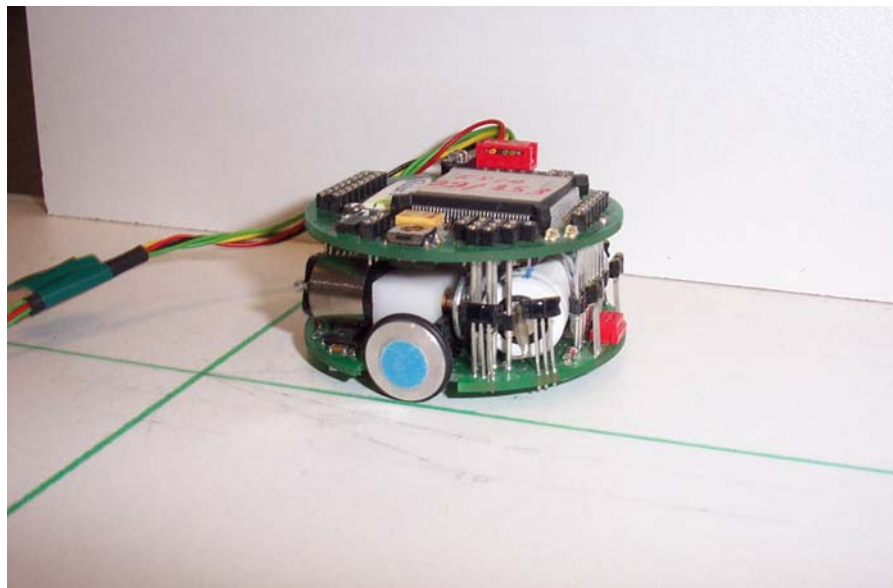


*Rédigé par :* Laurent BALAT, Fabrice BERNA, Brice CARBOU,  
Alexandre FILIPIAK, Jérôme FUCHET  
*Dans le cadre du :* module Automatique  
*Encadré par :* Jean-Yves TIGLI

## Suivi de mur du robot Khepera



# Sommaire

- SOMMAIRE .....2**
- INTRODUCTION .....3**
- PRESENTATION DU ROBOT .....4**
  - PRESENTATION .....4
  - Caractéristiques :* .....4
  - LES COMMANDES DU ROBOT .....4
- CALIBRAGE .....5**
  - RAPPEL DU MODELE UTILISE .....5
  - METHODE DE CALIBRAGE .....5
  - RESULTATS OBTENUS .....6
    - Courbe du flux capteur 1* .....6
    - Paramètres calculés* .....6
- SUIVI DE MUR DANS LE CAS D'UN SYSTEME CONTINU .....7**
  - LOI DE COMMANDE .....7
  - IMPLEMENTATION .....9
  - ASTUCE .....9
- SUIVI DE MUR DANS LE CAS D'UN SYSTEME DISCRET .....10**
  - LOI DE COMMANDE .....10
  - IMPLEMENTATION .....11
- SUIVI DE MUR DANS LE CAS D'UN SYSTEME DISCRET AVEC OBSERVATEUR.....12**
  - LOI DE COMMANDE .....12
  - IMPLEMENTATION .....13
- CONCLUSION .....14**

## Introduction

L'objectif de ce projet est de faire suivre un mur selon plusieurs méthodes au robot Khepera. Le robot Khepera est un petit robot motorisé (deux roues) qui dispose de huit capteurs infrarouges (capteurs de proximité) ainsi que d'un odomètre.

Le travail à effectuer pour faire suivre le mur à notre robot Khepera se fait en quatre parties :

- Effectuer une campagne de mesures sur les différents capteurs de proximité. Ces mesures ont pour but de calibrer le robot.
- Faire suivre le mur par le Khepera en utilisant une loi de commande dans le cas d'un système continu.
- Faire suivre le mur par le Khepera en utilisant une loi de commande dans le cas d'un système discret.
- Faire suivre le mur par le Khepera en utilisant une loi de commande dans le cas d'un système discret avec observateur.



**Figure 1 : Le robot Khepera en pleine action**

## Présentation du robot

### Présentation

A la base le robot Khepera est un robot miniature (5,5 cm de diamètre) disposant de 8 capteurs infra rouges permettant la détection de la présence de sources lumineuses ou d'obstacles.

