}(GAUCHE + DROITE)$$

$$\theta_ROBOT = \theta_ROBOT_INIT + (DROITE - GAUCHE)$$

On dit que les 2 variables **D_ROBOT**, **theta_ROBOT** exprimées en nombre de pas, appartiennent au système d'unité *Robot*.

Dans la partie théorique de ce document ainsi que sur la figure on utilise les variables :

D pour le déplacement et **theta**, pour l'orientation exprimées dans le système *SU* (mètre ou cm et radian), avec la correspondance :

$$D_ROBOT = COEF_D * D \quad (D \text{ en mètre})$$

$$\theta_ROBOT = COEF_ROT * \theta \quad (\theta \text{ en radian})$$

COEF_D et COEF_ROT sont des constantes pré-calculées puis ajustées expérimentalement

On suppose le calcul échantillonné avec une période **Tech**.

A chaque instant **tn**, le robot reçoit les informations fournies par les codeurs incrémentaux et en déduit les valeurs de **Dn** et **Θn**.

Dn : distance curviligne parcourue par le centre de l'essieu.

Θn : orientation (axe longitudinal du robot par rapport à l'axe des x).

Sur la figure les points A et B représentent les positions du robot aux 2 instants consécutifs **tn-1** et **tn**, avec :

Θn-1 : orientation à l'instant **tn-1**

Θn : orientation à l'instant **tn**

L = Dn - Dn-1 : distance curviligne parcourue pendant une période **Tech**.

dx, **dy** : projections de l'arc **L** sur les 2 axes x et y.

Le robot ne recevant aucune information en dehors des instants d'échantillonnage, l'odométrie se présente ainsi :

A chaque instant **tn**, le robot calcule **dx** et **dy** en fonction de **Θn**, **Θn-1**, et **L**
Puis il procède à l'intégration numérique :

$$x_n = x_{n-1} + dx$$

$$y_n = y_{n-1} + dy$$

En théorie la mission est impossible, car le calcul des projections **dx** et **dy** de l'arc de trajectoire **L** dépend évidemment de sa forme. Hors les évolutions de la trajectoire entre A et B sont ignorées par le robot.

Afin de sortir de l'impasse, l'idée alors est de choisir une fréquence d'échantillonnage suffisamment rapide afin de permettre une approximation acceptable bien que simpliste à priori de la trajectoire entre les points A et B

Approximation linéaire :

Elle consiste à approximer l'arc de trajectoire entre A et B par la sécante AB (Soit l'approximation en distance **AB=L**).

On suppose également que l'orientation de la sécante est égale à la moyenne des orientations en A et B.

L'odométrie se résume alors aux équations :

Orientation de la sécante AB :

$$\theta_{moy} = \frac{1}{2}(\Theta_n + \Theta_{n-1})$$

Projection de la sécante AB sur les 2 axes :

$$dx = L \cdot \cos(\theta_{moy})$$

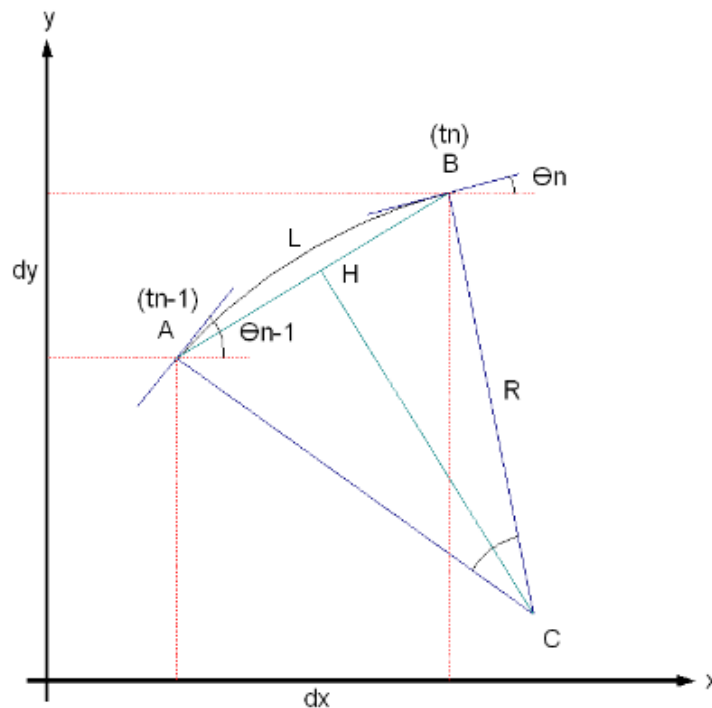
$$dy = L \cdot \sin(\theta_{moy})$$

Loi de récurrence de l'intégration numérique :

