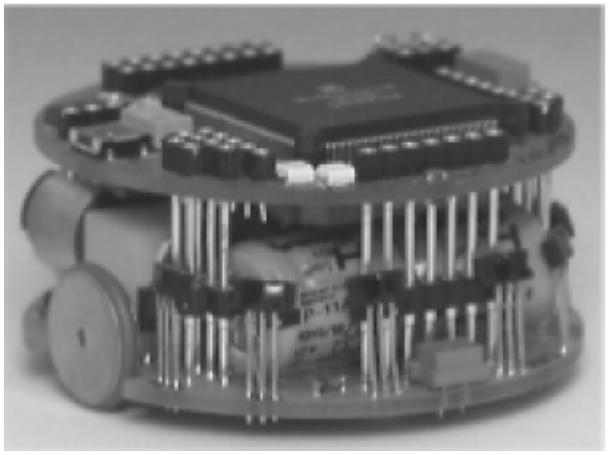
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Laboratoire d'Automatique



POSITIONING AND CONTROL OF MOBILE ROBOTS

Projet de Semestre / Eté
Année académique 2004 / 2005

Jean Martial Miamo Nguekam
Etudiant en 4ième année Electricité

Dr. Denis Gillet
Levente Bodizs
EPFL - LA

TABLE DES MATIERES

1 – INTRODUCTION 4

2 – MODÉLISATION DU ROBOT KHEPERA 5

3 – LE SYSTÈME DE MESURE 7

4 – LE TRAITEMENT D'IMAGE 8

4 – 1 Matrice de pixels couleur et matrice de pixels noirs et blancs 8

4 – 2 Détermination du centre de gravité des deux robots 8

4 – 3 Séparation de l'image 9

4 – 4 Calcul des positions de chaque robot dans le repère associé à chaque image 10
4 – 4 – 1 Cas de la coupe verticale 10
4 – 4 – 2 Cas de la coupe horizontale 11

4 – 5 Test du programme de traitement d'image 12

5 – LE FILTRE DE KALMAN ETENDU 13

5 – 1 La prédiction 13

5 – 2 La correction des mesures 15

5 – 3 Simulation du filtre de Kalman sous Matlab et Labview 16

5 – 4 Implantation du filtre de Kalman en temps réel 17

6 – LA COMMANDE PRÉDICTIVE 19

6 – 1 Schéma de commande 20

6 – 2 Génération de la trajectoire de référence 21

6 – 3 Les paramètres de la commande prédictive implantée 22

7 – TESTS EXPÉRIMENTAUX 23

7 – 1 Test de convergence du filtre de Kalman Etendu 23
7 – 1 – 1 Expérience 1 23
7 – 1 – 2 Expérience 2 24

7 – 2 Effet du filtre de Kalman suite aux perturbations 25
7 – 2 – 1 Expérience 1 : 4 perturbations 25
7 – 2 – 2 Expérience 2 : 3 perturbations 26

7 – 3 Trajectoire du maître et de l'esclave 28
7 – 3 – 1 Maître immobile 28

7 – 3 – 1 – 1 Expérience 1 28
7 – 3 – 1 – 2 Expérience 2 29
7 – 3 – 2 Maître mobile 30
7 – 3 – 2 – 1 Expérience 1 30
7 – 3 – 2 – 2 Expérience 2 32
7 – 3 – 2 – 3 Expérience 3 33

8 – CONCLUSION **34**

ANNEXE A **35**

ANNEXE B **37**

1 – Introduction

Le robot Khepera est un véhicule autonome muni de deux roues qui a été développé par le Laboratoire des systèmes autonomes de l'EPFL. Il a fait l'objet de plusieurs projets menés au sein du Laboratoire d'automatique de l'EPFL. Contrairement aux études précédentes, notre étude porte sur deux robots. L'objectif était d'implémenter en temps réel sur un Robot « esclave ou prédateur » une technique de contrôle commande qui lui permet de rattraper un deuxième robot « maître » qui se déplace selon une trajectoire aléatoire sur une plate-forme.

Le système de mesure que nous utilisons est celui proposé par Herrera [1]. Il s'agit principalement d'une camera digitale. L'image qu'elle fournit est traitée par un programme de traitement d'image développé sous LabView. Le résultat dudit traitement est l'obtention en temps réel de la position du robot mobile sur la plate-forme. Les positions mesurées sont entachées d'erreurs. C'est la raison pour laquelle nous avons choisi de les filtrer avant de les utiliser dans l'algorithme de commande : pour cela, le filtre de Kalman est utilisé. Les raisons de ce choix seront exposées dans les paragraphes suivants.

Pour commander le robot « esclave », la commande Prédictive a été utilisée.

Dans la suite, nous consacrerons un bref paragraphe à la modélisation du robot Khepera. Le système de mesure et le programme de traitement d'image pour deux robots mobiles feront l'objet d'un autre paragraphe détaillé. Ensuite, il sera question du Filtre de Kalman Etendu puis l'implémentation de la commande prédictive.

2 – Modélisation du robot Khepera

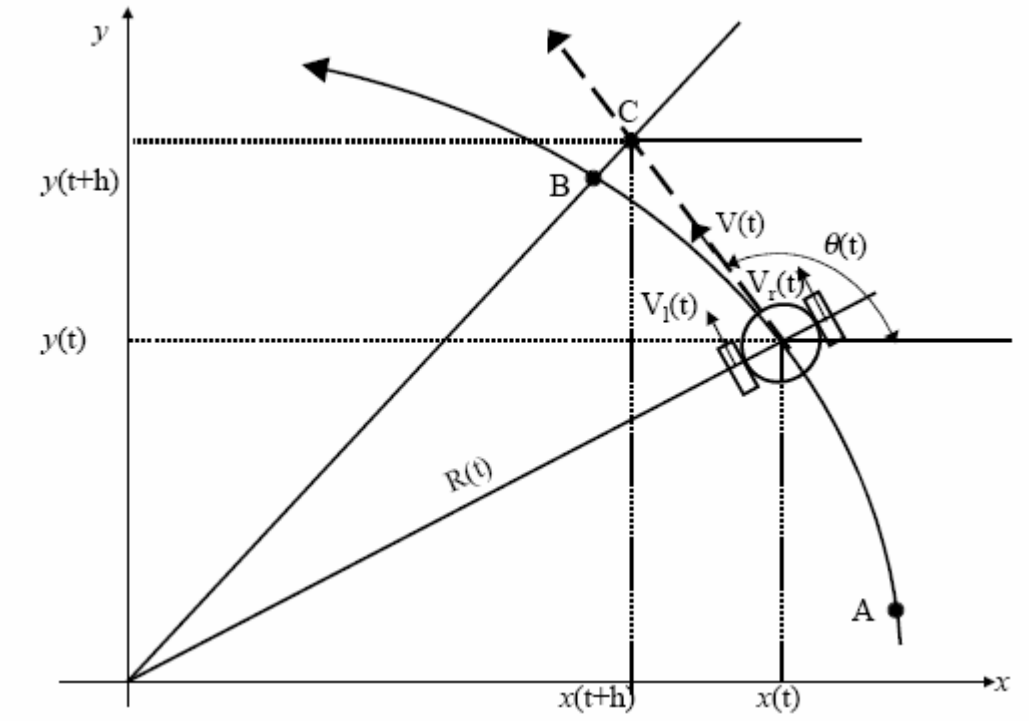
La modélisation d'un système dynamique est une étape très importante pour la conception d'une commande telle que la commande prédictive. La qualité d'un régulateur dépend fortement de celle du modèle du système dynamique qu'on veut commander. Herrera [1] et Niederberger [2] ont proposé deux modèles pour le robot Khepera. Seul celui basé sur une trajectoire rectiligne et qui considère le robot comme un point a été retenu. Il est causal et non linéaire :

$$x(k+1) = x(k) + \frac{h}{2}(V_r(k) + V_l(k))\cos\theta(k)$$

$$y(k+1) = y(k) + \frac{h}{2}(V_r(k) + V_l(k))\sin\theta(k)$$

$$\theta(k+1) = \theta(k) + \frac{h}{L}(V_r(k) - V_l(k))$$

Ce set d'équations montre qu'il s'agit d'un système multivariable qui possède deux entrées $V_r(k)$ et $V_l(k)$ qui sont respectivement les signaux qui permettent d'alimenter les roues droite et gauche du robot. Il y'a trois sorties qui sont $x(k)$, $y(k)$, $\theta(k)$ qui sont respectivement l'abscisse, l'ordonnée et la position angulaire du robot dans le repère définit comme sur la figure ci-dessous :



h est la période d'échantillonnage et L est la distance entre les deux roues du robots. Le modèle d'état du robot est donné ci-dessous :

$$\begin{pmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{pmatrix} = \begin{pmatrix} x(k) \\ y(k) \\ \theta(k) \end{pmatrix} + \underbrace{\begin{pmatrix} \frac{h}{2}cos\theta(k) & \frac{h}{2}cos\theta(k) \\ \frac{h}{2}sin\theta(k) & \frac{h}{2}sin\theta(k) \\ \frac{h}{L} & -\frac{h}{L} \end{pmatrix}}_{\mathbf{B}(\theta(k))} \underbrace{\begin{pmatrix} V_r(k) \\ V_l(k) \end{pmatrix}}_{\mathbf{v}(k)}$$

$$\mathbf{x}_{xy\theta}(k+1) = \mathbf{x}_{xy\theta}(k) + \mathbf{B}(\theta(k))\mathbf{v}(k)$$

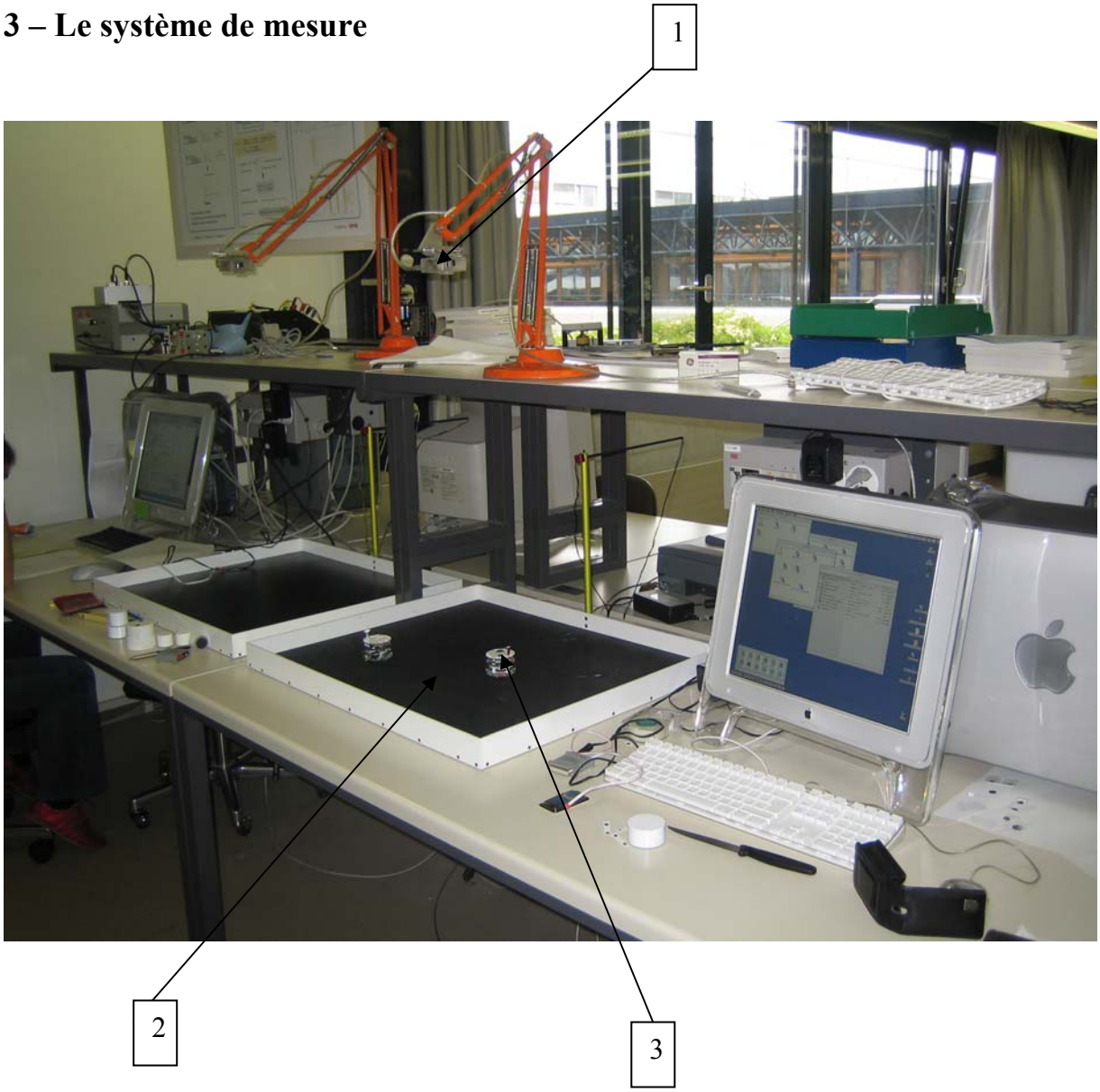
$$\mathbf{x}_{xy\theta}(k) = \begin{pmatrix} x(k) \\ y(k) \\ \theta(k) \end{pmatrix}$$

$$\mathbf{B}(\theta(k)) = \begin{pmatrix} \frac{h}{2}cos\theta(k) & \frac{h}{2}cos\theta(k) \\ \frac{h}{2}sin\theta(k) & \frac{h}{2}sin\theta(k) \\ \frac{h}{L} & -\frac{h}{L} \end{pmatrix}$$

$$\mathbf{v}(k) = \begin{pmatrix} V_r(k) \\ V_l(k) \end{pmatrix}$$

Le programme et les résultats de simulation dans Matlab et Labview du modèle se trouvent en annexe A. Le programme Labview qui permet de simuler le modèle doit être utilisé avec la version 6.1 de Matlab. Avant de l'exécuter, il faut au préalable ouvrir Matlab 6.1

3 – Le système de mesure



1 : camera digitale

2 : plate-forme

3 : robot Khepera

Sur la plate-forme de couleur noire, on peut voir deux robots : l'esclave et le maître. Vu de dessus, le maître est de couleur blanche alors que l'esclave l'est aussi mais avec deux cercles remplis en noire qui permettent au programme de traitement d'image de calculer sa position angulaire. La camera digitale fournit une image en couleur qui sera ensuite traitée par le programme de traitement d'image afin de déterminer les trois états de chaque robot.

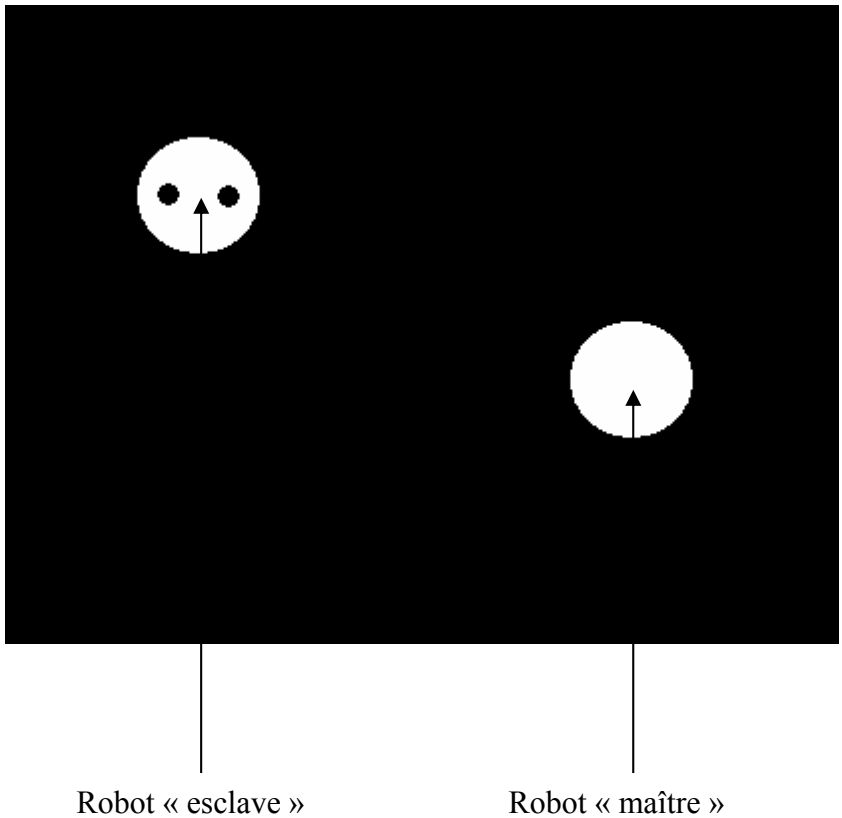
4 – Le Traitement d'image

Nous nous sommes largement inspirés du programme de traitement d'image développé par Herrera [1] qui permet de calculer les trois états x, y, θ d'un seul robot. Le programme que nous avons développé tient compte de la présence de deux robots sur la plate forme.

La procédure de traitement est divisée en plusieurs étapes.