#### Caractéristiques :

- Actionneurs : 2 mini moteurs, système différentiel
- Capteurs : 8 capteurs de proximité / capteur de luminosité
- Capteurs : 2 capteurs de déplacement des roues
- Autres : 2 minis LED témoin.
- Microprocesseur : Motorola 68000 16 Mhz
- Dimensions : cylindre d'environ 30 mm de haut et 55 mm de diamètre
- Poids : 80 g
- Vitesse max : 60 cm/s
- Vitesse min : 2 cm/s
- Autonomie : 45 minutes par batteries Ni-Cad



Figure 2 : Gros plan sur le Khepera

### Les commandes du robot

Nous utilisons les commandes suivantes dans notre implémentation :

- **G(adr, pos1, pos2)** : initialise les compteurs de l'odomètre à pos1 et pos2.
- **D(adr, vit1, vit2)** : permet de changer la vitesse des roues du robot, on les initialise à 0, puis on donne une vitesse constante pour effectuer le calibrage.
- **H(adr, pos1, pos2)** : permet de relever la valeur des compteurs de l'odomètre. A chaque incrémentation, 0.08 mm est parcouru par la roue concernée.
- **N(adr, flux1, flux2, flux3, flux4, flux5, flux6, flux7, flux8)** : relève la valeur du flux reçue sur chaque proximateur numéroté de 1 à 8. La valeur obtenue est comprise entre 0 et 1023.

# Calibrage

## Rappel du modèle utilisé

Le modèle physique et les données expérimentales nous ont permis de déduire un modèle raisonnable pour évaluer le flux lumineux détecté par un proximètre.

Ce modèle est représenté par la formule (gaussienne) suivante :

$$\phi(d, \theta) = (A/d^2) * \exp(-(\theta - \theta^*)/2\sigma^2)$$

avec:

$\phi$  = valeur du flux lumineux donnée par le capteur

A = constante (Amplitude)

d = distance du Khepera au mur

$\theta$  = angle de rotation du Khepera

$\theta^*$  = angle de rotation pour lequel le capteur obtient le maximum de flux lumineux

$\sigma$  = écart type de la gaussienne

Notre première tâche est la phase de calibrage, qui consiste à identifier les paramètres A,  $\theta^*$  et  $\sigma$ .

## Méthode de calibrage

Nous faisons faire un tour complet au robot à vitesse constante. Lors de cette rotation, on enregistre l'ensemble des valeurs retournées par un capteur en fonction de l'angle du robot.

La formule suivante permet de calculer l'angle du robot à partir de la position des roues :

$$\theta(t) = \theta(0) + (I_2(t) - I_1(t)) / 2r$$

Avec :

$\theta(t)$  = angle de rotation du Khepera à l'instant t.

$I_1(t)$  = la distance parcourue par la roue gauche depuis l'instant 0 (mesure effectuée par odomètre).

$I_2(t)$  = la distance parcourue par la roue droite depuis l'instant 0 (mesure effectuée par odomètre).

On obtient ainsi une courbe de l'intensité du flux en fonction de l'angle qui comporte des bruits de mesure. Comme ces bruits sont de haute fréquence, nous pouvons utiliser un filtre passe bas pour les supprimer :

$$\phi(\theta_{(t)}) = 0.7 * \phi(\theta_{(t)}) + 0.2 * \phi(\theta_{(t-1)}) + 0.1 * \phi(\theta_{(t-2)})$$

