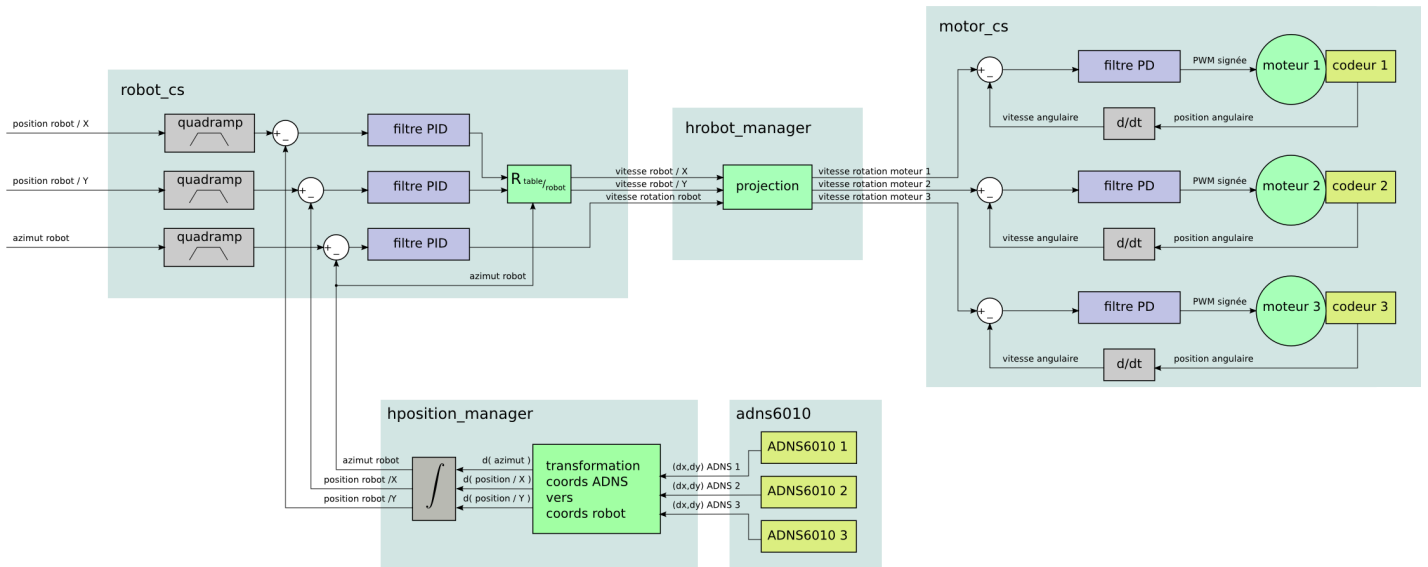

Rob'Otter 2k9

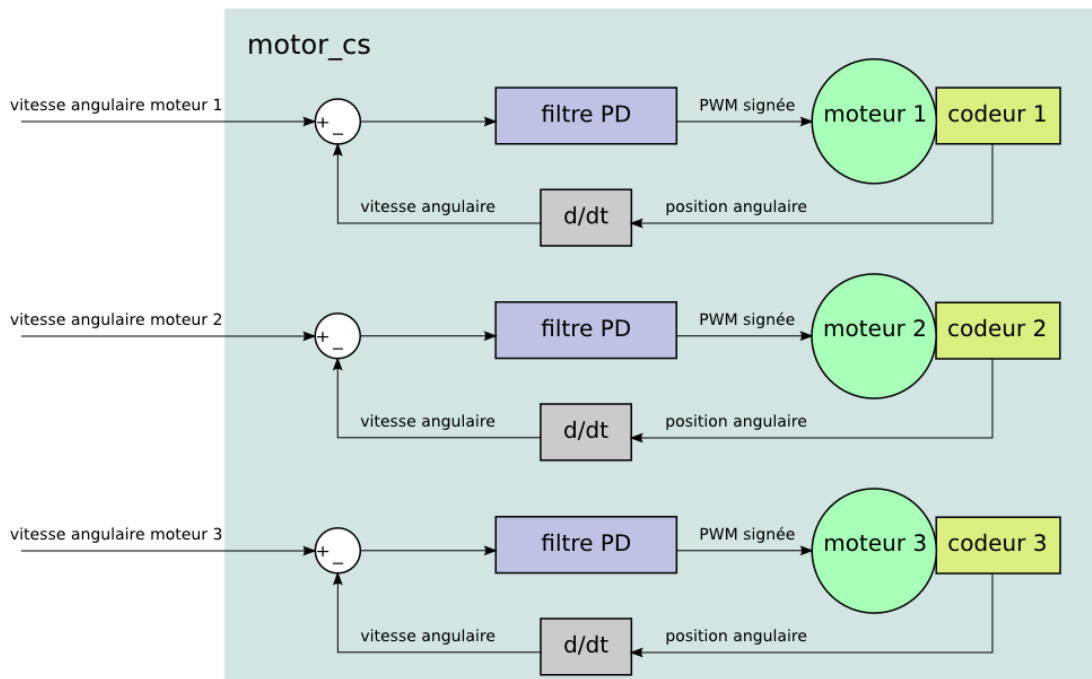
Galipeur



I ASSERVISSEMENT



1.1 ASSERVISSEMENT EN VITESSE DES MOTEURS



Chaque moteur est individuellement asservi en vitesse par un régulateur PD (Proportionnel, Dérivée) et un codeur incrémental HEDS5540 fixé à l'axe moteur.

1.1.1 CONSIGNE

La consigne est ici une vitesse moteur exprimée en *pas codeurs / tick asservissement*.

1.1.2 CORRECTION

La correction de l'asservissement est appliquée sous la forme d'une PWM + signe envoyée aux drivers moteurs, permettant de contrôler le moteur en tension.

1.1.3 FEEDBACK

L'information de vitesse est récupérée depuis les codeurs incrémentaux HEDS5540, branchée sur le FPGA de l'UNIOC-NG, traités pour obtenir une position absolue de l'axe moteur puis transmis à l'ATmega128 de l'UNIOC-NG.

Un filtre dérivateur est appliqué sur la boucle de retour afin d'obtenir la vitesse de rotation de l'axe moteur.

1.1.4 CODE