$$x_n = x_{n-1} + dx$$

$$y_n = y_{n-1} + dy$$

Approximation circulaire. Version1 :



On approxime la trajectoire entre A et B par un arc de cercle.

L'angle au centre inscrit (CA,CB) correspondant vaut en valeur algébrique :

$$\Delta\theta_n = \theta_n - \theta_{n-1}$$

On en déduit le rayon de courbure de la trajectoire

$$R = L/\Delta\theta_n$$

(sur la figure : $L > 0$; $\Delta\theta_n < 0$; donc $R < 0$)

On considère la relation algébrique : $\mathbf{AB} = \mathbf{CB} - \mathbf{CA}$

L'orientation de CB est égale à $\theta_n + \pi/2$.

L'orientation de CA est égale à $\theta_{n-1} + \pi/2$.

On projette sur les axes x et y la relation algébrique $\mathbf{AB} = \mathbf{CB} - \mathbf{CA}$

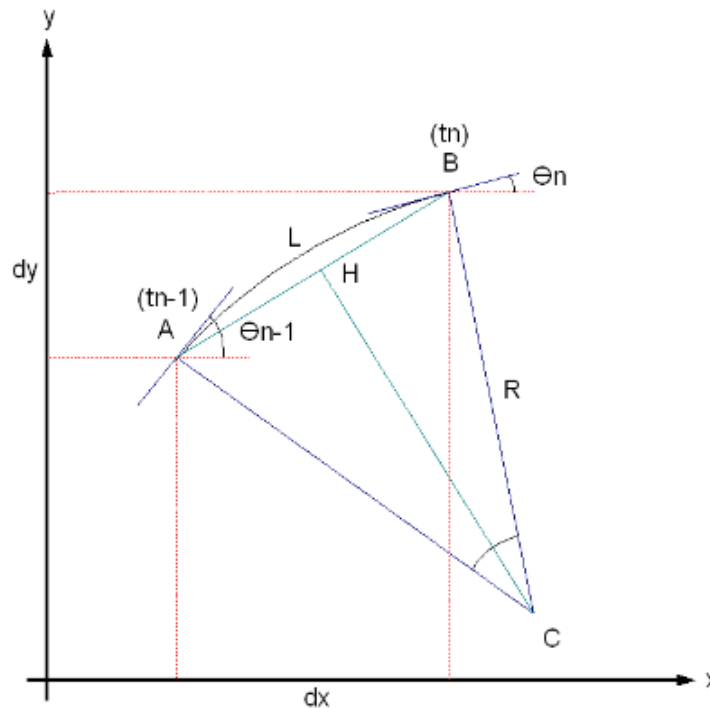
$$dx = R.(\sin(\theta_n) - \sin(\theta_{n-1}))$$

$$dy = -R.(\cos(\theta_n) - \cos(\theta_{n-1}))$$

Avec toujours la loi de récurrence de l'intégration numérique :

$$x_n = x_{n-1} + dx$$

$$y_n = y_{n-1} + dy$$

Approximation circulaire. Version 2:

L'angle au centre inscrit (CA,CB) correspondant vaut en valeur algébrique :

$$\Delta\theta_n = \theta_n - \theta_{n-1}$$

On en déduit le rayon de courbure

$$R = L/\Delta\theta_n \quad \text{relation algébrique}$$

Dans le triangle rectangle AHC :

$$AH = R \cdot \sin(\Delta\theta_n/2) \quad \text{relation algébrique}$$

(Sur la figure $L > 0$; $\Delta\theta_n < 0$; $R < 0$; $AH > 0$)