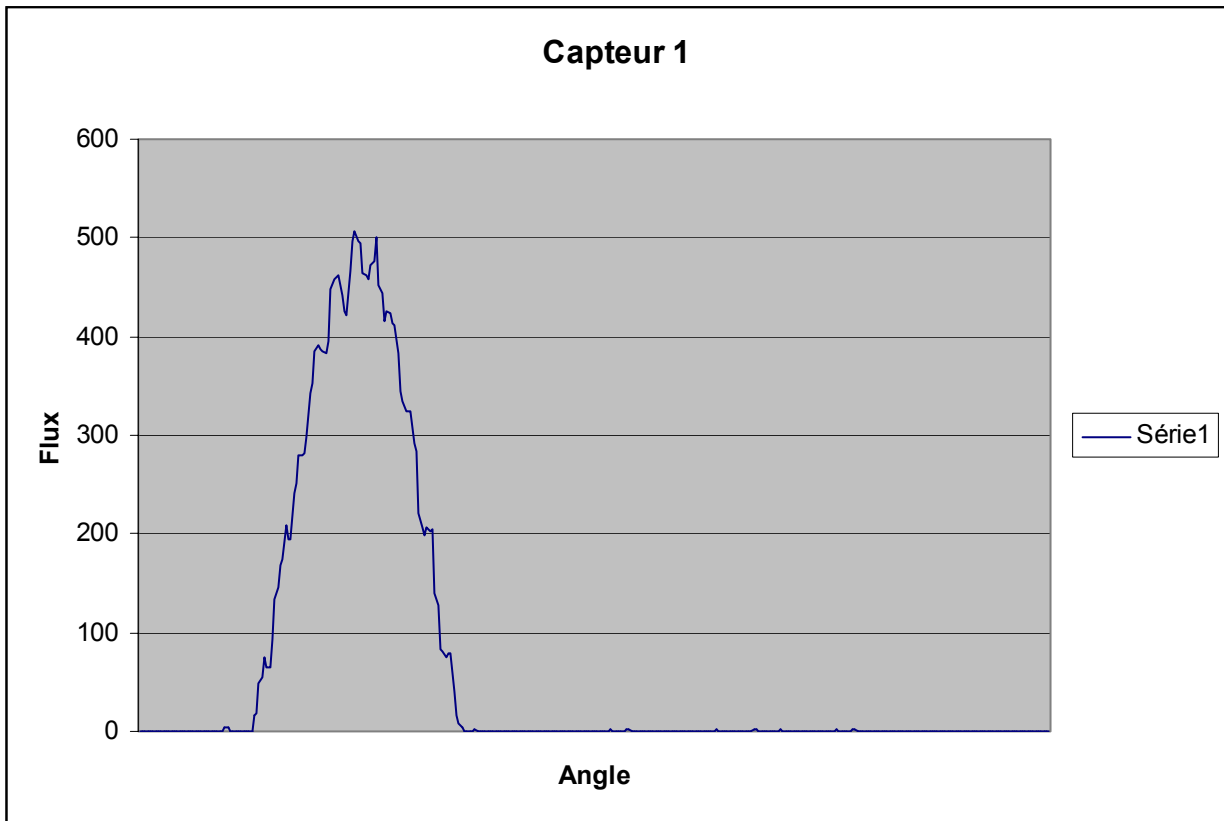
Avec :

$\phi(\theta_{(t)})$  = flux pour angle de rotation  $\theta$  à l'instant t.

Une fois que nous avons cette courbe, nous pouvons calculer les constantes :

- L'angle  $\theta^*$  correspond au sommet de la gaussienne.
- La valeur maximale du flux est  $(A/d^2)$ . Nous pouvons donc déduire A, sachant que nous connaissons d.
- L'écart type est la moitié de la largeur de la parabole à 60% de sa hauteur.

*Courbe du flux capteur 1*



Pour une distance de 2cm, avec des parasites lumineux (néons).

*Paramètres calculés*

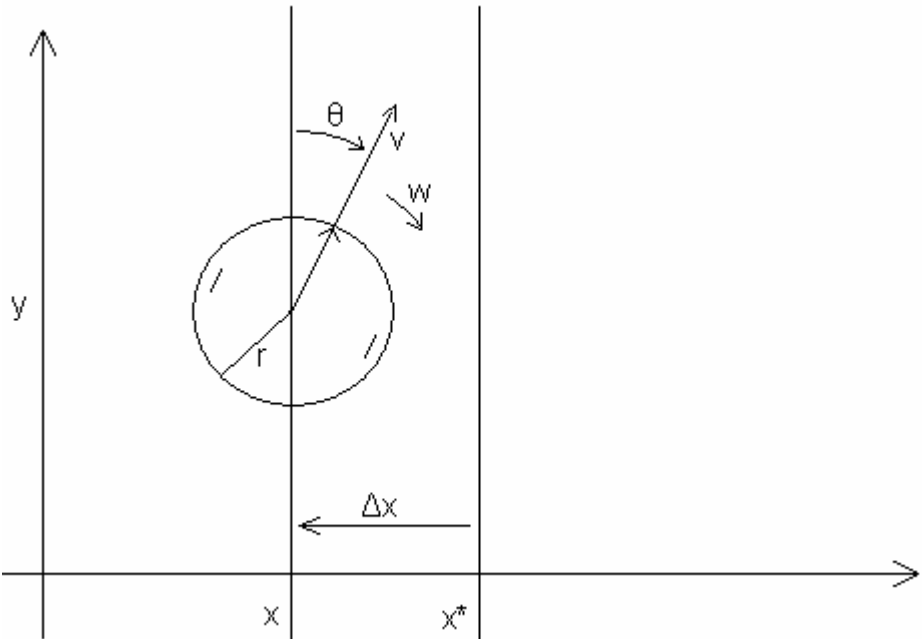
A partir de plusieurs essais de calibrage, nous avons obtenu les paramètres suivants, pour notre Khepera :