L'asservissement en vitesse des moteurs est réalisé dans les fichiers `unioc_asserv/motor_cs.c`, `unioc_asserv/motor_cs_config.h` et `unioc_asserv/motor_cs_config.h`.

INITIALISATION

`unioc_asserv/main.c` ligne 224 :

```
//-----  
// Initialize control systems for motors  
printf("# Initializing motors control systems : ");  
motor_cs_init();  
printf("# OK\n");
```

La fonction `motor_cs_init` assure l'initialisation des `control_system_managers`, filtres PID et PWM moteurs.

MISE À JOUR

La mise à jour de l'asservissement est effectuée par la fonction `motor_cs_update()` appelée par la fonction `hrobot_set_motors()` du module `hrobot_manager` lors de l'envoi des consignes aux moteurs.

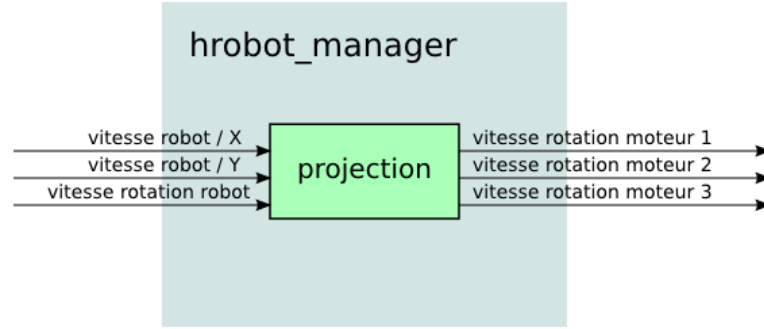
`modules/devices/hrobot_manager/hrobot_manager.c` ligne 74 :

```
// set motors speeds  
if(hrs->motors_accessor)  
(hrs->motors_accessor)(hrs->motors_accessor_params, v0, v1, v2);
```

Le pointeur `hrs->motors_accessor` est assigné à `motor_cs_update()` dans `unioc_asserv/main.c` ligne 208 :

```
hrobot_set_motors_accessor(&system, motor_cs_update, NULL);
```