4 – 1 Matrice de pixels couleur et matrice de pixels noirs et blancs

Tout d'abord, l'image en couleur acquise est transformée en une matrice de pixels dont chacun est codé sur 24 bits. Cette matrice est ensuite transformée en une autre matrice toujours codée sur 24 bits mais dont les couleurs sont noires ou blanches. Cette matrice donne une image qui a l'apparence suivante :



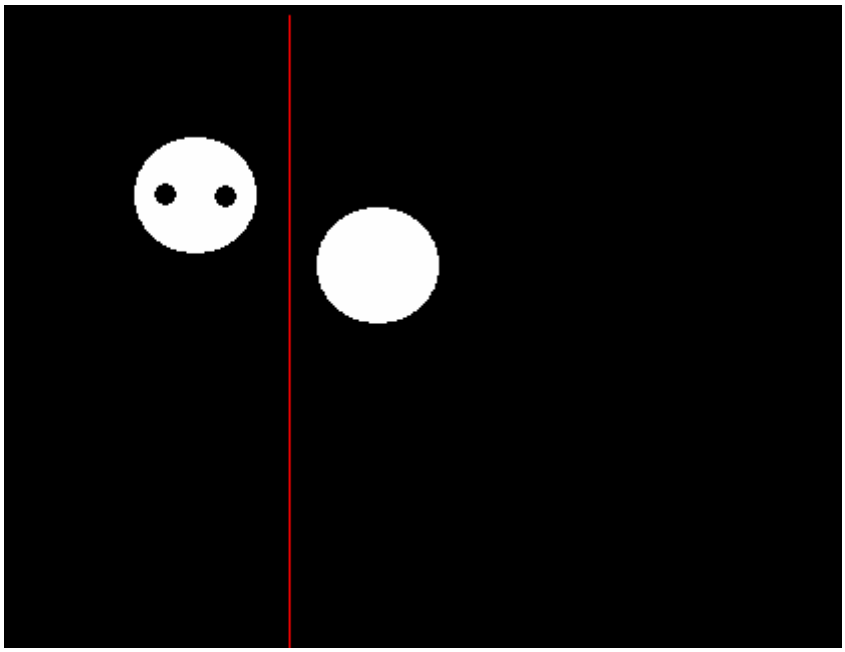
4 – 2 Détermination du centre de gravité des deux robots

Dans l'optique de diviser l'image ci-dessus en deux images distinctes, l'abscisse et l'ordonnée du centre de gravité des deux robots sont calculées.

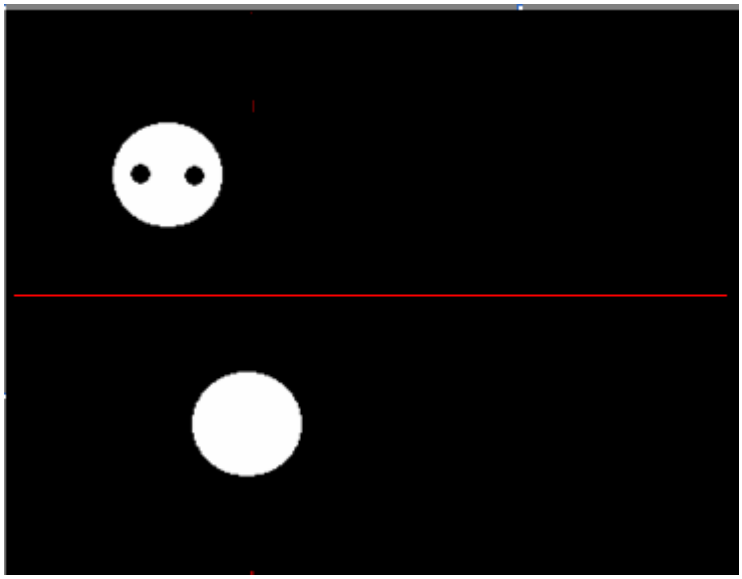
4 – 3 Séparation de l'image

Selon le résultat d'un critère qui porte sur la position d'un robot par rapport à l'autre, une coupe verticale ou horizontale de l'image est effectuée.

Si les deux robots ont par exemple la position suivante, alors une coupe verticale est faite :



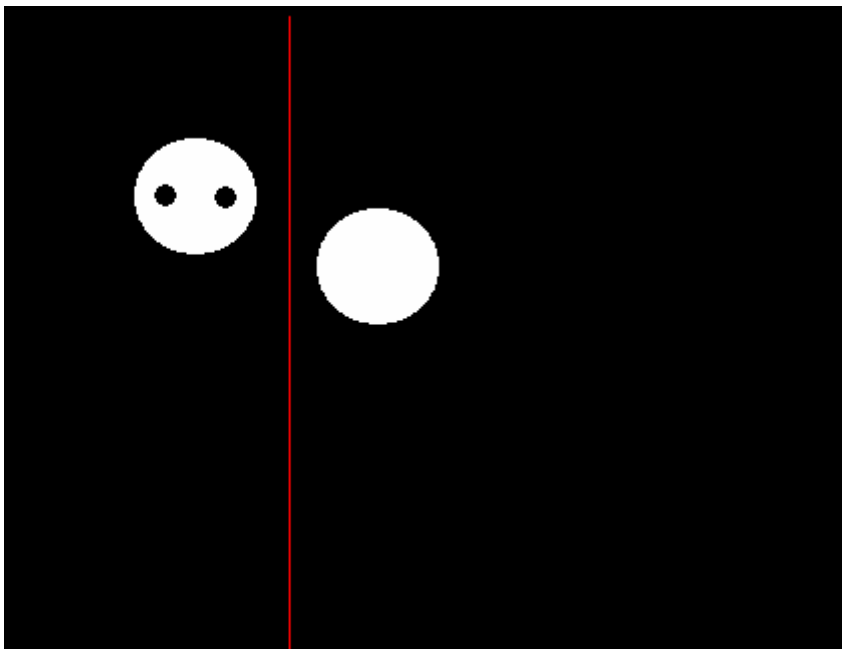
Une coupe horizontale est faite dans le cas suivant :

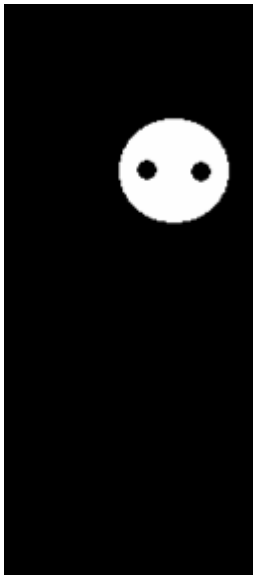


4 – 4 Calcul des positions de chaque robot dans le repère associé à chaque image

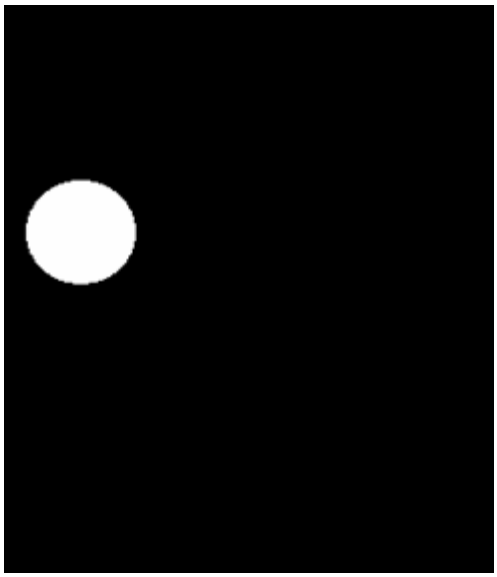
Après avoir effectué les coupes, chaque robot se trouve dans un système d'axe dans lequel on déterminera sa position. Pour le robot « esclave », les trois états seront calculés alors seuls l'abscisse et l'ordonnée du robot « maître »seront estimées.

4 – 4 – 1 Cas de la coupe verticale



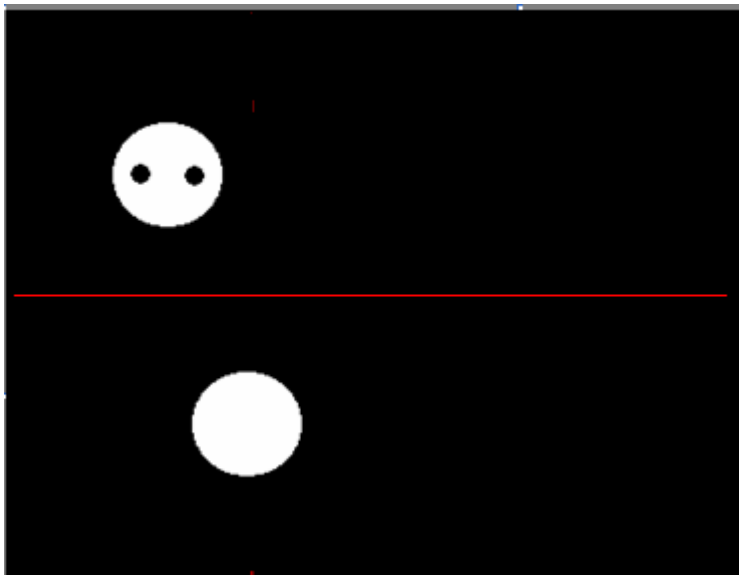


Calcul de x_1, y_1, θ_1



Calcul de x_2, y_2

4 – 4 – 2 Cas de la coupe horizontale



Calcul de x_1, y_1, θ_1

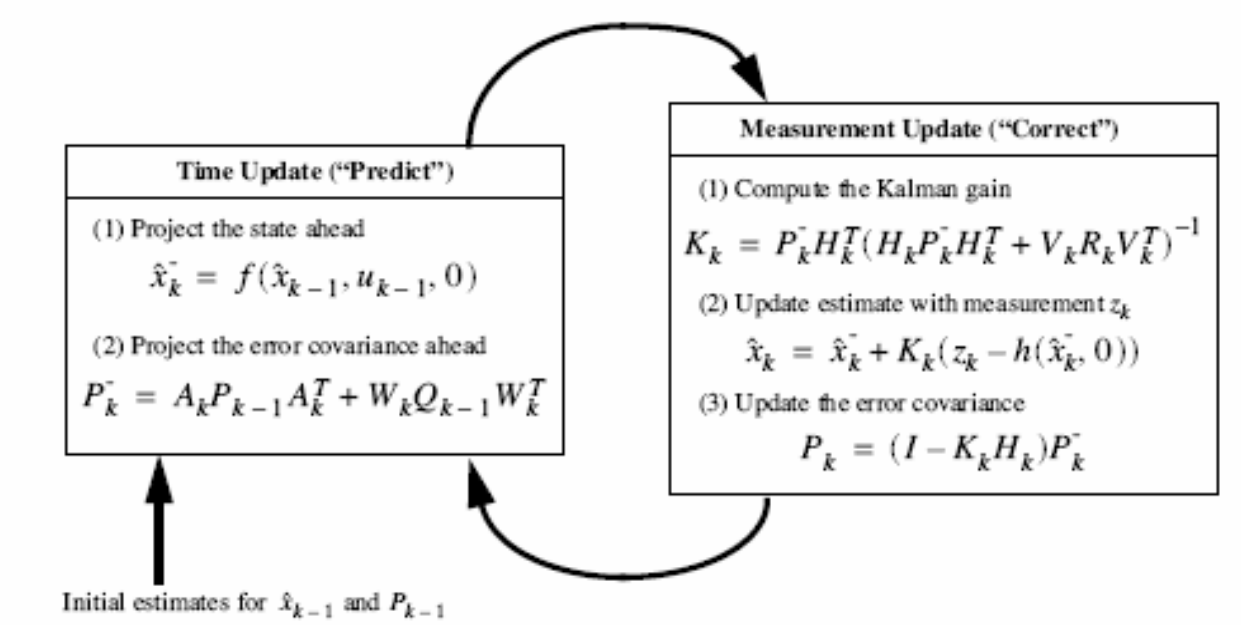
Calcul de x_2, y_2

4 – 5 Test du programme de traitement d'image

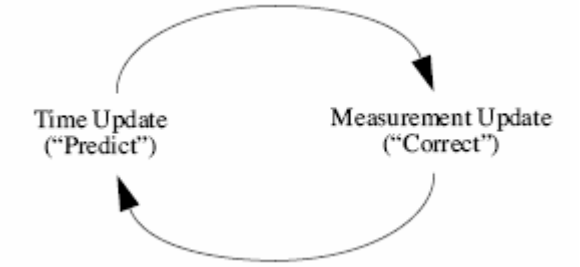
L'acquisition d'image et son traitement prennent un temps relativement grand. En ce qui concerne les résultats fournis par le programme, ils sont « noyés » dans du bruit. C'est la raison pour laquelle nous devons les filtrer. Le filtre de Kalman set bien adapté pour ce type d'expérience.

5 – Le filtre de Kalman Etendu

Le filtre de Kalman a été développé dans les années 1960 par R.E Kalman. Il est constitué d'un set d'équations discrètes qui permettent d'estimer les états d'un système par rapport aux données mesurées. Etant donné que le modèle du système est non linéaire, nous avons choisi le filtre de Kalman Etendu.



L'exécution du filtre de Kalman Etendu se fait en deux étapes et d'une façon cyclique.



5 – 1 La prédiction

Connaissant le modèle d'état du système, on estime l'état a priori du système.

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0)$$

$$\begin{pmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{pmatrix} = \begin{pmatrix} x(k) \\ y(k) \\ \theta(k) \end{pmatrix} + \underbrace{\begin{pmatrix} \frac{h}{2}cos\theta(k) & \frac{h}{2}cos\theta(k) \\ \frac{h}{2}sin\theta(k) & \frac{h}{2}sin\theta(k) \\ \frac{h}{L} & -\frac{h}{L} \end{pmatrix}}_{\mathbf{B}(\theta(k))} \underbrace{\begin{pmatrix} V_r(k) \\ V_l(k) \end{pmatrix}}_{\mathbf{v}(k)}$$

Il faut initialiser l'exécution avec le vecteur d'état nul c'est-à-dire $\hat{x}_0 = \begin{pmatrix} x=0 \\ y=0 \\ \theta=0 \end{pmatrix}$.

Le vecteur $u_{k-1} = \begin{pmatrix} v_r^{k-1} \\ v_l^{k-1} \end{pmatrix}$ est fourni en temps réel par le robot « esclave » qui se déplace.

Pour calculer P_K^-, il faut initialiser P_{k-1} par $P_0 = \begin{pmatrix} 1,0,0 \\ 0,1,0 \\ 0,0,1 \end{pmatrix}$ la matrice identité de dimension 3.

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$$

$$f = \begin{pmatrix} x + \frac{h}{2}\cos\theta * v_r + \frac{h}{2} * \cos\theta * v_l \\ y + \frac{h}{2} * \sin\theta * v_r + \frac{h}{2} * \sin\theta * v_l \\ \theta + \frac{h}{L} * v_r - \frac{h}{L} * v_l \end{pmatrix}$$

$$X = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

$$u = \begin{pmatrix} v_r \\ v_l \end{pmatrix}$$

$$A_k = \begin{pmatrix} 1,0, -1/2 * h * \sin(\theta(k)) * v_r(k) - 1/2 * h * \sin(\theta(k)) * v_l(k) \\ 0,1, \ 1/2 * h * \cos(\theta(k)) * v_r(k) + 1/2 * h * \cos(\theta(k)) * v_l(k) \\ 0,0,1 \end{pmatrix} = jacobien(f, X)$$

$$W_k = \begin{pmatrix} 1,0,0 \\ 0,1,0 \\ 0,0,1 \end{pmatrix}$$

$Q_k = \begin{pmatrix} \sigma_1^2, 0, 0 \\ 0, \sigma_2^2, 0 \\ 0, 0, \sigma_3^2 \end{pmatrix}$ Matrice constante obtenue par tuning qui modélise le bruit du système.

5 – 2 La correction des mesures

On calcule le gain de Kalman K_k par l'expression suivante :

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1}$$

$H_k = \begin{pmatrix} 1, 0, 0 \\ 0, 1, 0 \\ 0, 0, 1 \end{pmatrix}$

$V_k = \begin{pmatrix} 1, 0, 0 \\ 0, 1, 0 \\ 0, 0, 1 \end{pmatrix}$

$R_k = \begin{pmatrix} \sigma_x^2, 0, 0 \\ 0, \sigma_y^2, 0 \\ 0, 0, \sigma_\theta^2 \end{pmatrix}$ Matrice constante qui modélise le bruit sur les mesures.