- A = 700
- d = 2.5 cm
- $\theta^* = 0.35$
- $\sigma = 0.3$

# Suivi de mur dans le cas d'un système continu

**Loi de commande**

---



$\theta$  = angle  
 $v$  = vitesse  
 $w$  = vitesse angulaire

Modèle d'état du Khépéra :

$$\begin{aligned} \dot{x} &= -v \times \sin(\theta) \\ \dot{y} &= v \times \cos(\theta) \\ \dot{\theta} &= w \end{aligned}$$

Sous forme matricielle :

$$\frac{d}{dt} \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = 0 \times \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\sin(\theta) & 0 \\ \cos(\theta) & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} v \\ w \end{pmatrix}$$

On linéarise au voisinage de 0 en utilisant la consigne suivantes:

$$\begin{aligned} x^*(t) &= d \\ v^*(t) &= v^* \text{ constante} \\ y^*(t) &= v^* t + y_0 \\ w^*(t) &= 0 \\ \theta^*(t) &= 0 \end{aligned}$$

et les coordonnées sous forme de  $\Delta = \text{grandeur} - \text{grandeur}^*$

On obtient :

---

$$\Delta \dot{x} = -v^* \times \Delta \theta$$

$$\Delta \dot{y} = \Delta v$$

$$\Delta \dot{\theta} = \Delta w$$

Et sous forme matricielle :

$$\frac{d}{dt} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{pmatrix} = \begin{pmatrix} 0 & 0 & -v^* \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} \Delta v \\ \Delta w \end{pmatrix}$$

On peut alors découpler l'équation en 2 :

$\Delta v$  contrôle y

$$\frac{d}{dt}(\Delta y) = 0 \times \Delta y + 1 \times \Delta v \quad (\text{Système 1})$$

Et  $\Delta w$  contrôle x et  $\theta$

$$\frac{d}{dt} \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} = \begin{pmatrix} 0 & -v^* \\ 0 & 0 \end{pmatrix} \times \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \times \Delta w \quad (\text{Système 2})$$

Commandabilité :

$\zeta_1 = (B1) = 1 \neq 0 \Rightarrow$  système 1 gouvernable

$\zeta_2 = (B2 \ A2 \times B2) = \begin{pmatrix} 0 & -v^* \\ 1 & 0 \end{pmatrix}$  de rang 2  $\Rightarrow$  système 2 gouvernable

On peut donc implémenter notre loi de commande : un régulateur par bouclage d'état linéaire :

$$U = -KX \text{ avec } K = \begin{pmatrix} k_1 \\ k_x \\ k_\theta \end{pmatrix}$$

On obtient donc:

$$\frac{d}{dt}(\Delta y) = 0 \times \Delta y - k \times \Delta y$$

Et :

$$\frac{d}{dt} \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} = \begin{pmatrix} 0 & -v^* \\ 0 & 0 \end{pmatrix} \times \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \end{pmatrix} \times (k_x \ k_\theta) \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} = \begin{pmatrix} 0 & -v^* \\ 0 & 0 \end{pmatrix} \times \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} (-k_x \Delta x - k_\theta \Delta \theta)$$

On fait en sorte que  $k_x$  et  $k_\theta$  donnent une valeur propre double  $k_2$

$$\triangleright k_\theta = 2k_2$$

$$\triangleright k_x = -\frac{k_2^2}{v^*}$$

Et on trouve les valeurs de  $k_1$  et  $k_2$  avec Matlab.