1.2 PROPULSION VECTORIELLE



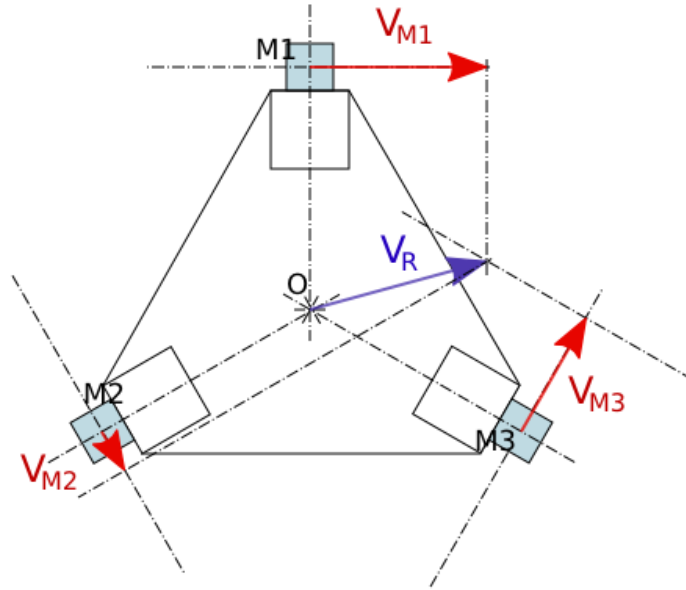
1.2.1 THÉORIE

La propulsion vectorielle permet de transformer une consigne en vitesse exprimée dans l'espace du robot :

$$\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix}_{robot} \quad (1.1)$$

En une consigne en vitesse sur chaque moteur :

$$\begin{pmatrix} v_{M1} \\ v_{M2} \\ v_{M3} \end{pmatrix}_{moteurs} \quad (1.2)$$



Soit \vec{u}_{M1} , \vec{u}_{M2} , \vec{u}_{M3} respectivement les vecteurs unitaires colinéaires au diamètre de chaque roue et appartenant au plan (\vec{x}, \vec{y}) , chaque vecteur positif dans le sens de poussée positive de chaque moteur.

Soit R la distance entre le point d'appui d'une omniwheel et le centre du robot.

On pose $\vec{V}_{robot/table_O} = \begin{pmatrix} v_x \\ v_y \\ 0 \end{pmatrix}_O$ vecteur vitesse du robot en O dans le repère lié à la table.

On effectue le déplacement du torseur cinématique du robot en O aux points de contacts de chaque roue M_1 , M_2 et M_3 .

Au point O le torseur cinématique du robot est :

$$\{\nu_{robot/table}\} = \left\{ \begin{array}{c} \omega_z \cdot \vec{z} \\ \overrightarrow{V_{robot/table}_O} \end{array} \right\}_O = \left\{ \begin{array}{cc} v_x & 0 \\ v_y & 0 \\ 0 & \omega_z \end{array} \right\}_O \quad (1.3)$$

Déplacement du torseur en M_1 :

$$\{\nu_{robot/table}\} = \left\{ \begin{array}{c} \omega_z \cdot \vec{z} \\ \overrightarrow{V_{robot/table}_O} \end{array} \right\}_O \quad (1.4)$$

$$= \left\{ \begin{array}{c} \omega_z \cdot \vec{z} \\ \overrightarrow{V_{robot/table}_O} + \overrightarrow{OM_1} \wedge \omega_z \cdot \vec{z} \end{array} \right\}_{M_1} \quad (1.5)$$

$$= \left\{ \begin{array}{c} \omega_z \cdot \vec{z} \\ \overrightarrow{V_{robot/table}_O} + R \cdot \omega_z \cdot \vec{u}_{M_1} \end{array} \right\}_{M_1} \quad (1.6)$$

On obtient de même en M_2 et M_3 :

$$\{\nu_{robot/table}\} = \left\{ \begin{array}{c} \omega_z \cdot \vec{z} \\ \overrightarrow{V_{robot/table}_O} + R \cdot \omega_z \cdot \vec{u}_{M_2} \end{array} \right\}_{M_2} \quad (1.7)$$

$$\{\nu_{robot/table}\} = \left\{ \begin{array}{c} \omega_z \cdot \vec{z} \\ \overrightarrow{V_{robot/table}_O} + R \cdot \omega_z \cdot \vec{u}_{M_3} \end{array} \right\}_{M_3} \quad (1.8)$$