Soit K le rapport entre les valeurs algébriques de la sécante AB et de l'arc de cercle

$$K = AB/L = \sin(\Delta\theta_n/2) / (\Delta\theta_n/2)$$

On voit apparaître la célèbre fonction en $\sin x/x$ ($K < 1$ et $K \rightarrow 1$ si $\Delta\theta_n \rightarrow 0$)

Orientation de la sécante AB :

$$\theta_{\text{moy}} = \frac{1}{2}(\theta_n + \theta_{n-1})$$

Projection de la sécante AB sur les 2 axes : (Sachant que $AB = K \cdot L$)

$$dx = K \cdot L \cdot \cos(\theta_{\text{moy}})$$

$$dy = K \cdot L \cdot \sin(\theta_{\text{moy}})$$

Loi de récurrence de l'intégration numérique :

$$x_n = x_{n-1} + dx$$

$$y_n = y_{n-1} + dy$$

Comparaison entre les 2 algorithmes circulaires :

Rappel algorithme circulaire version1 :

$$\begin{aligned} dx &= R.(\sin(\theta_n) - \sin(\theta_{n-1})) \\ dy &= -R.(\cos(\theta_n) - \cos(\theta_{n-1})) \\ \text{avec } R &= L/\Delta\theta_n \quad \text{appelé rayon de courbure} \end{aligned}$$

Rappel algorithme circulaire version2 :

$$\begin{aligned} dx &= K.L.\cos(\theta_{\text{moy}}) \\ dy &= K.L.\sin(\theta_{\text{moy}}) \\ \text{avec } K &= \sin(\Delta\theta_n/2) / (\Delta\theta_n/2) \\ \theta_{\text{moy}} &= \frac{1}{2}(\theta_n + \theta_{n-1}) \end{aligned}$$

En fait il s'agit des mêmes relations mais l'une est présentée sous la forme d'une différence de sinus cosinus et l'autre sous la forme d'un produit de sinus cosinus. Pour les incrédules, faire appel aux propriétés classiques des fonctions trigo pour démontrer qu'il y a bien équivalence entre les 2 versions ou se reporter à l'ANNEXE 1 en fin de document.

Que choisir ?

Si on tatillonne, on dira : Avantage à la version 2 pour le temps de calcul car 3 fonctions trigo à calculer au lieu de 4 pour la version1. Mais cet avantage n'est pas décisif.

Plus sérieusement, lorsque le robot se déplace en ligne droite (c'est-à-dire presque en ligne droite), $\Delta\theta_n$ qui doit rester nul peut prendre des valeurs très faibles et R des valeurs monstrueuses. La version 1 pour de trajectoires quasi-rectilignes multiplie alors un infiniment grand par un infiniment petit, ce qui pose peut-être des problèmes d'erreurs de quantification.

La version 2 est plus clean de ce point de vue car elle évite le calcul du rayon de courbure et ne fait intervenir que des valeurs raisonnables.

Ainsi si on reprend l'exemple du robot RCVA lancé en ligne droite à 0.8 m/s avec une période d'échantillonnage de 5 ms, ça donne :

$$L = 4\text{mm}$$

1LSB de variation sur le codeur d'une des roues donne alors :

$$\Delta\theta_n = 4.36 \cdot 10^{\text{puissance}-5} \text{ radians !} \quad (\text{Angle minuscule})$$

$$R = 91.7 \text{ mètres!} \quad (\text{Monstrueux si on compare à } L=4\text{mm})$$

Comparaison entre approximation linéaire et circulaire :

Rappel algorithme linéaire :

$$\begin{aligned} dx &= L.\cos(\theta_{\text{moy}}) \\ dy &= L.\sin(\theta_{\text{moy}}) \\ \text{avec } \theta_{\text{moy}} &= \frac{1}{2}(\theta_n + \theta_{n-1}) \end{aligned}$$

Comparons aux expressions de l'approximation circulaire version 2 ci-dessus

On est surpris de constater que les expressions sont équivalentes au facteur K près, la fameuse fonction en $\sin x/x$.

Prenons un exemple extrême pour chiffrer l'ordre de grandeur de K .

L'exemple est extrême dans la mesure où il peut représenter le pire des cas c'est-à-dire un robot à grande vitesse dans une courbe à rayon très court.