Mise à jour de l'état du système sur la base des mesures $z_k = \begin{pmatrix} x_{mesurée} \\ y_{mesurée} \\ \theta_{mesurée} \end{pmatrix}$:

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-, 0))$$

Mise à jour de l'erreur de covariance :

$$P_k = (I - K_k H_k) P_k^-$$

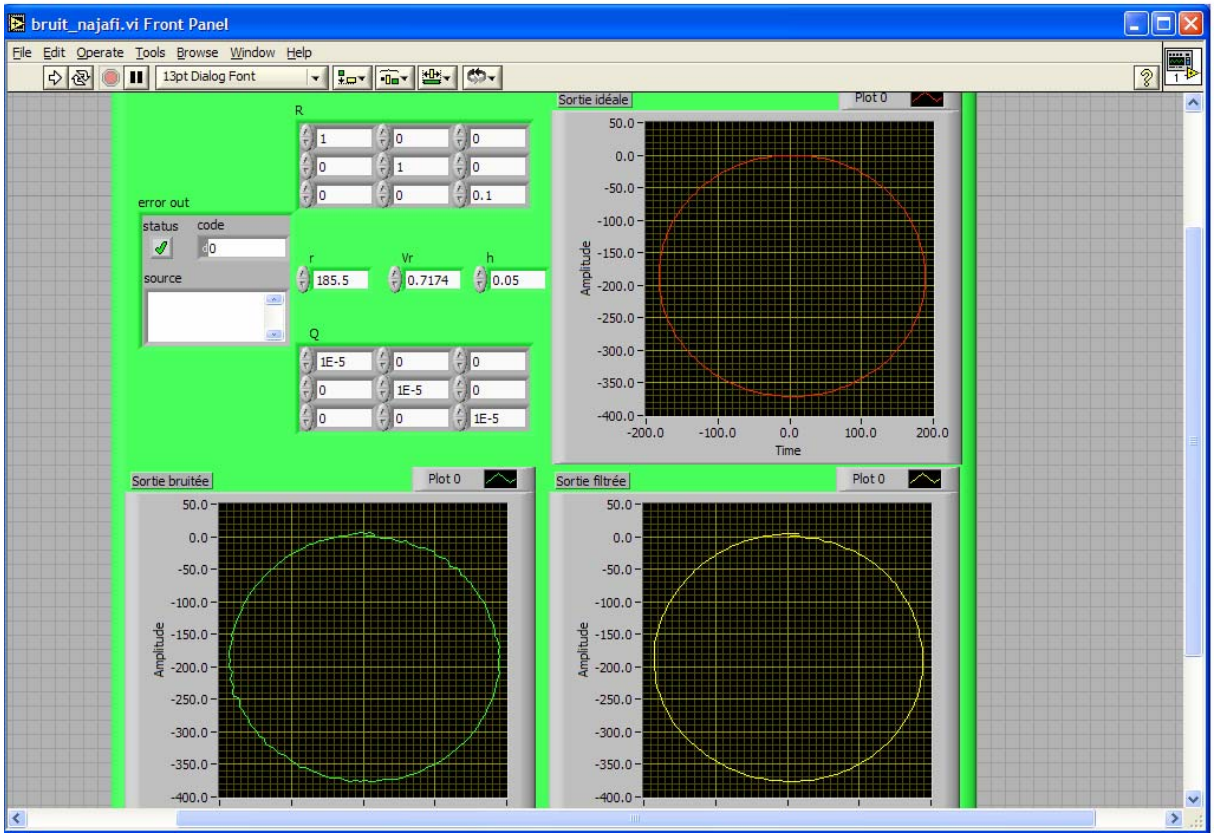
$I = \begin{pmatrix} 1, 0, 0 \\ 0, 1, 0 \\ 0, 0, 1 \end{pmatrix}$

5 – 3 Simulation du filtre de Kalman sous Matlab et Labview

Nous avons simulé le modèle avec un bruit blanc ce qui caractérise les mesures. Ensuite l'algorithme du filtre de Kalman Etendu a été utilisé pour filtrer ces mesures.

Le programme et les résultats de la simulation se trouvent en annexe B. Avant d'exécuter le programme sous Labview, il faut ouvrir Matlab 6.1.

Face avant



5 – 4 Implantation du filtre de Kalman en temps réel

Après avoir obtenu des bons résultats lors de la simulation, nous avons implanté filtre de Kalman Etendu afin qu'il puisse marché en temps réel. Nous avons remarqué deux problèmes :

- L'algorithme utilise comme état initial le vecteur $\hat{x}_0 = \begin{pmatrix} x=0 \\ y=0 \\ \theta=0 \end{pmatrix}$. Avant la mise en marche du robot, il occupe une position qui différente de celle du vecteur initial. Il faut donc attendre un certain temps pour que le résultat converge vers la vraie position du robot sur la plate-forme avant de mettre le robot en marche. Sinon les résultats seront faussés.
- La période d'échantillonnage utilisée pour la simulation n'est identique à celle de Labview. Nous l'avons d'abord estimée. Elle est environ 140ms.

Pour vérifier la qualité du filtre implanté, nous avons réalisé deux expériences :

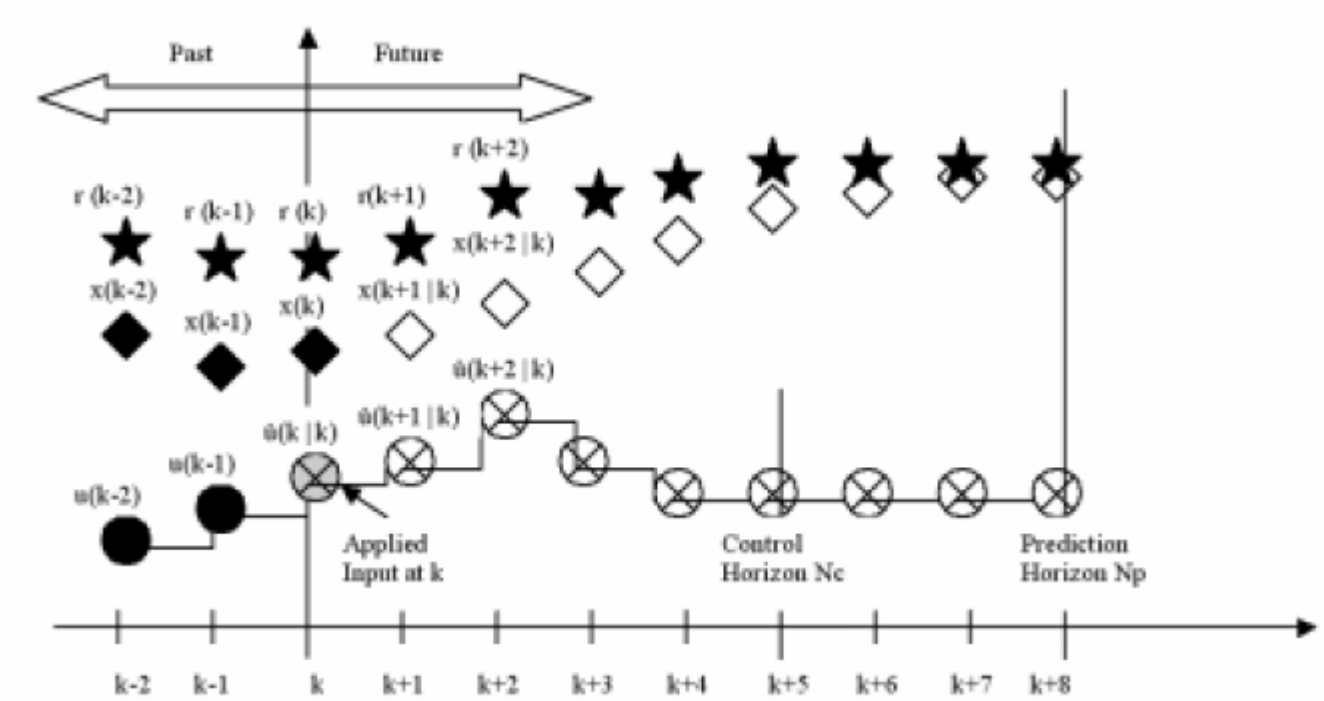
- le robot « esclave » décrivant une trajectoire rectiligne
- le robot « esclave » décrivant une trajectoire circulaire

Les résultats de ces deux expériences se trouvent en annexe B.

6 – La commande prédictive

Pour que le robot « esclave » puisse rattraper le robot « maître », nous avons opté d'utiliser la commande prédictive. Ce choix a été motivé par les études précédemment faites par Niederberger [2] et Herrera [1].

En fait, la commande prédictive consiste à trouver sur un horizon donné, la commande à appliquer sur un système et qui minimiserait une fonction de coût. Cette dernière dépend de la trajectoire de référence $r(k)$, de l'état du système $x(k)$ et de la grandeur de réglage $u(k)$.



La fonction de coût que nous avons choisie est de la forme suivante :

$$\mathbf{J_1} = (\mathbf{r} - \hat{\mathbf{x}})^{\mathbf{T}}\mathbf{Q}(\mathbf{r} - \hat{\mathbf{x}}) + \hat{\mathbf{u}}^{\mathbf{T}}\mathbf{\Lambda}\hat{\mathbf{u}}$$

Connaissant le modèle du système, le prédicteur est :

$$\hat{\mathbf{x}} = \mathbf{Z}x(k) + \mathbf{F}u(k-1)$$

La commande qui minimise la fonction de coût est la suivante :

$$u(k) = \mathbf{k^T}(\mathbf{r} - \hat{\mathbf{x}}^0)$$

$$\mathbf{K} = (\mathbf{F^T Q F} + \mathbf{\Lambda})^{-1}\mathbf{F^T Q} \in \Re^{m(N_c+1)\times nN_p}$$

$$\hat{\mathbf{x}}^0 = \mathbf{Z}x(k) \in \Re^{nN_p}$$

$$\mathbf{k}^T = [\mathbf{I}_{m\times m} \mid \mathbf{0}_{m\times mN_c}]\mathbf{K}$$

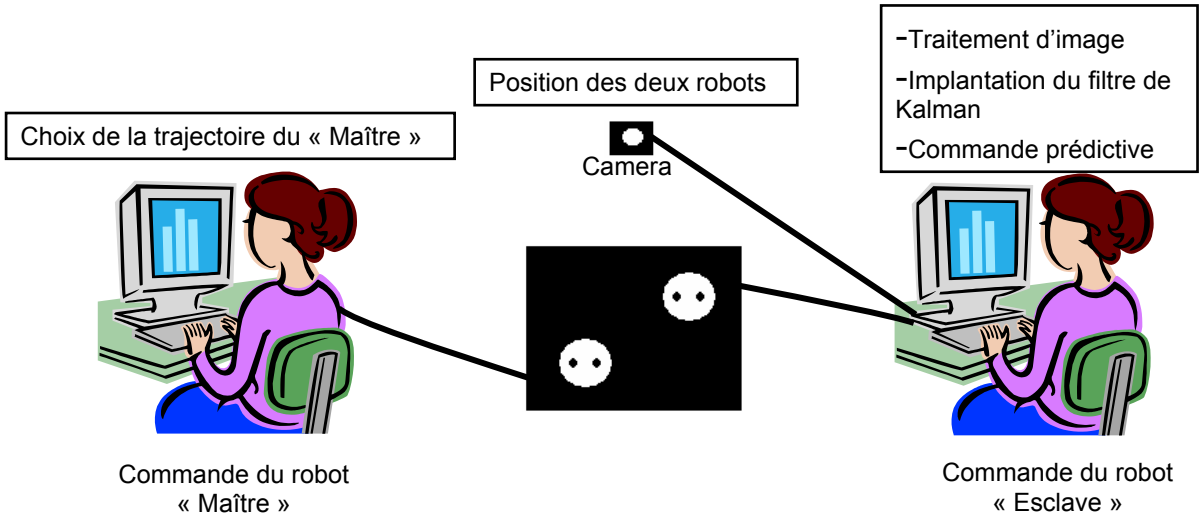
Avec :

$$\mathbf{r} = \begin{pmatrix} r(k+1) \\ \cdots \\ r(k+N_p) \end{pmatrix} \in \Re^{nN_p}$$

$$\hat{\mathbf{u}} = \begin{pmatrix} \hat{u}(k) \\ \hat{u}(k+1) \\ \cdots \\ \hat{u}(k+N_c) \end{pmatrix} \in \Re^{m(N_c+1)}$$

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{x}(k+1) \\ \cdots \\ \hat{x}(k+N_p) \end{pmatrix} \in \Re^{nN_p}$$

6 – 1 Schéma de commande



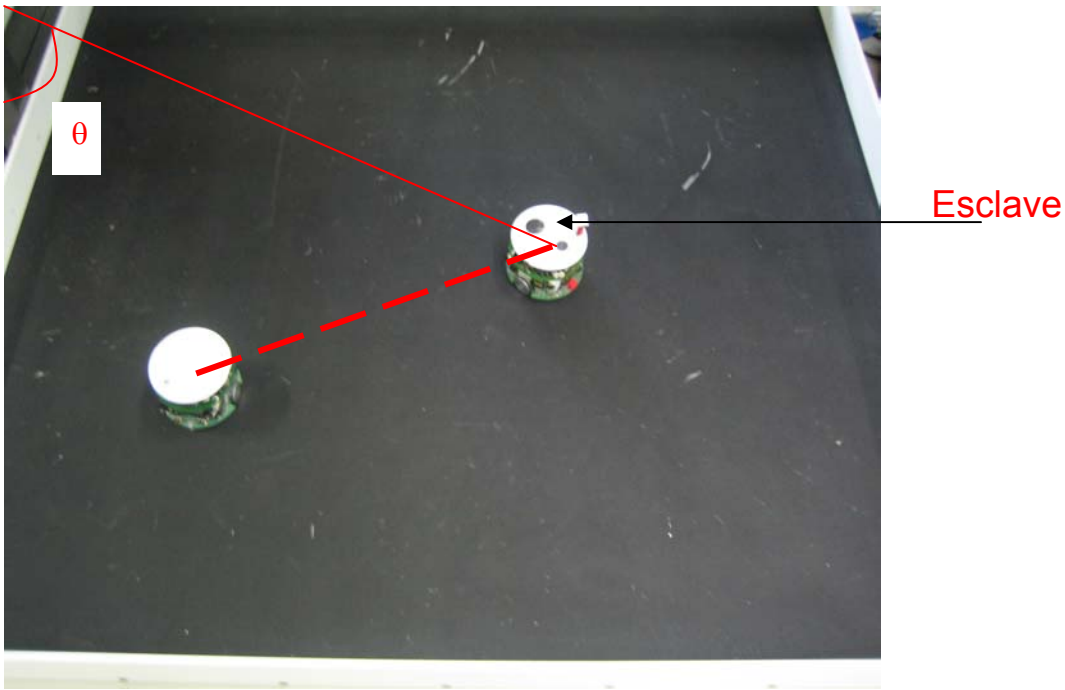
Le PC1 permet de commander le robot « maître » afin qu'il puisse se déplacer sur la plate-forme en décrivant une trajectoire aléatoire. La camera est relié au PC2.Sur ce dernier, est installé le programme de traitement d'image qui fournit en temps réel les positions des deux robots mobiles. Le filtre de Kalman Etendu y est aussi utilisé pour filtrer les mesures de position du robot « esclave ». Enfin la commande prédictive y est aussi implantée pour le robot « esclave ».

6 – 2 Génération de la trajectoire de référence

Pour utiliser la commande prédictive, il faut connaître la trajectoire de référence. Etant donné que les deux robots sont mobiles, cette trajectoire doit être générée après chaque période d'échantillonnage. A cause de cette trajectoire variable d'une période d'échantillonnage à l'autre, on pourrait dire qu'il ne s'agit plus de la commande prédictive classique.

La génération de cette trajectoire se fait en plusieurs étapes :

- Grâce à la camera digitale et au programme de traitement d'image, les positions des deux robots sont utilisées pour estimer en temps réel la distance qui les sépare.
- Ensuite on choisit la vitesse à laquelle le robot « esclave » doit se déplacer pour rattraper le robot « maître ».
- Connaissant la distance entre les deux robots et la vitesse de déplacement, on peut estimer le temps de parcours nécessaire au robot « esclave » pour rattraper le robot « maître ».
- Ce temps de parcours est ensuite converti en nombre d'échantillons.
- Par interpolation linéaire, on génère une trajectoire partant de la position de l'esclave à celle du maître.



6 – 3 Les paramètres de la commande prédictive implantée

Nous avons choisi un horizon de control $N_c = 5$ et un horizon de prédiction $N_p = 6$. Les matrices Q et Λ sont des matrices identité.