Chaque omniwheel n ne peut fixer la vitesse que suivant l'axe \vec{u}_{M_n} , on obtient donc, en notant θ_{M_n} l'orientation respective de chaque vecteur u_{M_n} :

$$v_{M_n} = \overrightarrow{V_{robot/table}_{M_n}} \cdot \vec{u}_{M_n} \quad (1.9)$$

$$= (\overrightarrow{V_{robot/table}_O} + R \cdot \omega_z \cdot \vec{u}_{M_n}) \cdot \vec{u}_{M_n} \quad (1.10)$$

$$= \begin{pmatrix} v_x \\ v_y \\ 0 \end{pmatrix} \cdot \begin{pmatrix} \cos \theta_{M_n} \\ \sin \theta_{M_n} \\ 0 \end{pmatrix} + R \cdot \omega_z \quad (1.11)$$

$$= v_x \cdot \cos \theta_{M_n} + v_y \cdot \sin \theta_{M_n} + R \cdot \omega_z \quad (1.12)$$

Ce qui donne le système d'équations suivant :

$$v_{M_1} = v_x \cdot \cos \theta_{M_1} + v_y \cdot \sin \theta_{M_1} + R \cdot \omega_z \quad (1.13)$$

$$v_{M_2} = v_x \cdot \cos \theta_{M_2} + v_y \cdot \sin \theta_{M_2} + R \cdot \omega_z \quad (1.14)$$

$$v_{M_3} = v_x \cdot \cos \theta_{M_3} + v_y \cdot \sin \theta_{M_3} + R \cdot \omega_z \quad (1.15)$$

1.2.2 CODE

INITIALISATION

Le module `hrobot_manager` est initialisé dans le fichier `main.c` lignes 205 à 211 :

```
printf("# Initializing robot manager : ");
hrobot_init(&system);
hrobot_set_motors_accessor(&system, motor_cs_update, NULL);
printf("# OK\n");

printf("# Robot manager is GO\n\n");
```

MISE À JOUR

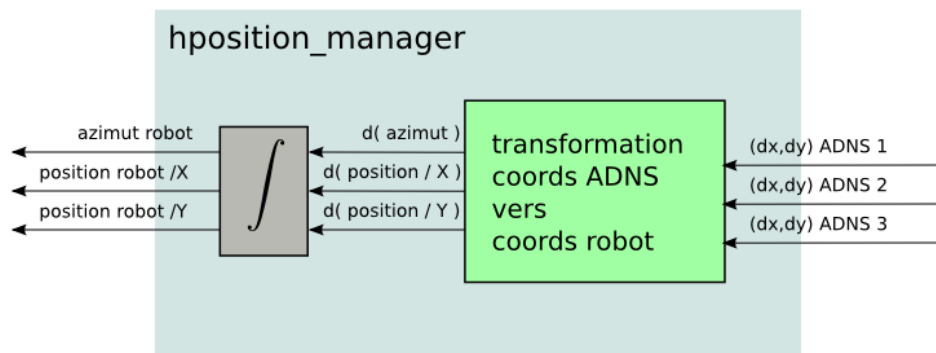
Ces transformations sont implémentées dans la fonction `hrobot_set_motors` du module `hrobot_manager` aux lignes 62 à 70 :

```
// project speed vector on each motor
v0 = vx*HROBOT_MOTOR0_COS_COURSE + vy*HROBOT_MOTOR0_SIN_COURSE;
v1 = vx*HROBOT_MOTOR1_COS_COURSE + vy*HROBOT_MOTOR1_SIN_COURSE;
v2 = vx*HROBOT_MOTOR2_COS_COURSE + vy*HROBOT_MOTOR2_SIN_COURSE;

//
v0 += omega;
v1 += omega;
v2 += omega;
```

Notons que dans la version présentée du code aucun effort de mise en conformité des unités n'a été effectuée R étant égal à 1 ici.