Vitesse=1m/s avec $R=25$ cm.

Tech=5 ms

Ca correspond à une force centrifuge de 60N pour un robot de 15 kg
(Notre robot RCVA est largement en perdition dans ces conditions).

$L=5\text{mm}$

$\Delta\theta_n = L/R = 0.02$ rd

$K = \sin(0.01)/0.01 = 0.999983...$

K vaut 1 à $2 \cdot 10^{\text{puissance}-5}$ près !

Et il s'agit on le rappelle d'un cas extrême.

A chacun d'en tirer ses conclusions mais il semblerait que l'approximation linéaire soit suffisante en odométrie dans la plupart des cas d'un robot EUROBOT.

Il est vrai que le seul calcul de K permet de passer facilement d'une version linéaire à une version circulaire. Ca coûte 1 ou 2 lignes de code supplémentaires et ça mérite des essais comparatifs sur le terrain.

(Des vidéos devraient suivre pour illustrer ces résultats)

ANNEXE 1 :

Algorithme circulaire : Equivalence entre expressions des versions 1 et 2

Si on fait appel à sa mémoire reptilienne...ou plus sûrement à Wikipedia

$$\sin(p) - \sin(q) = 2\sin[(p-q)/2] \cdot \cos[(p+q)/2]$$

$$\cos(p) - \cos(q) = -2\sin[(p-q)/2] \cdot \sin[(p+q)/2]$$

On applique aux expressions dx et dy de la version1

$$dx = R \cdot (\sin(\theta_n) - \sin(\theta_{n-1}))$$

$$dy = -R \cdot (\cos(\theta_n) - \cos(\theta_{n-1}))$$

$$dx = 2R \cdot \sin(\Delta\theta_n/2) \cdot \cos(\theta_{\text{moy}})$$

$$dy = 2R \cdot \sin(\Delta\theta_n/2) \cdot \sin(\theta_{\text{moy}})$$

$$\text{avec } R = L/\Delta\theta_n$$

Et on aboutit aux expressions de la version2

$$dx = L \cdot [\sin(\Delta\theta_n/2)] / (\Delta\theta_n/2) \cdot \cos(\theta_{\text{moy}})$$

$$dy = L \cdot [\sin(\Delta\theta_n/2)] / (\Delta\theta_n/2) \cdot \sin(\theta_{\text{moy}})$$

ANNEXE 2:**Exemple de programme en langage C**

Dans ce programme les variables distance et orientation sont exprimées dans le système d'unité *ROBOT* c'est-à-dire en nombre de pas incrémentaux.

Bien sûr, les orientations seront provisoirement converties en radians pour les calculs des sinus et cosinus.

Toutes les variables pourraient être de type flottant mais pour des raisons d'éthique et autre, les types des variables seront adaptés à leur nature.

```

/***** Définition des constantes *****/

#define COEF_D      3892L           // 1 cm → 3892 pas incrémentaux
#define COEF_ROT    2036L           // 1 degré → 2036 pas
#define PI          3.14159265
#define CONV_RD     PI/(180*COEF_ROT) // conversion teta en pas → teta en radian

#define LINEAIRE     1
#define CIRCULAIRE   2

/***** Définition des variables globales *****/
En fait ces variables seront rangées avantageusement dans des structures
*****/
long gauche, droite, distance, distance_precedente ; // variables entières sur 32 bits
long orient, orient_precedente, orient_init ;
float x, y;
int x_cm, y_cm ; // variables sur 16 bits

char algo ; // Variable définie uniquement pour des raisons de présentation
// En pratique le choix sera définitif entre algo linéaire ou circulaire

/***** Conditions initiales *****/
Fonction appelée 1 fois au début du programme qui initialise les variables orientation,
x, y en fonction de la position du robot au départ
*****/

void fonction_initialisation (float x0, float y0, long teta0)
{
    orient_init = teta0 ;
    orient_precedente = teta0 ;
    x = x0 ;
    y = y0 ;
    distance = 0 ;
    distance_precedente = 0 ;
}