7 – Tests expérimentaux

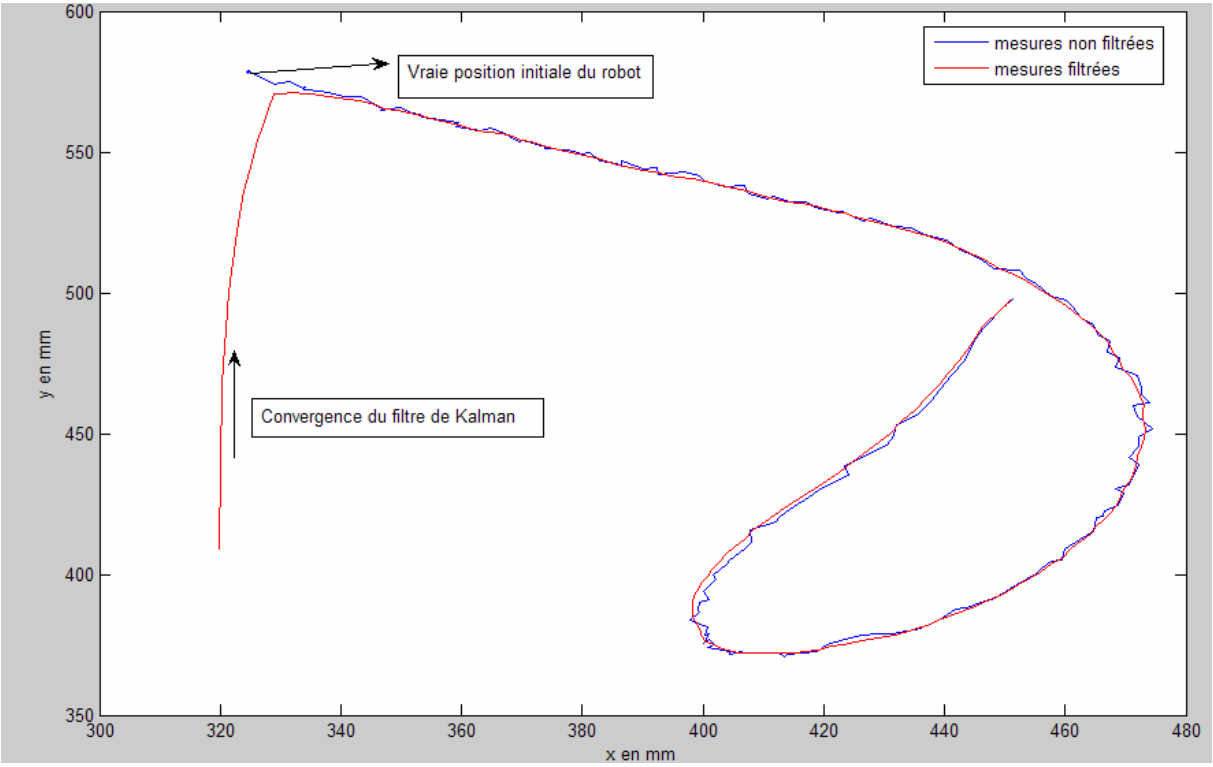
L'esclave doit rattraper le maître quelques soit la trajectoire que ce dernier décrit. L'esclave ne doit pas percuter le maître. Le robot « esclave » s'arrête lorsque la distance qui sépare les centres de gravité des deux robots est égale à 5 fois L (distance qui sépare les deux roues d'un robot).

7 – 1 Test de convergence du filtre de Kalman Etendu

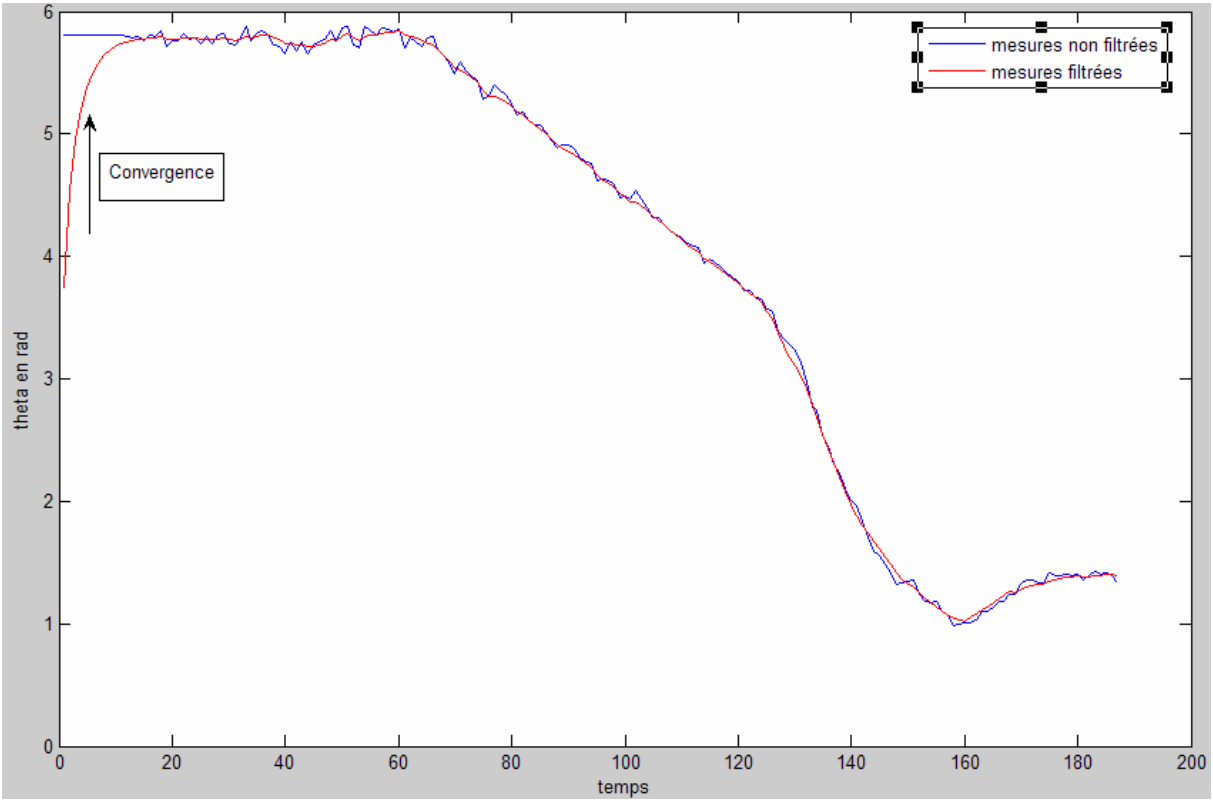
L'algorithme du filtre de Kalman Etendu que nous avons implanté a été initialisé avec le vecteur $(x = 0, y = 0, \theta = 0)$. Expérimentalement, le robot « esclave » peut être initialement positionné en n'importe quel point sur la plate-forme. C'est la raison pour laquelle avant de démarrer l'expérience, on laisse d'abord quelques secondes afin que le filtre de Kalman converge vers la vraie position initiale du robot.

7 – 1 – 1 Expérience 1

- **Trajectoire d'un robot**

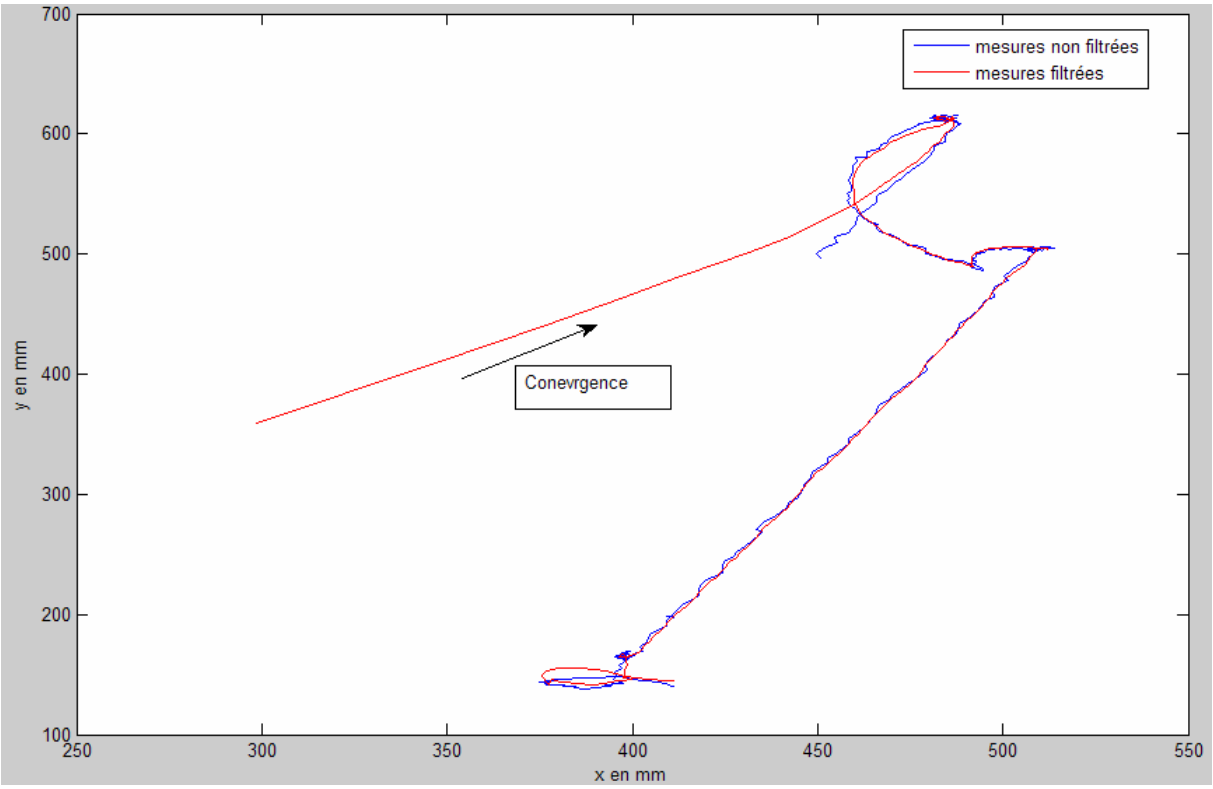


- **Angle θ**

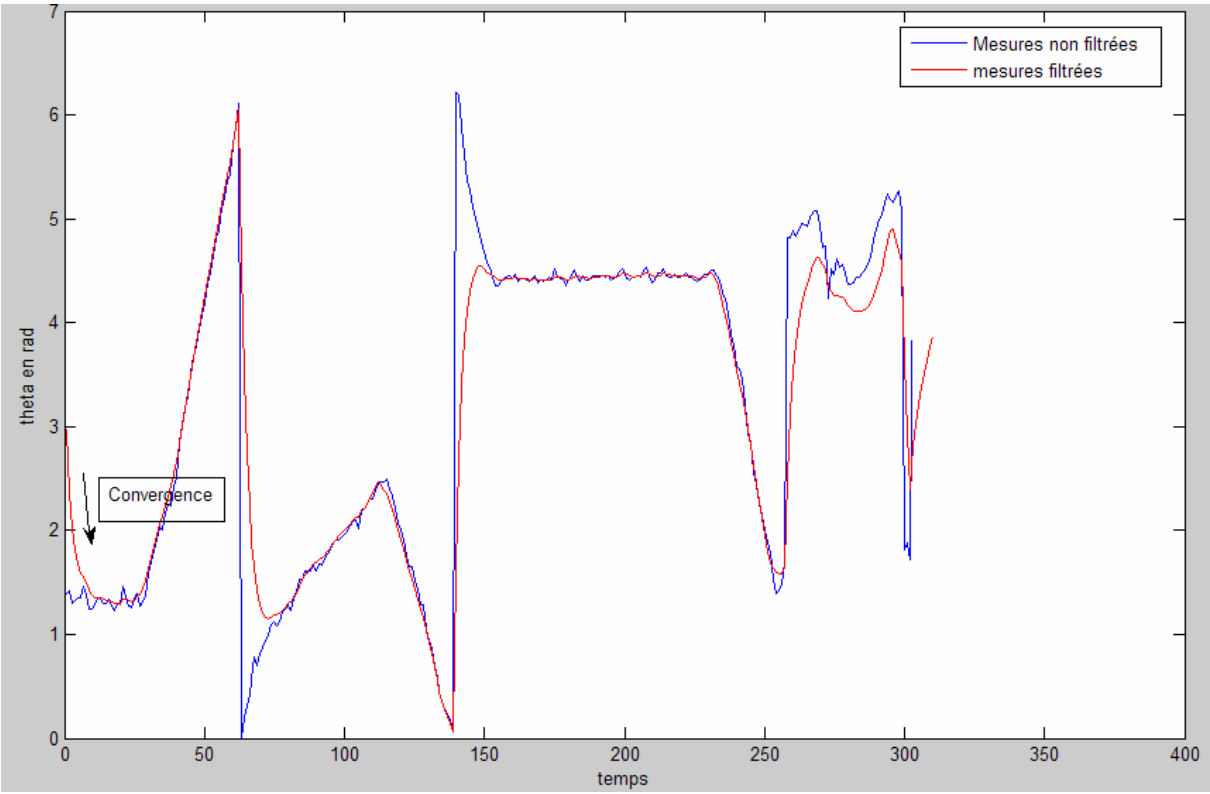


7 – 1 – 2 Expérience 2

- **Trajectoire d'un robot**



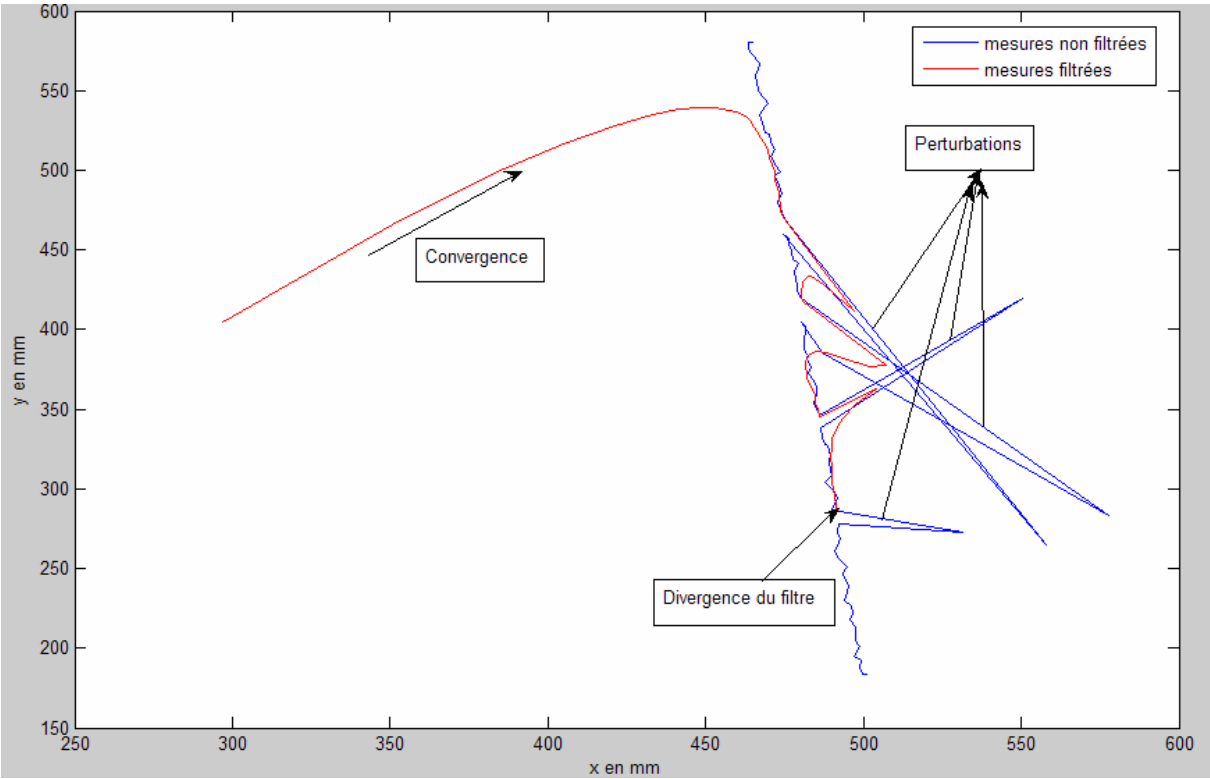
- **Angle θ**



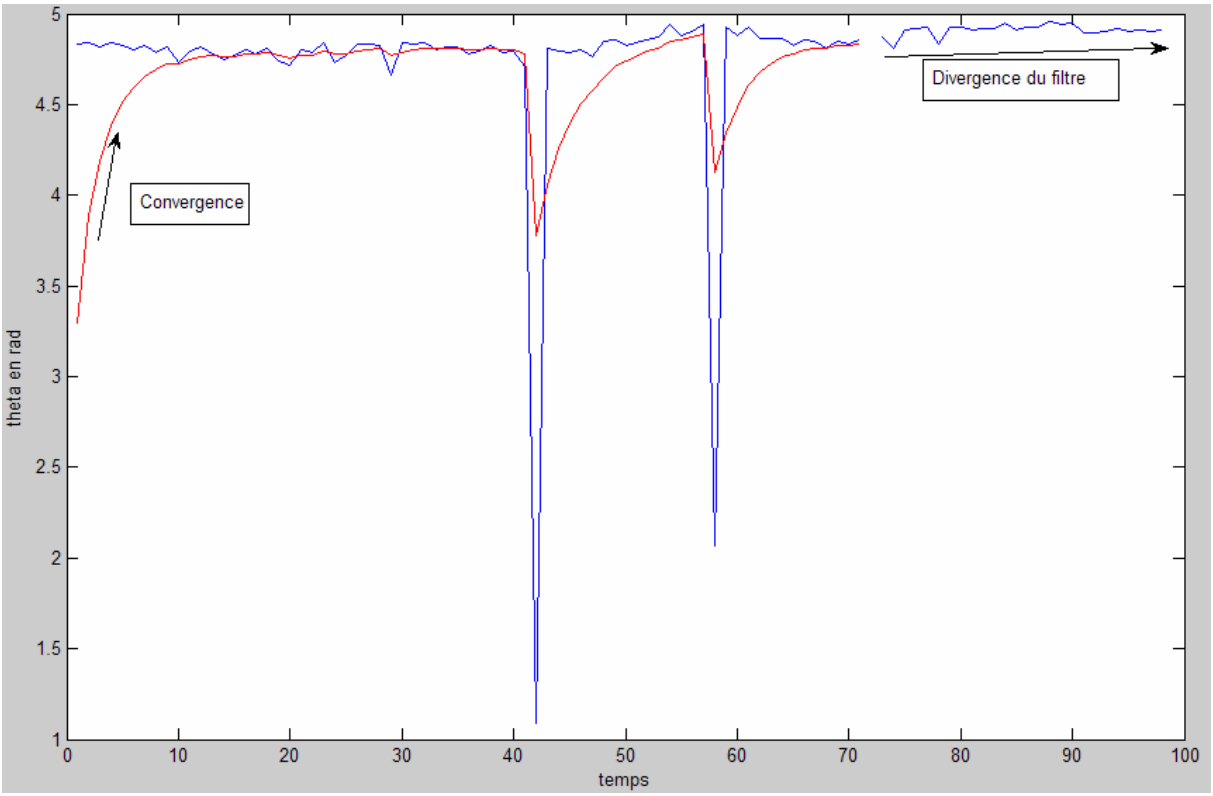
7 – 2 Effet du filtre de Kalman suite aux perturbations