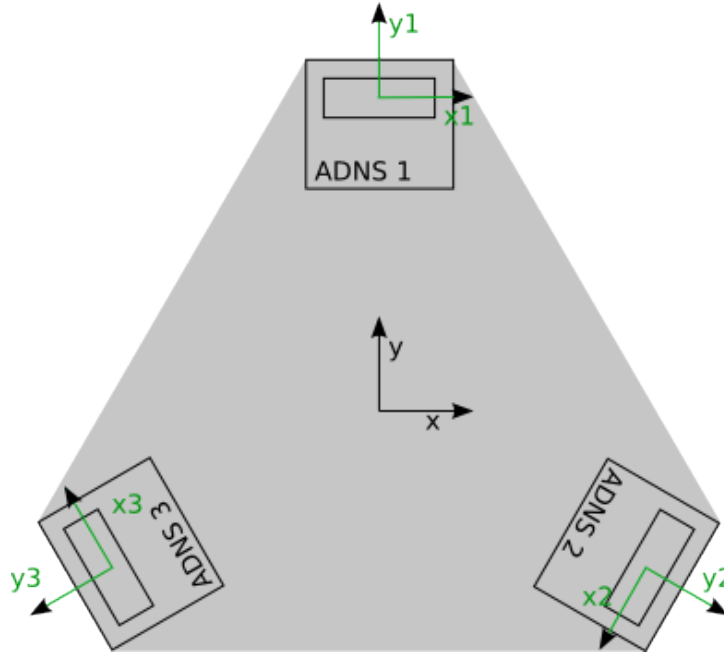
1.3 POSITIONNEMENT



1.3.1 TRANSFORMATION VITESSES ADNS VERS VITESSES ROBOT

THÉORIE

La carte ADNS6010 donne les vitesses de déplacement de chaque capteur dans son repère.



Soit le vecteur $\begin{pmatrix} v_{x1} \\ v_{y1} \\ v_{x2} \\ v_{y2} \\ v_{x3} \\ v_{y3} \end{pmatrix}$ représentant les informations de vitesses renvoyées par le capteur ADNS.

Sachant que chaque repère ADNS est une transformation linéaire du repère du robot on peut dire, en notant $\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix}$ la vitesse du robot, qu'il existe une matrice A telle que :

$$\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix} = A \begin{pmatrix} v_{x1} \\ v_{y1} \\ v_{x2} \\ v_{y2} \\ v_{x3} \\ v_{y3} \end{pmatrix} \quad (1.16)$$

La matrice A est donc une matrice 3x6 obtenue par identification du capteur ADNS.

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \end{pmatrix} \quad (1.17)$$

CODE

Initialisation

Le module `hposition_manager` est initialisé dans le fichier `main.c` lignes 216 à 221 :

```
printf("# Initializing position manager : ");
hposition_init( &position );
hposition_set( &position, 0.0, 0.0, 0.0 );
printf("# OK\n");
```



```
printf("# Position manager is GO\n\n");
```

Mise à jour

La transformation, réduite à un produit matriciel, est implémenté dans la fonction `hposition_update` du module `hposition_manager` aux lignes 119 à 137.

```
//-----  
// Transform speed in ADNS coordinates to robot coordinates  
  
// compute speed in ADNS coordinates  
for(i=0;i<6;i++)  
    v[i] = hpos->pAdnsVectors[i] - adns6010.vectors[i];  
  
// update previous ADNS vectors  
for(i=0;i<6;i++)  
    hpos->pAdnsVectors[i] = adns6010.vectors[i];  
  
// for each robot coordinate (x,y,a) compute a dx of mouvement  
for(k=0;k<3;k++)  
{  
    dp[k] = 0.0;  
  
    // for each ADNS coordinate (vx1,vy1,vx2,vy2,vx3,vy3)  
    for(i=0;i<6;i++)  
    {  
        dp[k] += hrobot_adnsMatrix[k][i]*v[i];  
    }  
}
```

1.3.2 INTÉGRATION DES VITESSES ADNS

THÉORIE

L'élément $\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix}$ calculé précédemment est donné dans le repère lié au capteur ADNS.