```



```

/ ***** Interruption *****
Fonction déclenchée par Timer réglé à 5 ms.
Cette fonction est appelée toutes les 5 ms pendant toute la durée du match
*****/

void prog_interruption (...)
{
    saisie_capteurs_incrementaux(); //Actualise les variables gauche et droite
    calcul_xy(); // La fonction qui nous intéresse
    calcul_des_consignes(); // Les fameuses consignes trapézoïdales
    asservissement_polaire(); // Asservit le robot sur ses consignes
    ...
}

/***** Calcul de x et y *****/
Cette fonction récupère les 2 variables globales gauche et droite et actualise les
valeurs de x et y. Le calcul de x_cm et y_cm peut être fait en dehors de la fonction
*****/

void calcul_xy(void)
{
    int delta_d, delta_orient;
    long orient_moy;
    float orient_moy_radian, delta_orient_radian;
    float K, dx, dy;

    // Correspondance avec notation du document théorique
    Distance = ( droite + gauche)/2; // distance en pas parcourue à tn
    orient = orient_init + (droite - gauche); //correspond à θn mais en pas
    delta_d = distance - distance_precedente; // correspond à L mais en pas
    delta_orient = orient - orient_precedente; // correspond à Δθn mais en pas
    orient_moy = (orient + orient_precedente)/2; //correspond à θmoy en pas
    delta_orient_radian = CONV_RD * delta_orient; //correspond à Δθn en rd
    orient_moy_radian = CONV_RD * orient_moy; //correspond à θmoy en rd
    if (algo==LINEAIRE)
        K=1;
    else // Choix de l'algo circulaire
        { if (delta_orient==0) // Pour éviter la division par zéro
            K=1;
          else
            K=( sin(delta_orient_radian /2)) / (delta_orient_radian /2);
        }
    dx=K* delta_d *cos(orient_moy_radian);
    dy=K* delta_d *sin(orient_moy_radian);

    x= x + dx; // valeurs exprimées dans le système d'unité robot
    y= y + dy;

    x_cm= (int) (x/COEF_D); // Ce calcul peut être fait en dehors de la fonction
    y_cm= (int) (y/COEF_D); // quand on en a besoin
    orient_precedente = orient; // actualisation de θn-1
    distance_precedente = distance; //actualisation de Dn-1
}

```


ANNEXE 3

Correction de force centrifuge :

Un véhicule se déplaçant sur une trajectoire courbe est soumis à la force centrifuge. (Centrifuge ou centripète, c'est comme on veut).

Cette force qui tire le robot à l'extérieur de sa trajectoire, induit des déplacements non pris en compte par une odométrie classique.

La méthode présentée ici permet de corriger dans une certaine mesure ces erreurs odométriques induites.

La force centrifuge F étant perpendiculaire à la tangente à la trajectoire, l'idée est de supposer que le déplacement latéral résultant est proportionnel à cette force. Le coefficient de proportionnalité sera déterminé expérimentalement.

Cette force centrifuge peut prendre les 3 formes équivalentes.

$$F = M.v^2/R = M.(d\theta/dt)^2.R = M.(d\theta/dt).v$$

F: force centrifuge

M: masse du véhicule

v: vitesse linéaire

dθ/dt : vitesse de rotation

R: Rayon de courbure de la trajectoire

Pour ce calcul, la 3eme forme $F = M.(d\theta/dt).v$ est la plus intéressante car elle ne fait pas intervenir R .

Si on se reporte à l'exemple du programme en C

dθ/dt est représenté par **delta_orient**

v est représenté par **delta_d**

Cette force centrifuge étant perpendiculaire à la trajectoire, à l'instant t_n , son orientation est perpendiculaire à l'orientation **orient_moy**

Si on admet que la dérive du robot est proportionnelle à cette force et dans la même direction, en projetant sur les 2 axes ça donne :

$$\begin{aligned} \text{derive_x} &= \text{CORFUGE} * \text{delta_orient} * \text{delta_d} * \sin(\text{orient_moy_radian}); \\ \text{derive_y} &= - \text{CORFUGE} * \text{delta_orient} * \text{delta_d} * \cos(\text{orient_moy_radian}); \end{aligned}$$

CORFUGE: constante de correction centrifuge déterminée expérimentalement.