## Implémentation

---

Les valeurs de  $K_x$  et  $K_\theta$  s'obtiennent avec le script MathLab suivant :

```
v=30; % vitesse en mm/s
T=0.3; % Position des pôles
A=[0,-v;0,0];
B=[0;1];
C=[1,0;0,1];
D=[0;0];

syscont=ss(A,B,C,D); % Systeme continu
P=[-1/T,-1/T]; % Limites sur les pôles
K=acker(A,B,P) % Calcul des pôles

0.3704 -6.6667
```

## Astuce

---

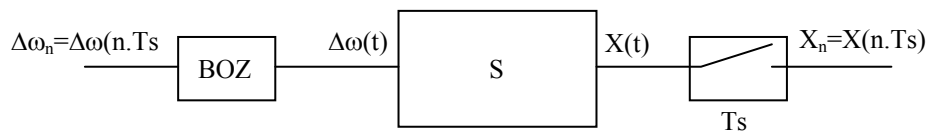
Pour limiter les mouvements brutaux, les calculs des distances sont effectués de telle sorte que le Khepera ne pense jamais être éloigné de plus de 10 mm de la trajectoire désirée. Ainsi il ne corrige pas sa trajectoire de manière brutale, et se comporte comme s'il n'était qu'à 10 mm de sa destination.

## Suivi de mur dans le cas d'un système discret

### Loi de commande

Dans le cas d'un système discret, la loi de commande reste la même que celle du cas continu. En revanche le système d'état est modifié.

Soit S le système continu que nous avons précédemment et  $T_s$  la période d'échantillonnage. Nous pouvons obtenir le système en temps continu en intercalant un échantillonneur sur les mesures effectuées par le système et un bloqueur d'ordre zéro (BOZ) sur les commandes. Nous obtenons alors le schéma suivant :



Cet ensemble forme le système discret  $S_d$  :  $X_{n+1} = A(T_s).X_n + B(T_s).\Delta\omega_n$

Il est possible de déduire les matrices  $A(T_s)$  et  $B(T_s)$  à partir des matrices  $A$  et  $B$  du système continu à l'aide des équations suivantes :

$$A(T_s) = e^{A.T_s}$$

$$B(T_s) = \int_0^{T_s} e^{A.T_s} . B . dt$$

Pour le système continu nous avons les matrices suivantes :  $A = \begin{bmatrix} 0 & -v^* \\ 0 & 0 \end{bmatrix}$  et  $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$$\text{D'où : } A(T_s) = \sum_{n=0}^{+\infty} \frac{1}{n!} (A.T_s)^n = I + A.T_s = \begin{bmatrix} 1 & -v^*.T_s \\ 0 & 1 \end{bmatrix}$$

$$B(T_s) = \int_0^{T_s} e^{A.T_s} . B . dt = \int_0^{T_s} (I + A.T_s) . B . dt = (I + \frac{1}{2} A.T_s^2) . B . T_s = \begin{bmatrix} -v^* . \frac{T_s^2}{2} \\ T_s \end{bmatrix}$$

Vérifions que le système est toujours contrôlable :

$$\begin{bmatrix} B \\ A.B \end{bmatrix} = \begin{bmatrix} -v^* . \frac{T_s^2}{2} & -\frac{3}{2} . v^* . T_s^2 \\ T_s & T_s \end{bmatrix}$$

Le déterminant de cette matrice n'est pas nul et elle est de rang 2, le système est donc toujours contrôlable.

Occupons nous à présent du placement des pôles.

Soit  $T$  la constante de temps du système, c'est à dire que les valeurs de l'état  $X(t)$  décroissent du tiers en  $T$  secondes et de 95% en  $3T$  secondes.

Nous choisissons d'utiliser des valeurs propres doubles avec une partie imaginaire nulle pour éviter les oscillations.

La partie réelle quand à elle est choisie égale à  $e^{-\frac{T_s}{T}}$ .

De même que précédemment la valeur de la matrice  $K$  utilisée pour la loi de commande est calculée à l'aide de Matlab .

## Implémentation

---

Les valeurs de  $K_x$  et  $K_\theta$  s'obtiennent avec le script MathLab suivant :

```
v=24;           % vitesse en mm/s
Ts=0.1;        % Position des poles
T=1;           % Vitesse de convergence
A=[0,v;0,0];
B=[0;1];
C=[1,0;0,1];
D=[0;0];

syscont=ss(A,B,C,D); % Système continu
sysdisc=c2d(syscont,Ts); % passage du continu au discret avec paramètre Ts

Ad=sysdisc.a;
Bd=sysdisc.b;

P=[exp(-Ts/T);exp(-Ts/T)]; % Pôles
K=acker(Ad,Bd,P) % Calcul de Kx et Ktheta
```

## Suivi de mur dans le cas d'un système discret avec observateur

### Loi de commande

---

Pour pouvoir stabiliser le Khepera en utilisant uniquement le capteur de flux, nous allons stabiliser le système 2:

$$\frac{d}{dt} \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} = \begin{pmatrix} 0 & -v^* \\ 0 & 0 \end{pmatrix} \times \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \times \Delta w$$

En utilisant un observateur.