7 – 2 – 1 Expérience 1 : 4 perturbations

- **Trajectoire d'un robot**

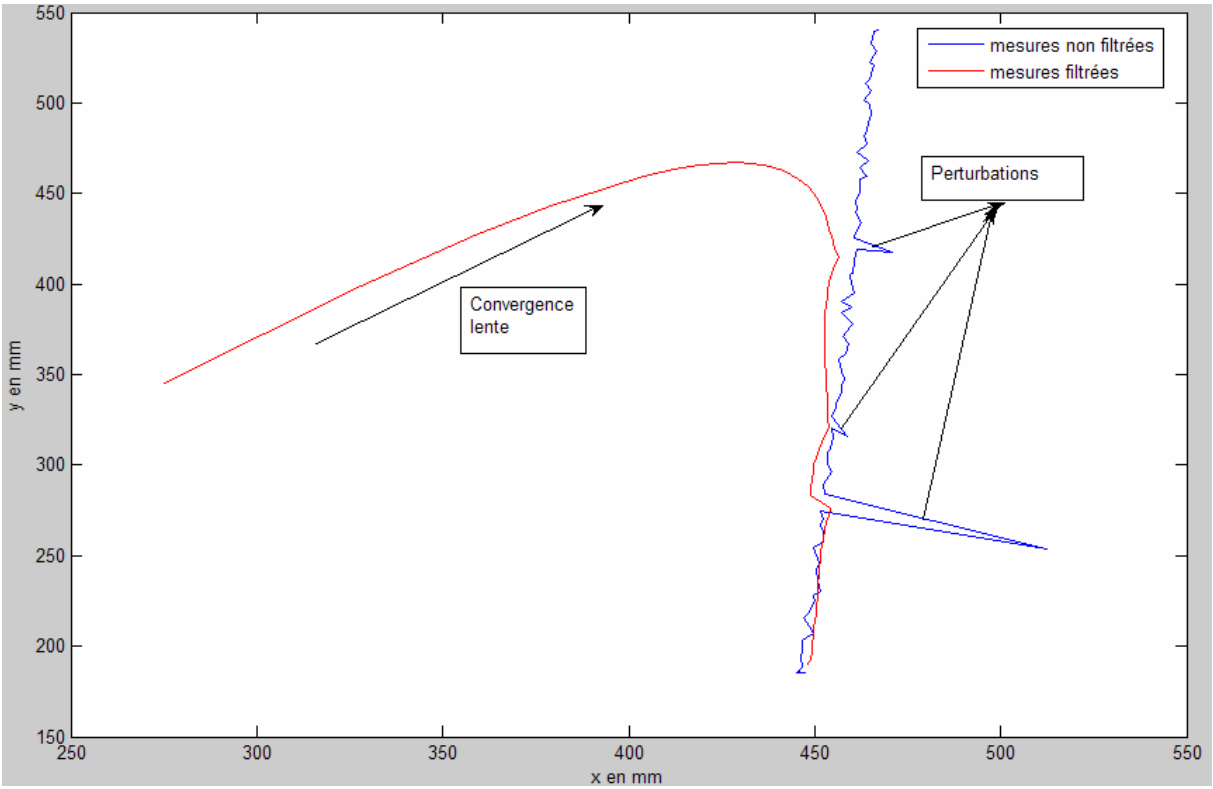


- **Angle θ**

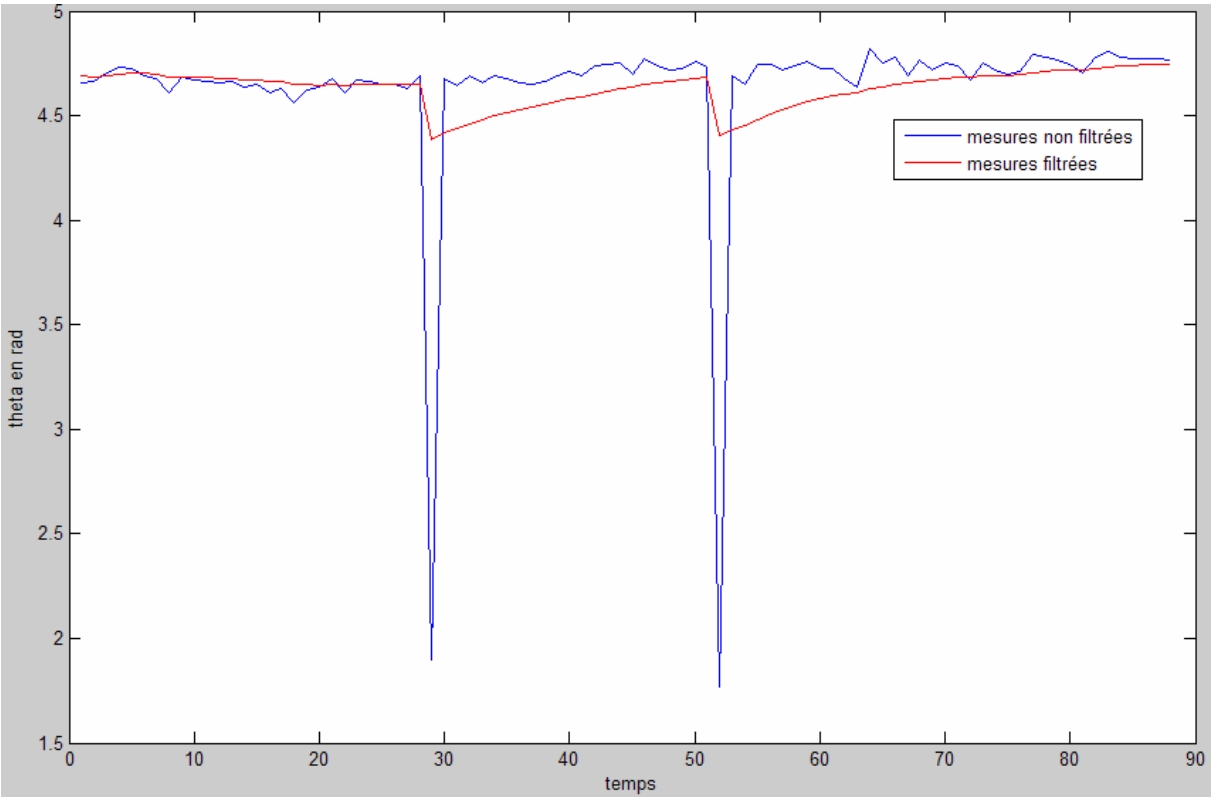


7 – 2 – 2 Expérience 2 : 3 perturbations

- **Trajectoire d'un robot**



- **Angle θ**



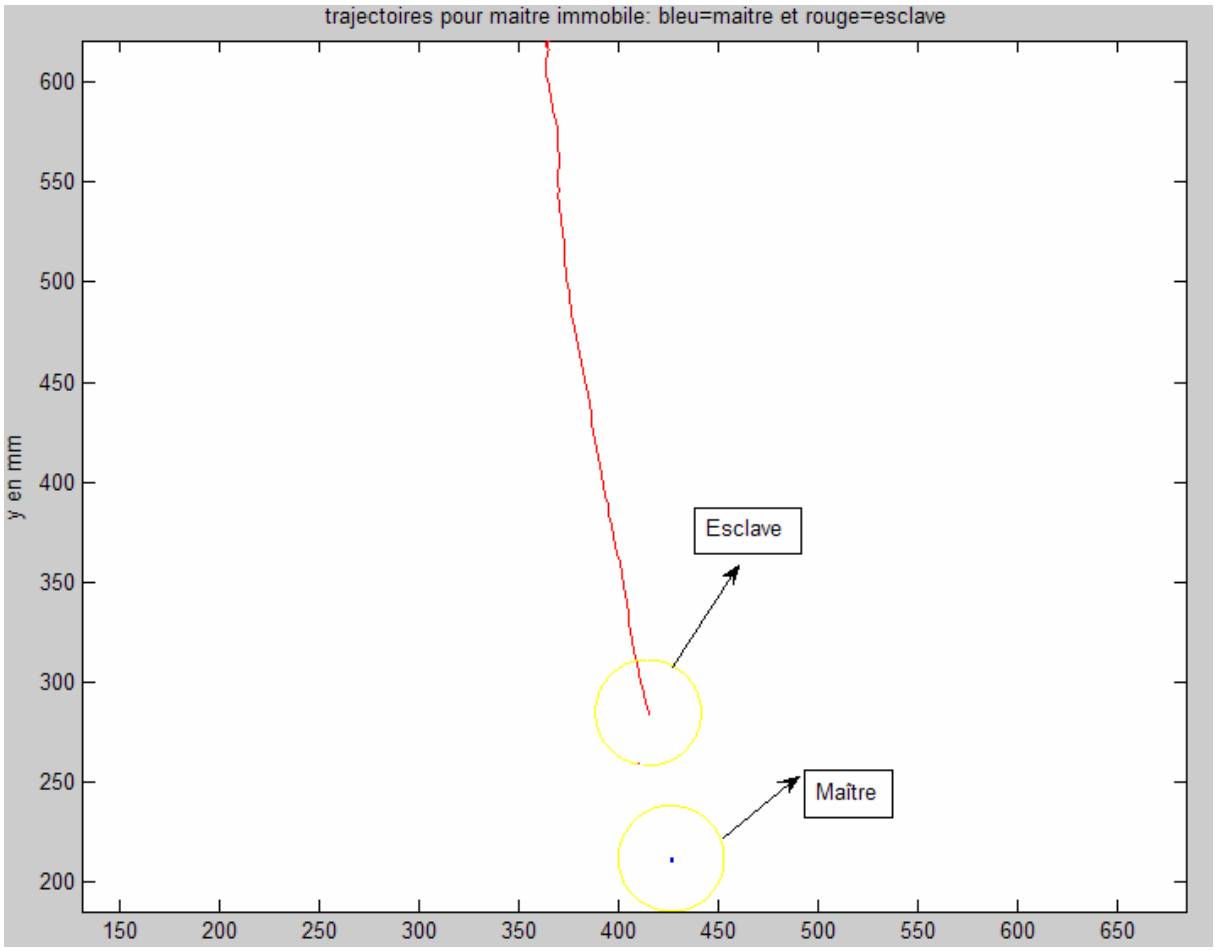
L'expérience 2 a été réalisée avec les valeurs des matrices de covariance différentes de celles de l'expérience 1: Pour la seconde expérience, les éléments diagonaux de la matrice Q sont 10 fois inférieures à celle de la première expérience. Le filtre de Kalman de la première expérience est moins robuste aux perturbations. Alors qu'il atténue quasiment bien les perturbations lors de la seconde expérience.