Rappelons les expressions de **dx** et **dy** sans correction centrifuge

$$\begin{aligned} dx &= K * \text{delta_d} * \cos(\text{orient_moy_radian}); \\ dy &= K * \text{delta_d} * \sin(\text{orient_moy_radian}); \end{aligned}$$

On peut alors écrire en combinant :

```
derive_x = CORFUGE* delta_orient*dy;
derive_y = - CORFUGE* delta_orient*dx;
```

Exemple CORFUGE=1.2 (valeur déterminée pour le robot RCVA mais à adapter)

A noter la relation croisée entre x et y.

A noter aucune nouvelle fonction trigo à calculer !

Dans ces conditions la nouvelle fonction avec correction centrifuge deviendrait :

```
void calcul_xy_avec_correction_centrifuge(void)
{
    int    delta_d, delta_orient ;
    long   orient_moy ;
    float  orient_moy_radian, delta_orient_radian ;
    float  K, dx, dy, derive_x, derive_y ;

    // Correspondance avec notation du document théorique
    distance = ( droite + gauche)/2 ;           // distance en pas parcourue à tn
    orient = orient_init + (droite - gauche) ;   // correspond à θn mais en pas
    delta_d = distance - distance_precedente ;  // correspond à L mais en pas
    delta_orient = orient - orient_precedente ; // correspond à Δθn mais en pas
    orient_moy = (orient + orient_precedente)/2 ; // correspond à θmoy en pas
    delta_orient_radian = CONV_RD * delta_orient ; // correspond à Δθn en rd
    orient_moy_radian = CONV_RD * orient_moy ; // correspond à θmoy en rd
    if (algo==LINEAIRE)
        K=1;
    else // Choix de l'algo circulaire V2
        { if (delta_orient==0) // Pour éviter la division par zéro
            K=1;
          else
            K=( sin(delta_orient_radian /2)) / (delta_orient_radian /2);
        }
    dx=K* delta_d *cos(orient_moy_radian);
    dy=K* delta_d *sin(orient_moy_radian);

    derive_x = CORFUGE* delta_orient*dy; // Correction de dérive
    derive_y = - CORFUGE* delta_orient*dx;

    x= x + dx + derive_x; // valeurs exprimées dans le système d'unité robot
    y= y + dy + derive_y;

    orient_precedente = orient ;
    distance_precedente = distance ;
}
```


En guise de conclusion :

L'odométrie par codeurs incrémentaux procédant par intégration est soumise à une dérive c'est-à-dire à une incertitude sur le positionnement qui croît avec le temps. La valeur de cette dérive dépend bien sûr de la qualité de l'algorithme choisi.

Mais bien d'autres facteurs vont influencer sur la précision du résultat :

- La qualité mécanique de la réalisation du système de roues codeuses (Absence de jeu, alignement parfait avec les roues motrices, parallélisme parfait des 2 roues codeurs avec l'axe longitudinal du robot).

- La résolution des codeurs ((Nombre de pas par tours). En pratique une résolution d'au moins 1000 points semble nécessaire.

- Le choix du diamètre des roues codeuses (Trop petites, elles vont subir les irrégularités du terrain. Trop grandes elles font perdre en résolution)

- La qualité du module de comptage décomptage (Filtrage numérique des signaux, prise en compte des fronts montants et descendants des 2 canaux au front près).

- Le choix de la fréquence d'échantillonnage. Une fréquence trop grande va conduire à des erreurs de quantifications excessives. Une fréquence trop faible va allonger les arcs de trajectoire donc augmenter l'erreur dans l'approximation (qu'elle soit linéaire ou circulaire).

Si toutes ces conditions ne sont pas réunies, des recalages en cours de match sont à prévoir, les plus simples à mettre en œuvre étant les recalages bordure.

Quelques liens à propos de la navigation inertielle :

http://www.st.com/stonline/products/families/sensors/motion_sensors.htm

Produits industriels :

<http://www.sbg-systems.com/products/ig-500a>

<http://www.jr-france.com/site/produits/Capteurs/Systèmes%20inertiels>

Sur l'odométrie par balise, lire l'article remarquable :

http://zer0.droids-corp.org/microb_technology_balises_2010_v0_8.pdf