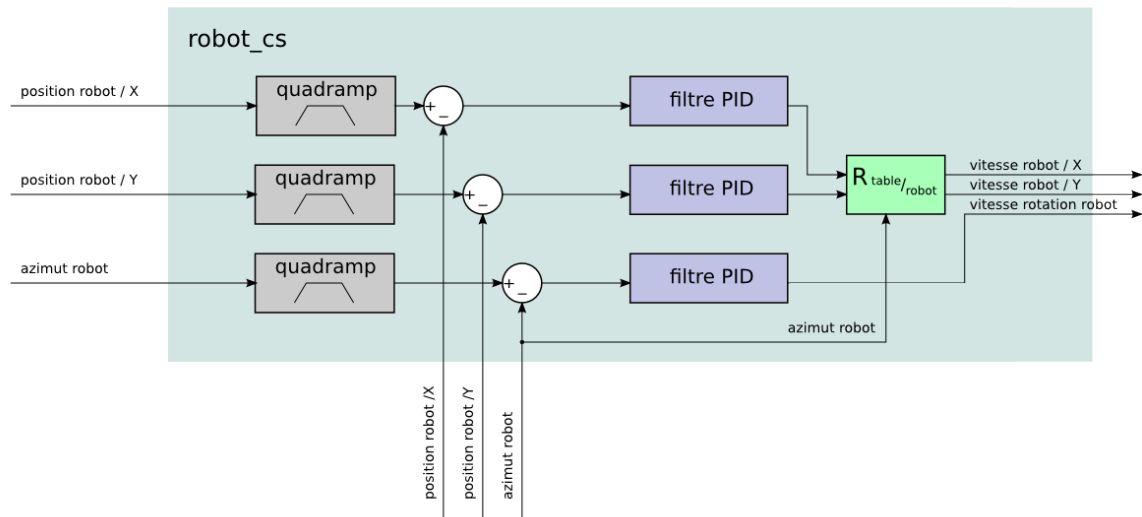
Il convient donc de le transformer par une rotation de α , angle du robot par rapport à la table.

CODE

L'intégration de l'élément de position calculé est effectué à la suite dans la fonction `hposition_update` aux lignes 142 à 147.

```
alpha = hpos->position.alpha;  
  
x = hpos->position.x + dp[HROBOT_DX]*cos(alpha) - dp[HROBOT_DY]*sin(alpha);  
y = hpos->position.y + dp[HROBOT_DX]*sin(alpha) + dp[HROBOT_DY]*cos(alpha);  
  
alpha += dp[HROBOT_DA];
```

1.4 ASSERVISSEMENT EN POSITION



Le robot est asservi en position par un régulateur PID sur chaque dimension : translation suivant \vec{x} , translation suivant \vec{y} , rotation suivant \vec{z} . La dernière étape consiste en une rotation du vecteur afin de le passer du repère R_{table} au repère R_{robot} .

1.4.1 BOUCLES D'ASSERVISSEMENT

CONSIGNE

La consigne est une position exprimée en mm et en radians.

Un filtre *quadramp* est appliqué à la consigne permettant de limiter ses dérivées premières et secondes, limitant ainsi l'accélération et la vitesse du robot.

CORRECTION

La correction de l'asservissement est un vecteur vitesse passé en consigne à l'asservissement des moteurs.

FEEDBACK

L'information de position est récupérée depuis le module `hposition_manager`, calculée depuis le retour de la carte ADNS6010.

CODE

L'asservissement en position du robot est réalisé dans les fichiers `unioc_asserv/robot_cs.c` et `unioc_asserv/robot_`.

INITIALISATION

`unioc_asserv/main.c` ligne 230 :

```
// Initialize control systems for robot
printf("# Initializing robot control systems : ");
robot_cs_init(&robot_cs);
robot_cs_set_hrobot_manager(&robot_cs, &system);
```

```
robot_cs_set_hposition_manager(&robot_cs,&position);
printf("# OK\n");
```

La fonction `robot_cs_init` assure l'initialisation des `control_system_managers`, filtres PID et quadramps.