7 – 3 Trajectoire du maître et de l'esclave

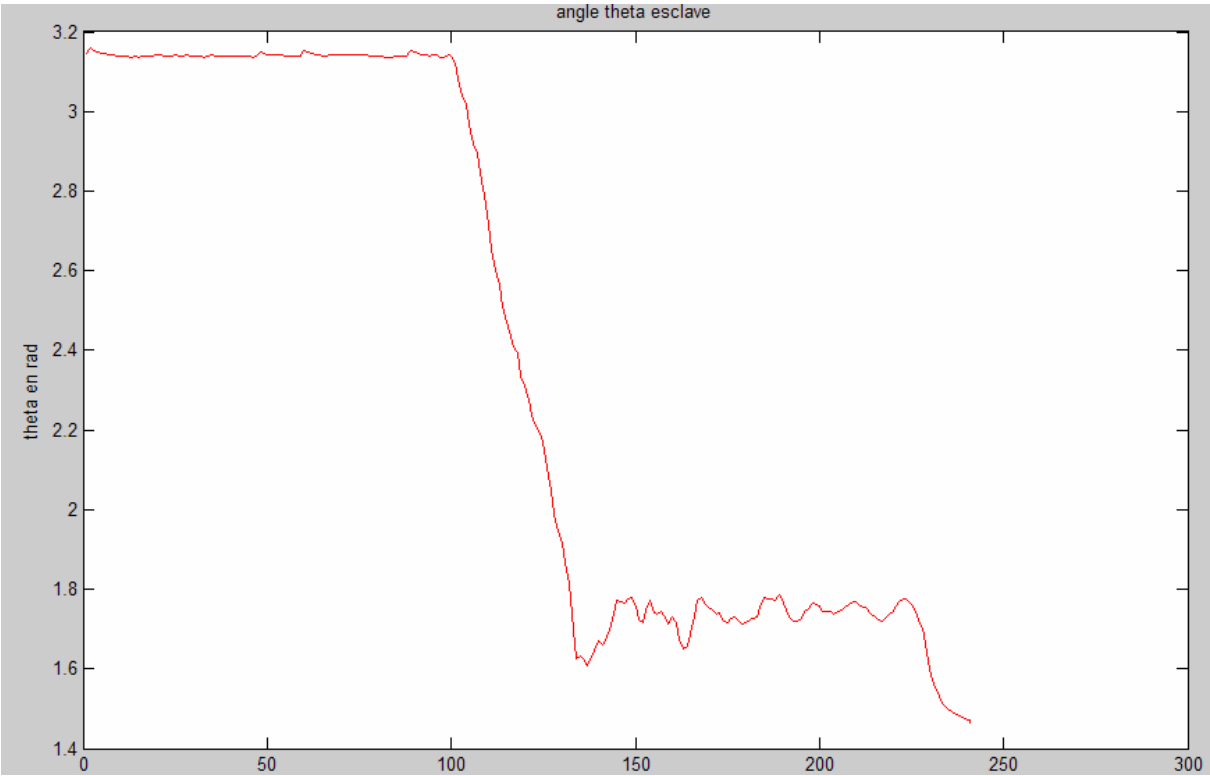
7 – 3 – 1 Maître immobile

7 – 3 – 1 – 1 Expérience 1

- **Trajectoire**

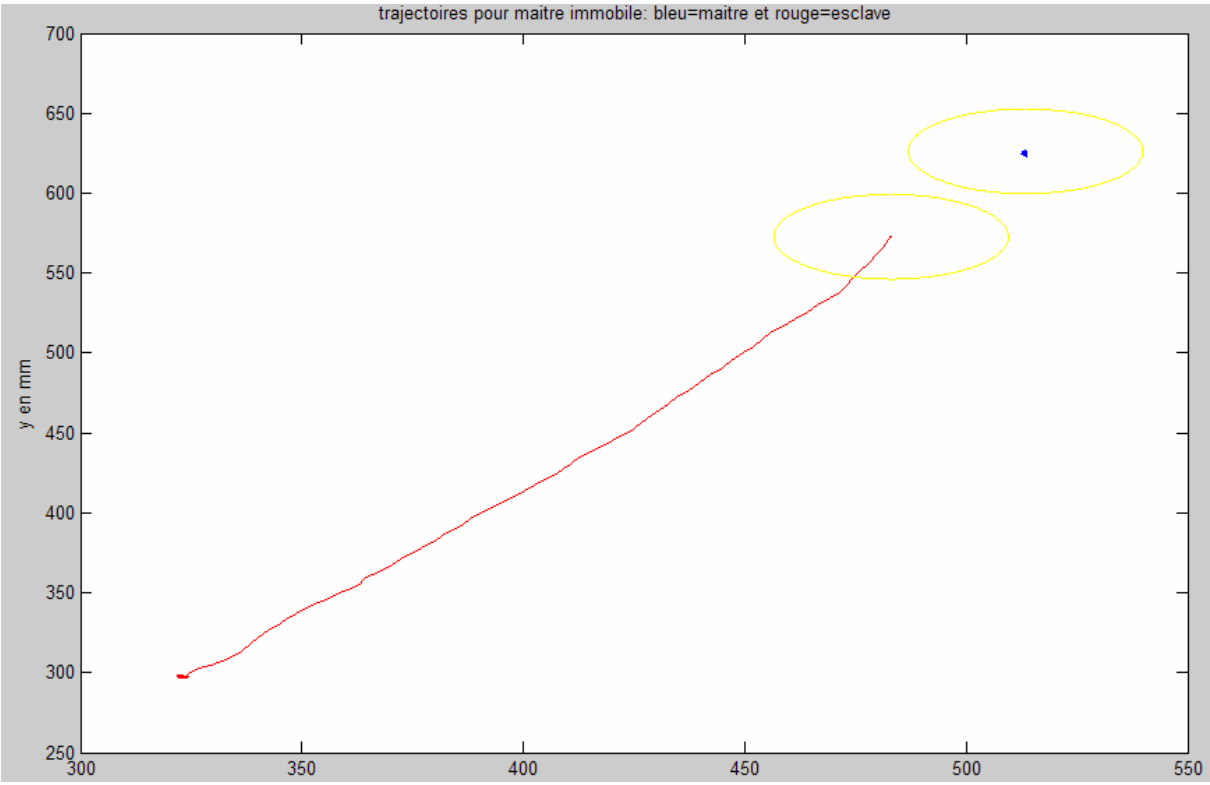


- **Angle θ**

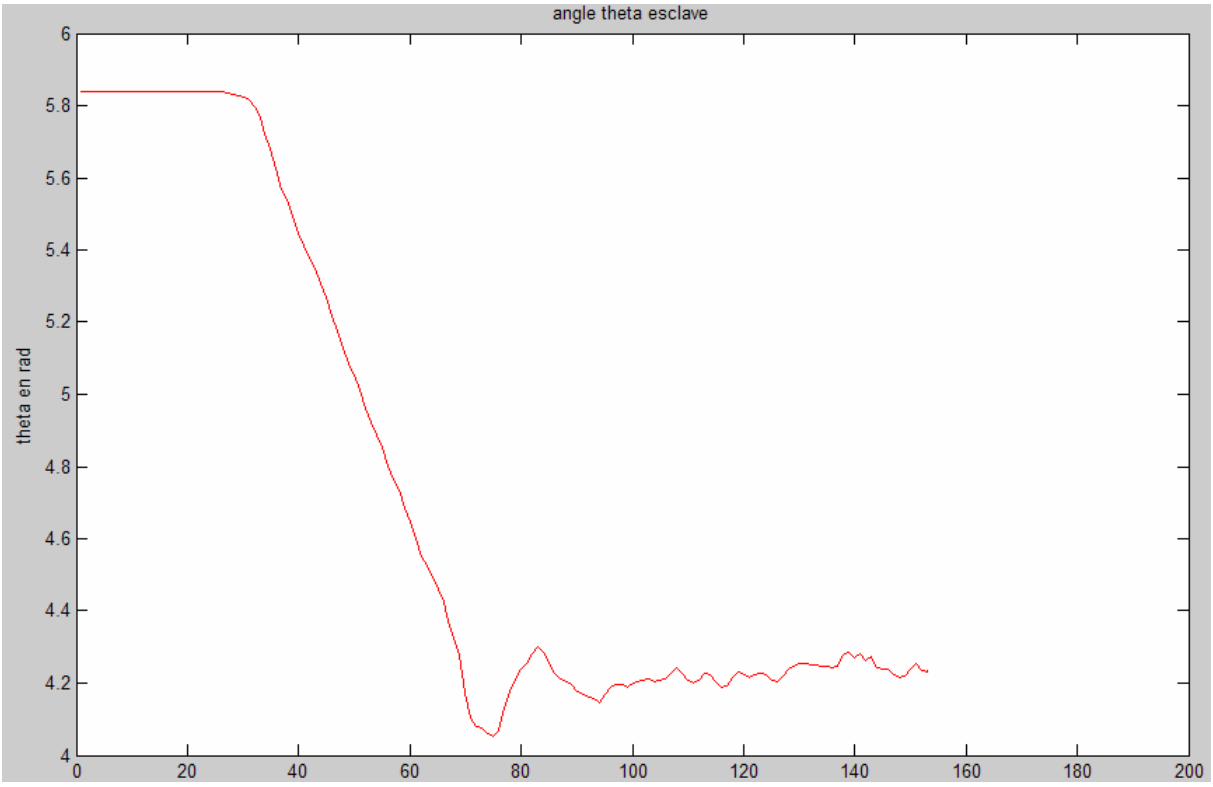


7 – 3 – 1 – 2 Expérience 2

- **Trajectoire**



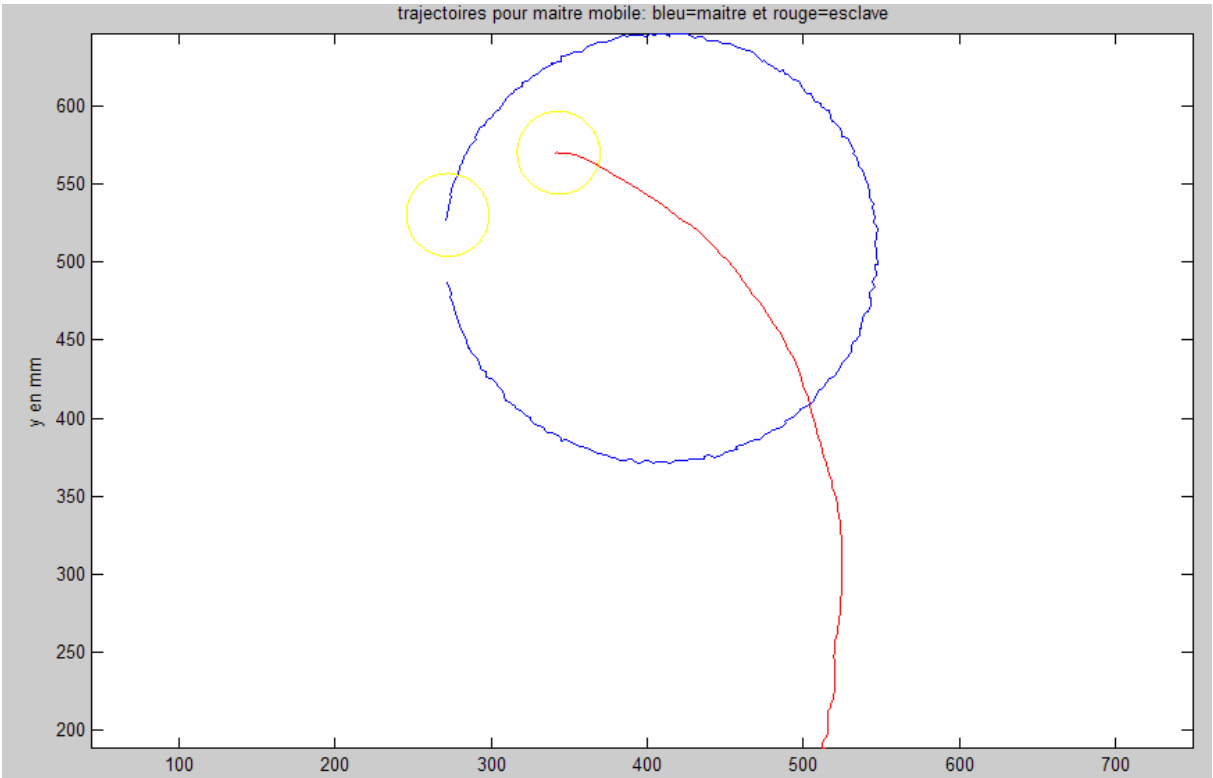
- **Angle θ**



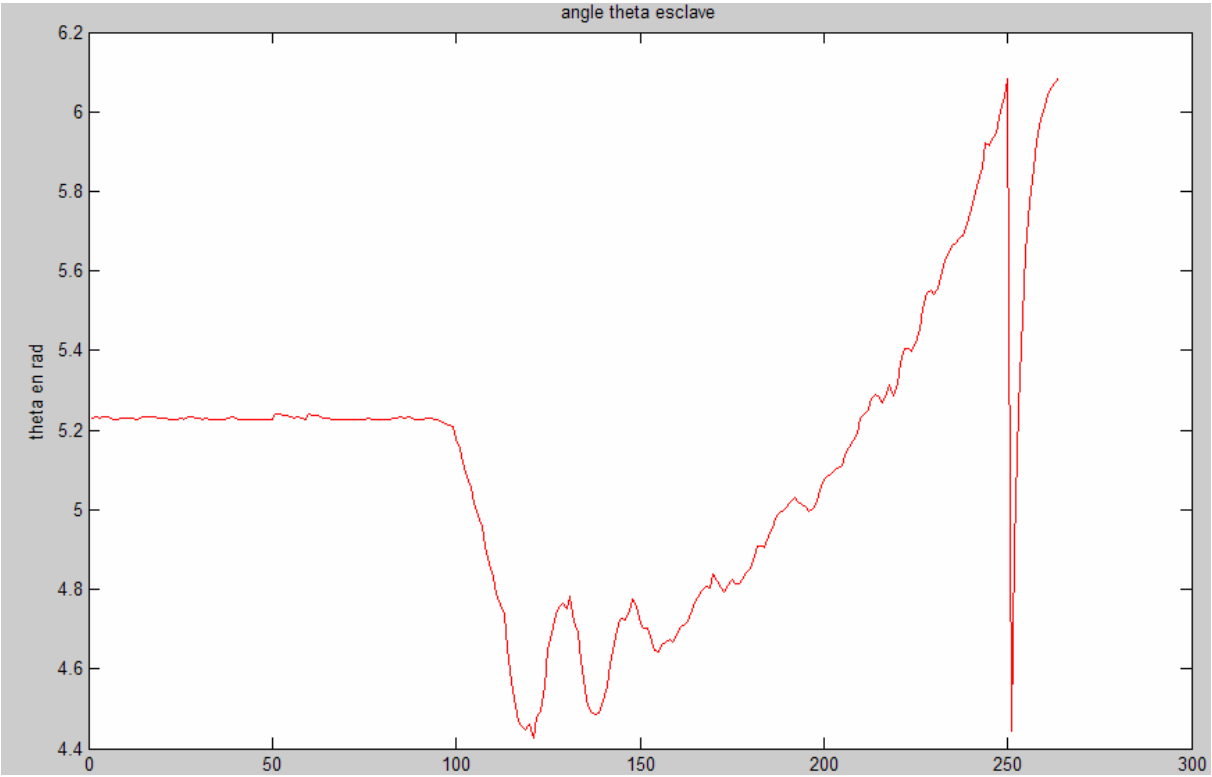
7 – 3 – 2 Maître mobile

7 – 3 – 2 – 1 Expérience 1

- **Trajectoire**



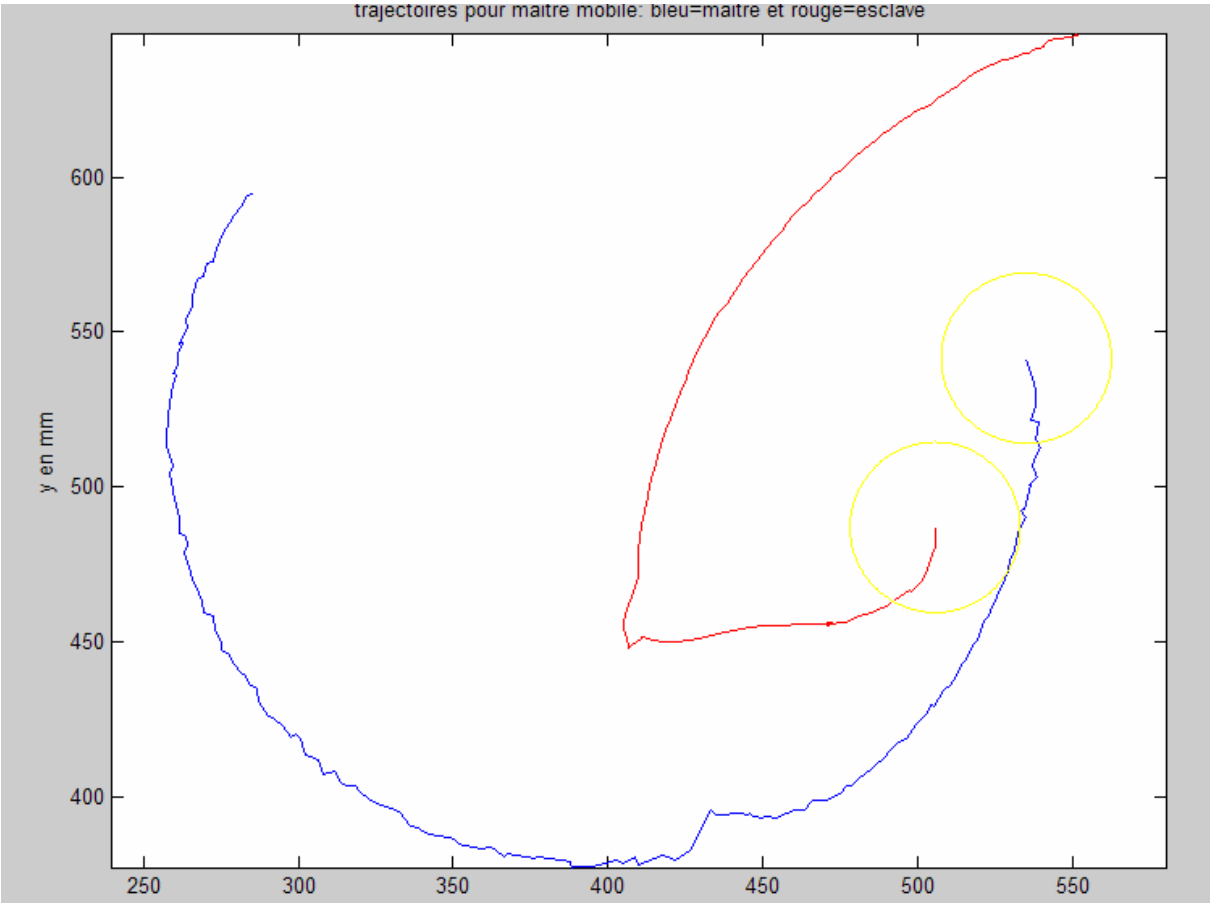
- **Angle θ**



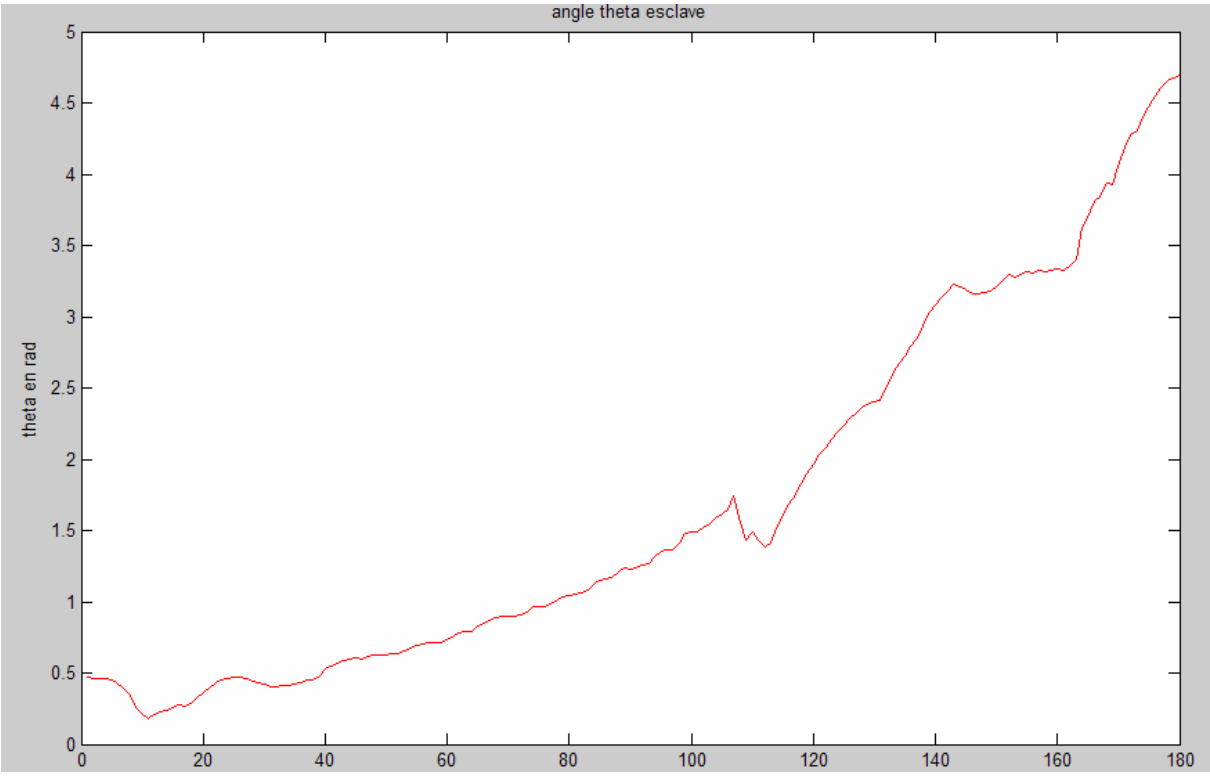
Lorsqu'on observe la variation de l'angle θ tout au long de cette expérience, on se rend compte que lorsque l'esclave est proche du maître, il varie brusquement. Ceci est dû au programme de traitement d'image.