MISE À JOUR

La mise à jour de l'asservissement est effectuée par la fonction `robot_cs_update()` appelée sur interruption par le module `scheduler` :

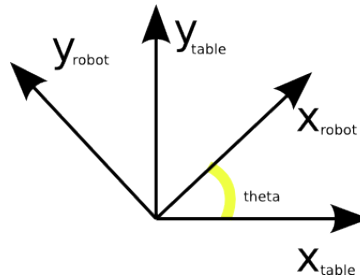
`unioc_asserv/main.c` ligne 324 :

```
// Unleash control systems
scheduler_add_periodical_event_priority(&robot_cs_update, &robot_cs,
                                        10,100);
```

1.4.2 CHANGEMENT DE REPÈRE

THÉORIE

La correction calculée par l'asservissement en position est exprimée dans le repère lié à la table de jeu R_{table} . Avant de l'envoyer aux asservissements moteurs il faut effectuer un changement de repère de R_{table} vers R_{robot} .



On a :

$$\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix}_{R_{table}} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix}_{R_{robot}} \quad (1.18)$$

$$\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix}_{R_{robot}} = \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix}_{R_{table}} \quad (1.19)$$

CODE

Le changement de repère est effectué dans la fonction `robot_cs_update()` :

```
vx_t      = cs_get_out(&csm_x);
vy_t      = cs_get_out(&csm_y);
omegaz_t = cs_get_out(&csm_angle);

hposition_get(rcs->hpm, &hvec);
```

```
alpha = -hvec.alpha;

vx_r = vx_t*cos(alpha) - vy_t*sin(alpha);
vy_r = vx_t*sin(alpha) + vy_t*cos(alpha);
```

1.5 FONCTIONNEMENT NOMINAL

1.5.1 INITIALISATION

La fonction `hposition_update` assurant la mise à jour de la position du robot à partir des informations retournées par les ADNS est appelée sur interruption, par le biais du module scheduler.

L'appel est initialisé dans `main.c` :

```
scheduler_add_periodical_event_priority(&hposition_update, &position, 400, 50);
```

La fonction `robot_cs_update` assurant la mise à jour de l'asservissement est appelée elle aussi sur interruption, l'appel est initialisé dans `main.c` :

```
scheduler_add_periodical_event_priority(&robot_cs_update, &robot_cs, 10, 100);
```

1.5.2 MISE À JOUR DE L'ASSERVISSEMENT

La mise à jour de l'asservissement se déroule de la manière suivante :

- appel de `robot_cs_update`, mise à jour des asservissements en translation suivant \vec{x} , translation suivant \vec{y} et angle selon \vec{z} ;
- appel de `hrobot_set_motors` avec les valeurs calculées par les 3 asservissements en position, afin de transformer une consigne en vitesse dans le repère du robot vers une consigne en vitesse sur chaque moteur ;
- appel de `motor_cs_update` avec les consignes en vitesse de chaque moteur, mise à jour des asservissement en vitesses des 3 moteurs ;
- mise à jour des PWMs avec les valeurs calculées par les asservissements en vitesse.

II IDENTIFICATION DU CAPTEUR ADNS

2.1 THÉORIE

Le capteur de position constitué de 3 capteurs optiques ADNS6010 n'est pas parfaitement ajusté dans les axes de la base roulante et il est, par conséquent, difficile de calculer la matrice A vue précédemment à partir des mesures du capteur.

$$\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix} = A \begin{pmatrix} v_{x1} \\ v_{y1} \\ v_{x2} \\ v_{y2} \\ v_{x3} \\ v_{y3} \end{pmatrix} \quad (2.1)$$

La méthode est donc d'effectuer une série de mesure de couples (U, V) où $U = \begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix}$ et $V = \begin{pmatrix} v_{x1} \\ v_{y1} \\ v_{x2} \\ v_{y2} \\ v_{x3} \\ v_{y3} \end{pmatrix}$.

Par exemple effectuer depuis le point $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ une translation de $400mm$ suivant l'axe \vec{x} donne le couple (U_0, V_0) suivant :

$$\left(\begin{pmatrix} 400 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 30110 \\ -17327 \\ -30676 \\ -17198 \\ 972 \\ 35188 \end{pmatrix} \right) \quad (2.2)$$

Il reste à effectuer un grand nombre de mesure sur des coordonnées variées et d'ensuite identifier la matrice A au moyen d'un algorithme d'identification.

2.2 CODE

L'identification est effectuée au moyen de la méthode ARMA sous SciLab grâce au script `unio_c_asserv/scilab/adns_calibration.sci`.