7 – 3 – 2 – 2 Expérience 2

- **Trajectoire**

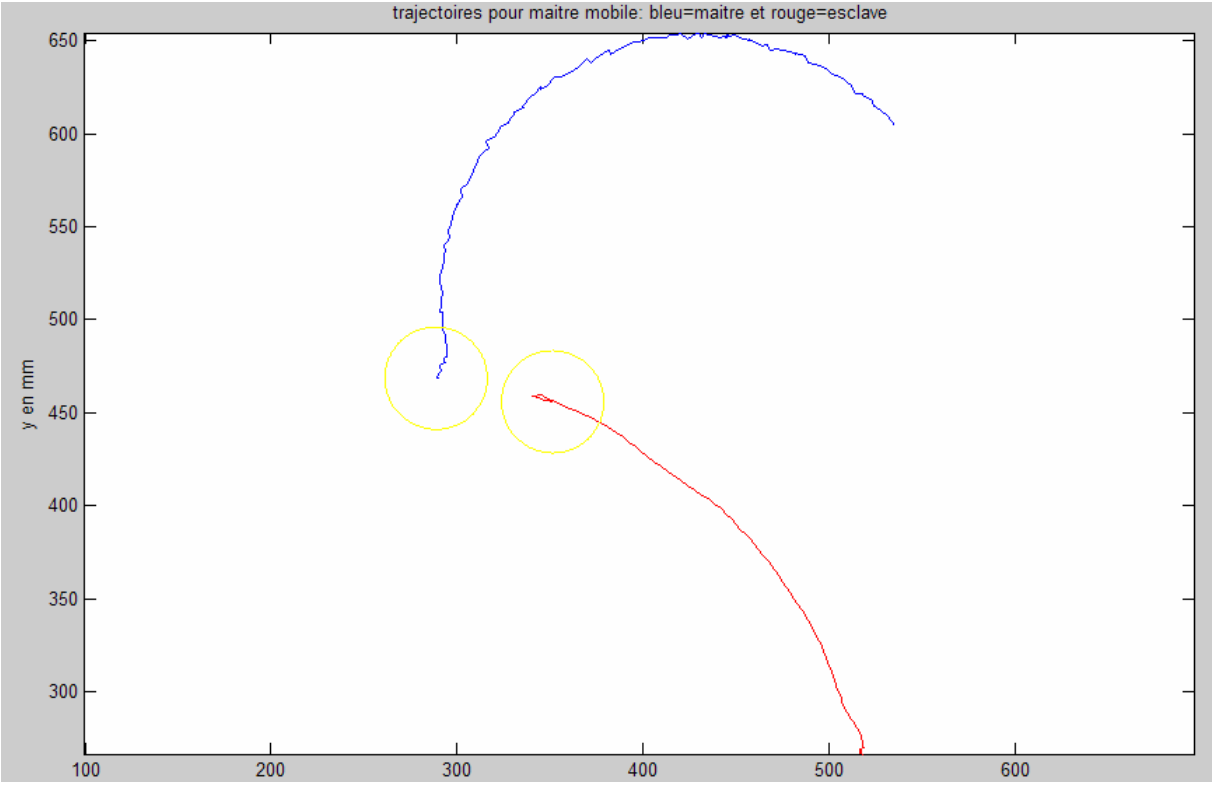


- **Angle θ**

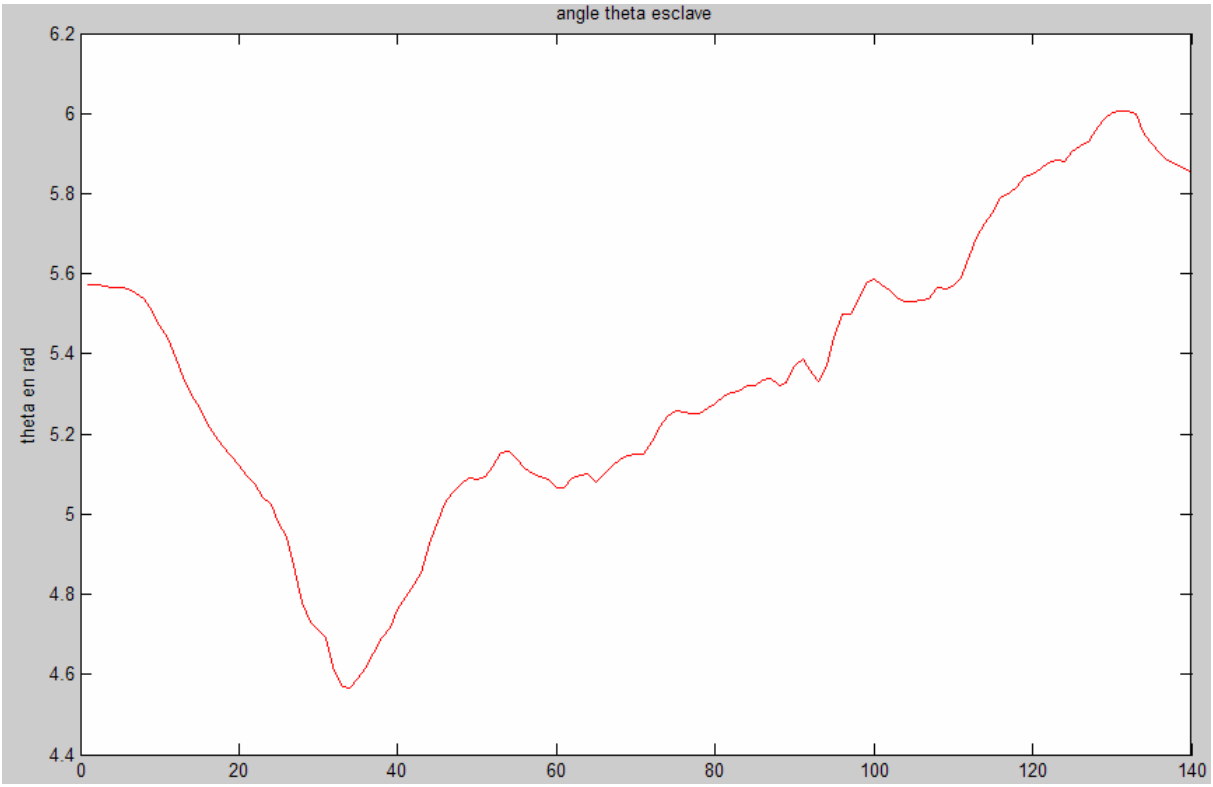


7 – 3 – 2 – 3 Expérience 3

- **Trajectoire**



- **Angle θ**



Pour les deux expériences précédentes, l'esclave percute le maître lorsqu'il le rattrape. La représentation dans Matlab des trajectoires et de la position finale des deux robots montre le contraire. Nous pensons que ceci est à la camera qui filme les deux robots sous un angle.

8 – Conclusion

Malgré plusieurs difficultés rencontrées lors de la phase d'implantation du filtre de Kalman et de la commande prédictive non classique, on peut dire que le cahier de charge a été quasiment respecté. Il reste encore des quelques améliorations à apportées, notamment sur l'amélioration du temps de traitement des différents algorithmes qui est encore relativement élevé et le programme de traitement d'image. Nous avons aussi bénéficié d'une solide base théorique réalisée par Niederberger et Herrera.

ANNEXE A

A1 : Programme Matlab de calcul du Jacobien

```
syms x y theta h L vd vg real
f = [x + h/2*cos(theta)*vd + h/2*cos(theta)*vg
     y + h/2*sin(theta)*vd + h/2*sin(theta)*vg
     theta + h/L*vd - h/L*vg];
X = [x y theta]';
u = [vd vg]';
dfdx_B = jacobian(f,X)
dfdu_B = jacobian(f,u)
```

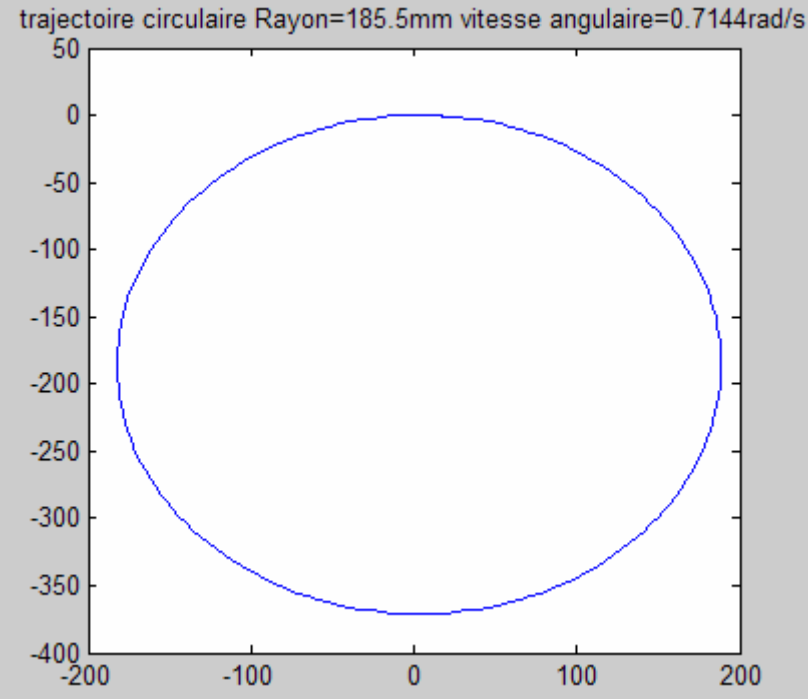
A2 : Programme Matlab pour le modèle

```
function [t,X,y] = model(tend,h,Vr,Vl)  % model based on straight segments
 A=eye(3,3);
 H=[1 0 0; 0 1 0];
 L=53; %distance between both wheels
%
 t = 0:h:tend;
 u=[Vr; Vl];
 X(:,1)=zeros(3,1);
 for k=1:length(t)-1
 theta(k)=X(3,k);
 B = [h/2*cos(theta(k)), h/2*cos(theta(k))
     h/2*sin(theta(k)), h/2*sin(theta(k))
     h/L, -h/L];

 X(:,k+1)=A*X(:,k)+B*u(:,k);
 y(:,k+1)=H*X(:,k+1);
 end
```

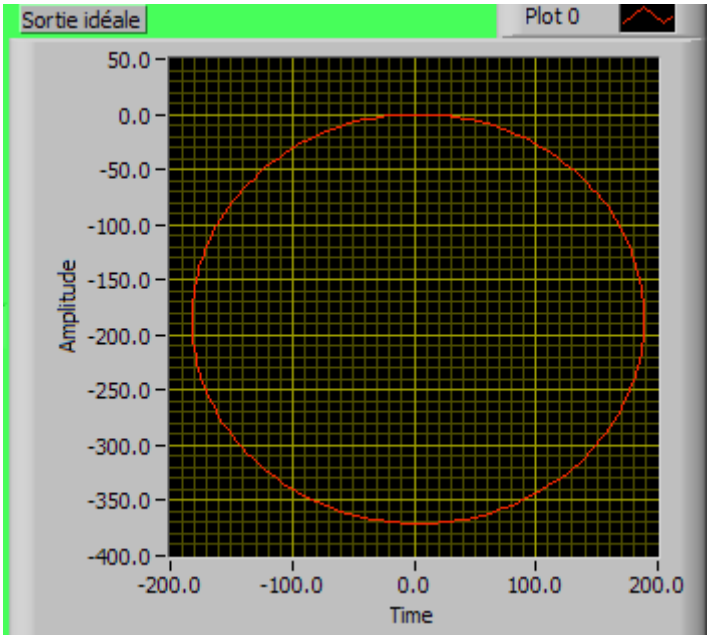
A3 : Programme de simulation du modèle

```
clear all;
% génération de la commande à priori
enter_pour_traj_circulaire
%formation des matrices constantes
 A=eye(3,3);
 H=[1 0 0; 0 1 0];
 % calcul des états
 X1=[0;0;0];
tend=t(end);
 [t,X,y] = model(tend,h,vd,vg);
 figure(3)
 abcisse_x = X(1,:); ordonnee_y = X(2,:);
 plot(abcisse_x,ordonnee_y)
```

A4 : résultat de la simulation



A5 : résultat simulation dans Labview



ANNEXE B

B1 : Programme qui simule le modèle bruité

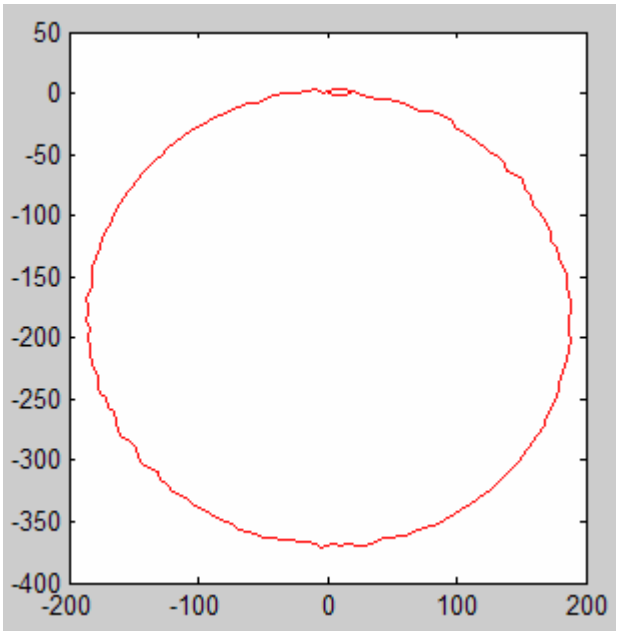
```
function [t,X,y] = plant(tend,h,Vr,Vl,Q,R)   %equivalent to 'model' adding noise in the process state and measurement
 A=eye(3,3);
 H=[1 0 0; 0 1 0;0 0 1];

 L=53; %distance between both wheels

 t = 0:h:tend;
 w = sqrt(diag(Q))*randn(size(t));  %size(t)=[1,length(t)] if t is a row vector
 v = sqrt(diag(R))*randn(size(t));
 u=[Vr; Vl];
 X(:,1)=zeros(3,1);

 for k=1:length(t)-1
 theta(k)=X(3,k);

 B = [h/2*cos(theta(k)), h/2*cos(theta(k))
     h/2*sin(theta(k)), h/2*sin(theta(k))
     h/L, -h/L];

 X(:,k+1)=A*X(:,k)+B*u(:,k)+w(:,k);
 y(:,k+1)=H*X(:,k+1)+v(:,k);
 end
```

B2 : Simulation du filtre de Kalman Etendu

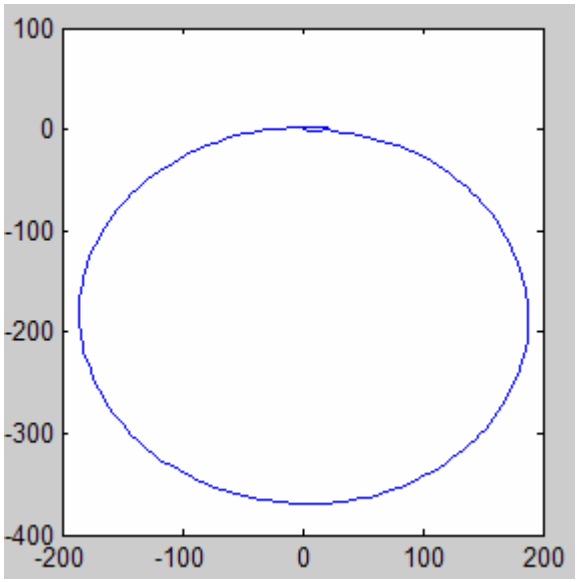
```
close all;
 Q=eye(3)*0.00001
 R=diag([1 1 0.1]);

 % création de la commande à priori
 enter_pour_traj_circulaire
 tend=t(end);

 % model_B sans le bruit (idéal)
[tideal,Xideal,yideal] = model(tend,h,vd,vg);

% model_B avec le bruit (mesure)
[tmeas,Xmeas,ymeas] = plant(tend,h,vd,vg,Q,R);

% paramètres constants
A=eye(3,3);
 H=[1 0 0; 0 1 0;0 0 1];
 L=53; %distance between both wheels

Pm=eye(3);
P=eye(3);

 u=[vd; vg];
 X(:,1)=zeros(3,1);


 for k=1:length(tmeas)-1
 theta(k)=X(3,k);

 A1 = [1,0, -1/2*h*sin(theta(k))*vd(k)-1/2*h*sin(theta(k))*vg(k)
       0,1,  1/2*h*cos(theta(k))*vd(k)+1/2*h*cos(theta(k))*vg(k)
       0,0,1]; % A1=jacobian(f,X) where f=(non linear model)

 W = [h/2*cos(theta(k)), h/2*cos(theta(k))
     h/2*sin(theta(k)), h/2*sin(theta(k))
     h/L, -h/L];
```

```
%Discrete Kalman Filter TIME UPDATE equations:

Xm(:,k+1)=A*X(:,k)+W*u(:,k); %a priori state estimate(k)
% Pm=A1*P*A1'+W*R*W'; %a priori error covariance estimate(k)
Pm=A1*P*A1'+Q; %a priori error covariance estimate(k)

%Discrete Kalman Filter MEASUREMENT UPDATE equations:

K=Pm*H'*inv(H*Pm*H'+R); %Kalman gain(k)
% K=Pm*H'*inv(H*Pm*H'); %Kalman gain(k)
X(:,k+1)=Xm(:,k+1)+K*(ymeas(:,k+1)-H*Xm(:,k+1)); %a posteriori state estimate(k) to calculate the following a priori state estimates
P=(eye(3)-K*H)*Pm; %a posteriori error covatiance estimate(k)

 end

figure(1)
subplot(3,1,1), plot(tmeas,ymeas(1,:),'b',tmeas,Xideal(1,:),'g',tmeas,X(1,:),'r')
subplot(3,1,2), plot(tmeas,ymeas(2,:),'b',tmeas,Xideal(2,:),'g',tmeas,X(2,:),'r')
subplot(3,1,3), plot(tmeas,ymeas(3,:),'b',tmeas,Xideal(3,:),'g',tmeas,X(3,:),'r')

figure(2)
 abcisse_x = X(1,:); ordonnee_y = X(2,:);
  plot(abcisse_x,ordonnee_y)
 hold on
  abcisse_xmeas = ymeas(1,:); ordonnee_ymeas = ymeas(2,:);
  plot(abcisse_xmeas,ordonnee_ymeas,'r')
```

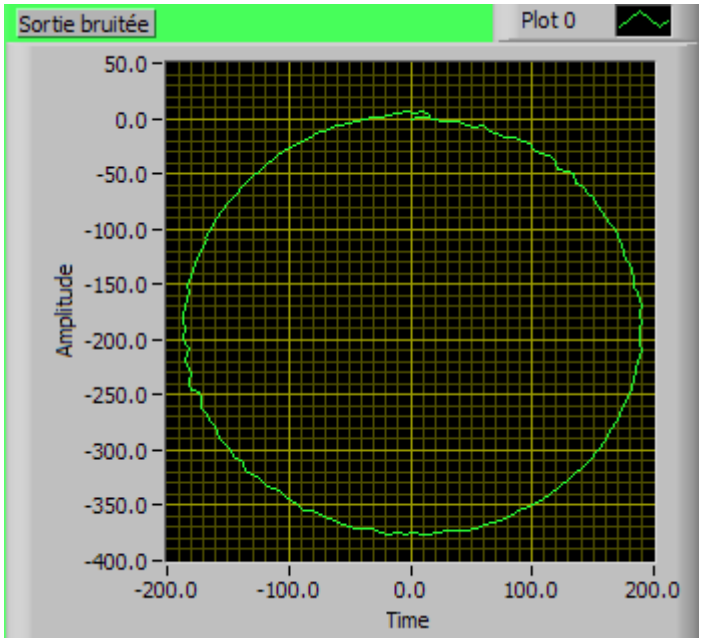
B3 : Résultat de la simulation du modèle bruité



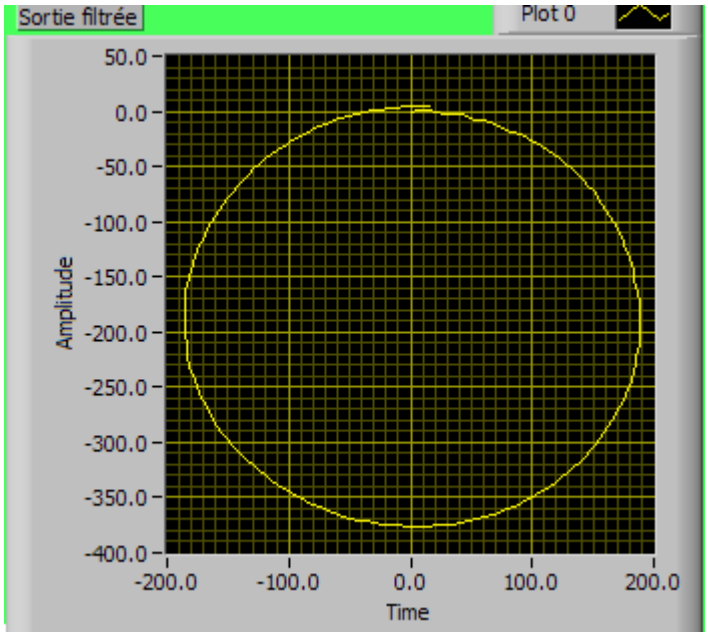
B4 : Résultat de la simulation du filtre de Kalman Etendu



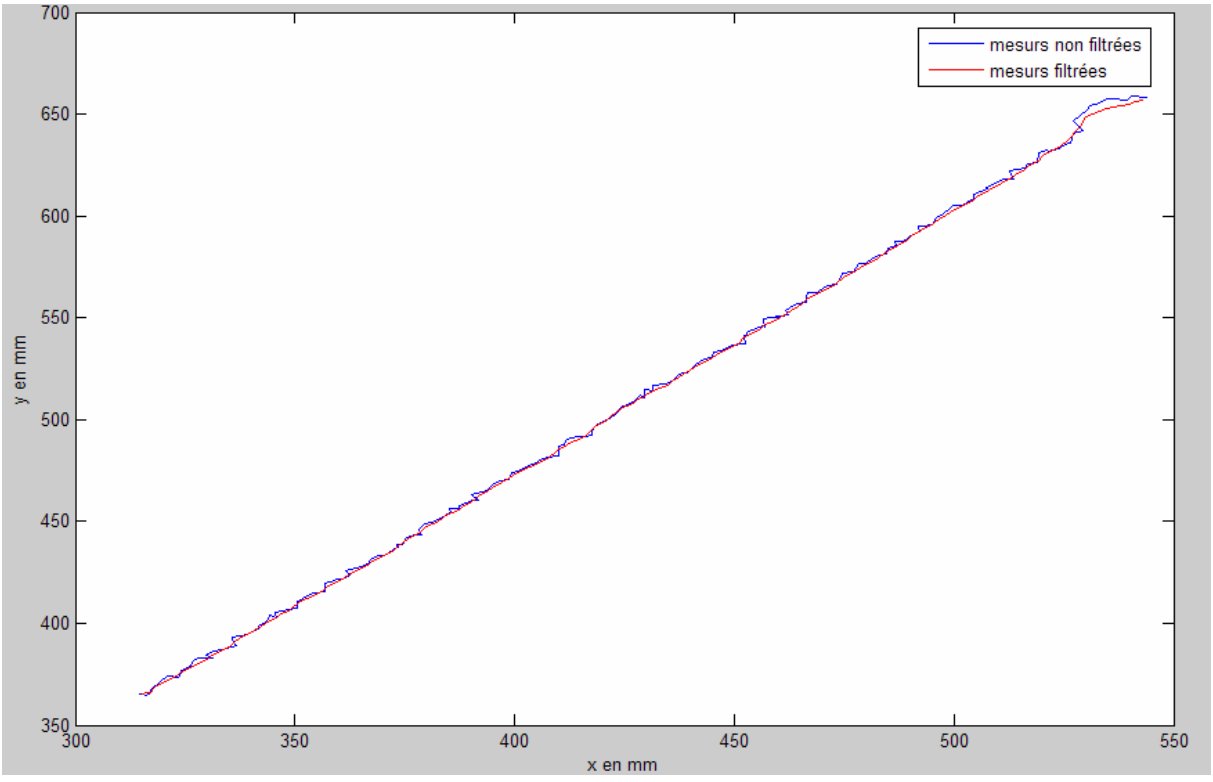
B5 : Résultat sous Labview : modèle bruité



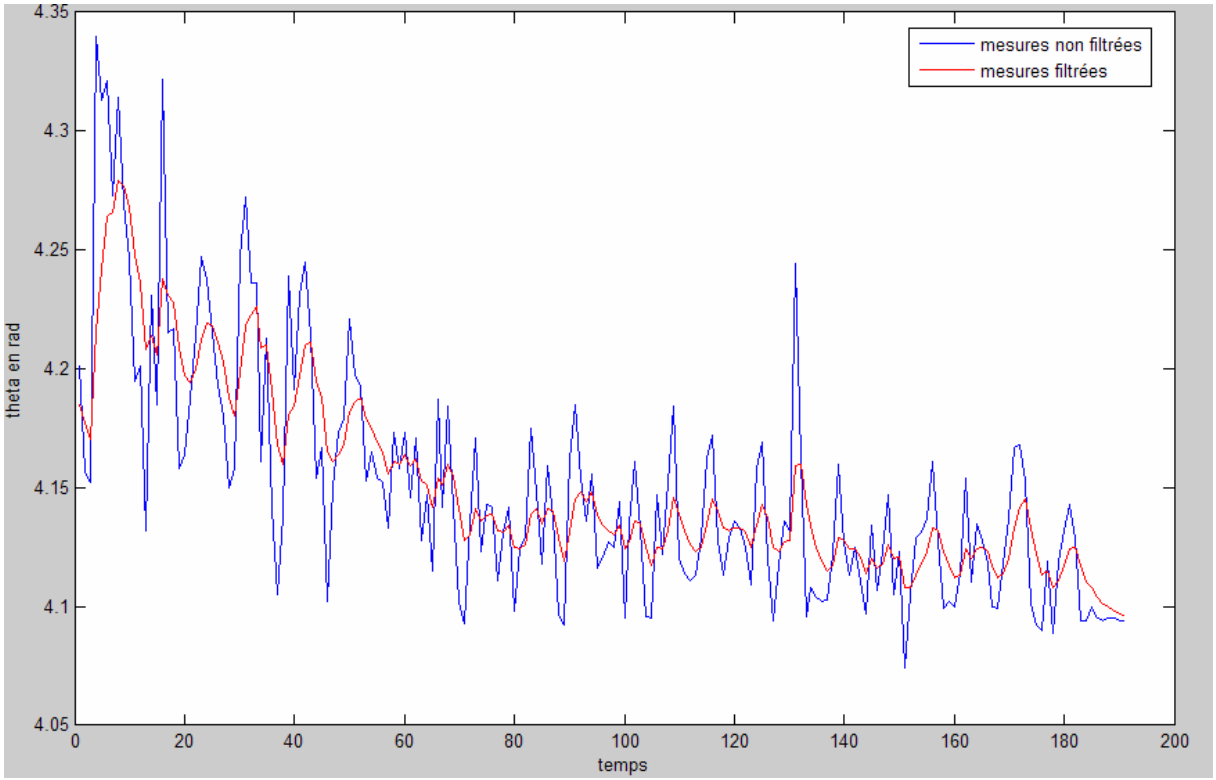
B6 : Résultat sous Labview : sortie filtrée



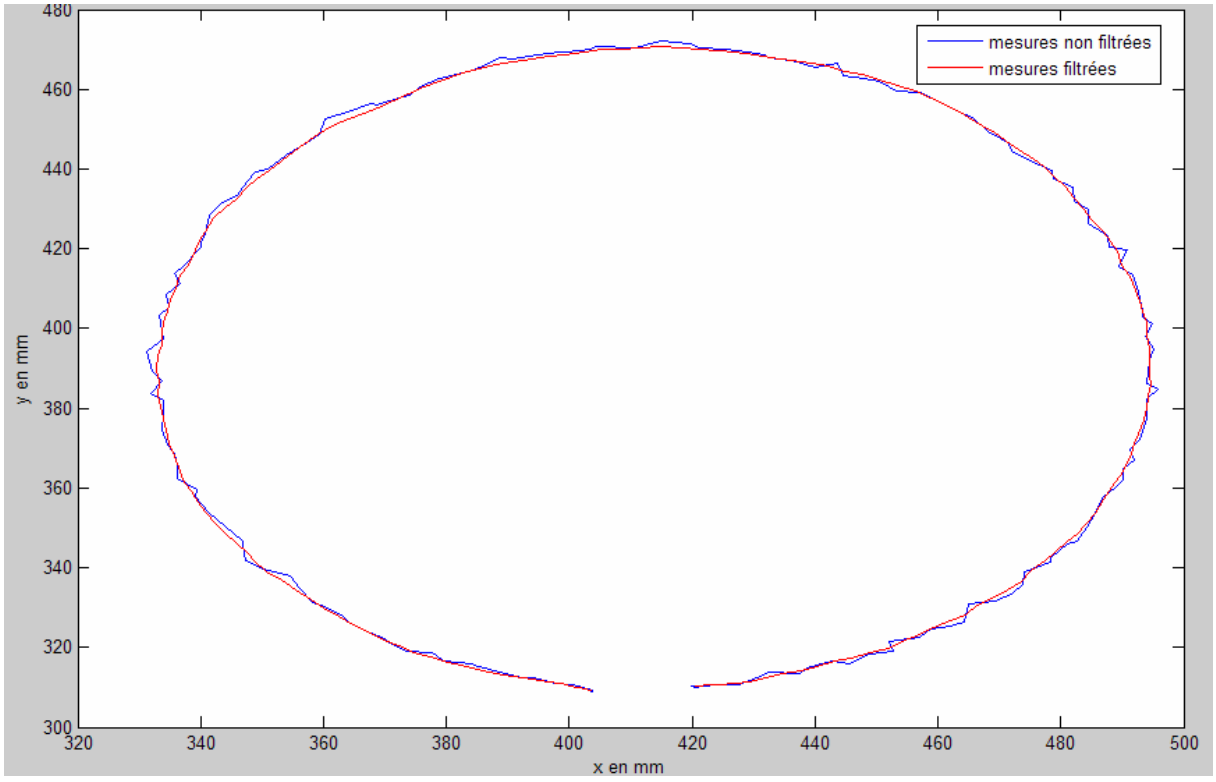
B7 : Simulation en temps réel : trajectoire rectiligne



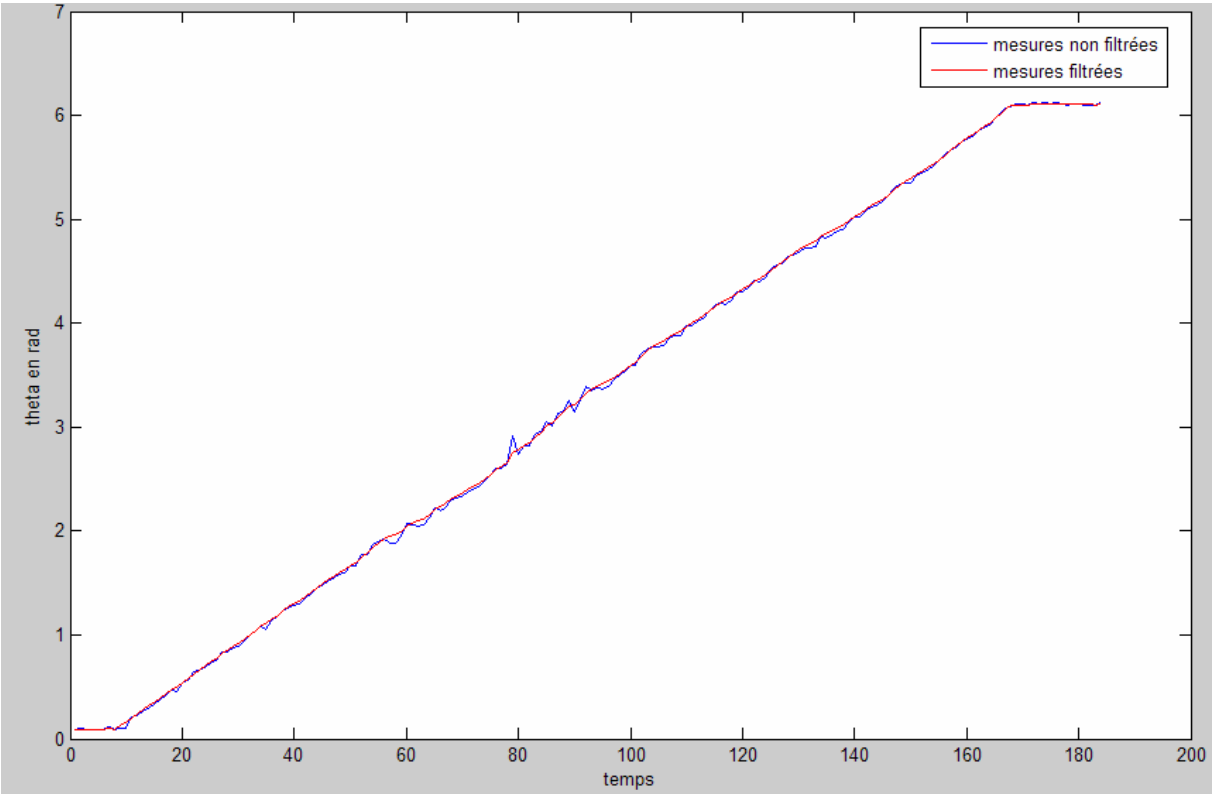
B8 : Simulation en temps réel : angle θ pour trajectoire rectiligne



B9 : Simulation en temps réel : trajectoire circulaire



B10 : Simulation en temps réel : angle θ pour trajectoire circulaire



BIBLIOGRAPHIE:

[1] Trajectory tracking trough predictive control: application to the Khepera robot, Juan Herrera, Diploma thesis, 1999

[2] Predictive control Design for a Khepera robot: Principles, Simulations and Real-time Implementation, André S.P Niederberger, Diploma thesis, February 2002

[3] Position estimation and control of an autonomous vehicle, Joana Maria Comas Marti, Diploma thesis, June 2005

[4] An introduction to the Kalman filter, Greg Welch and Gary Bishop, University of North California at Chapel Hill, April 2004