Lorsque l'on utilisait le capteur infrarouge et l'odomètre, nous avions accès à tout l'état, la sortie y était donnée par :

$$\begin{matrix} Y & C & X \\ 678 & 678 & 678 \\ \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} & = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times & \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} \end{matrix}$$

Maintenant que nous n'utilisons que le capteur infrarouge, la sortie y est le flux  $\phi$ .

On va donc découpler le  $\phi$  afin de retrouver la matrice C

Pour cela, on utilise les dérivées partielles du flux :

$$\phi = \frac{A}{(x^* - r + \Delta x)^2} \exp\left(\frac{-(\Delta \theta - \theta^*)}{2\sigma^2}\right)$$

Grâce à des approximations, on obtient :

$$\left\{ \begin{array}{l} \frac{\partial \phi}{\partial \Delta x}(0,0) = \frac{-2}{x^* - r} \phi(0,0) \\ \frac{\partial \phi}{\partial \Delta \theta}(0,0) = \frac{\theta^*}{\sigma} \phi(0,0) \end{array} \right.$$

Le flux est donc maintenant donné par l'équation :

$$\begin{matrix} Y & C & X \\ \Delta \phi & = \begin{pmatrix} \frac{-2}{x^* - r} & \frac{\theta^*}{\sigma} \end{pmatrix} \times & \begin{pmatrix} \Delta x \\ \Delta \theta \end{pmatrix} \end{matrix}$$

---

L'observateur consiste à obtenir une estimation de l'état complet à partir de sorties ne représentant qu'une partie de l'état.

Cette estimation de l'état nous permettra de calculer la commande à appliquer. Voici le système qui le résume :

$$\begin{cases} \dot{\hat{x}} = A\hat{x} + Bu - L(C\hat{x} - y) \\ u = -K\hat{x} \end{cases}$$

Notons que la matrice L sert à corriger l'erreur d'approximation de l'état. Si on note :  $e = \hat{x} - x$ , on obtient :

$$\begin{cases} \dot{\hat{x}} = (A - BK)x - BKe \\ \dot{e} = (A - LC)e \end{cases}$$

Pour que l'erreur  $e$  tende vers 0, il faut qu'on puisse placer les valeurs propres de la matrice A-LC à l'aide de la matrice L. Ceci est possible si le système est observable. Pour le vérifier, on calcule le rang de la matrice d'observabilité suivante :

$$O = \begin{pmatrix} C \\ CA \\ \dots \\ CA^{n-1} \end{pmatrix}$$

Dans le cas de notre système en temps discret, on obtient :

$$O = \begin{pmatrix} C_x & C_\theta \\ C_x & -vT_s C_x + C_\theta \end{pmatrix}$$

O est de rang 2, donc le système est observable.

On calcule ensuite les valeurs de la matrice L grâce à Matlab. Les valeurs de la matrice K ne changent pas.

## Implémentation

---

Les valeurs de  $L_x$  et  $L_\theta$  s'obtiennent avec le script MathLab suivant :

```
v=24; % vitesse en mm/s
Ts=0.5; % Position des pôles
T=0.3; % Vitesse de convergence
A=[0,v;0,0];
B=[0;1];
xEtoile=40; % distance du centre du Khepera au mur en mm
alphaEtoile=0; % angle du capteur par rapport au mur
ecartType=0.35; % écart type de l'équation du flux
C=[-2/xEtoile,alphaEtoile/ecartType^2];
D=[0;0];

syscont=ss(A,B,C,D); % Système continu
sysdisc=c2d(syscont,Ts); % passage du continu au discret avec paramètre Ts

Ad=sysdisc.a;

P=[exp(-Ts/T);exp(-Ts/T)]; % Pôles
Ltrans=acker(Ad',C',P);
L=Ltrans';
```

## Conclusion

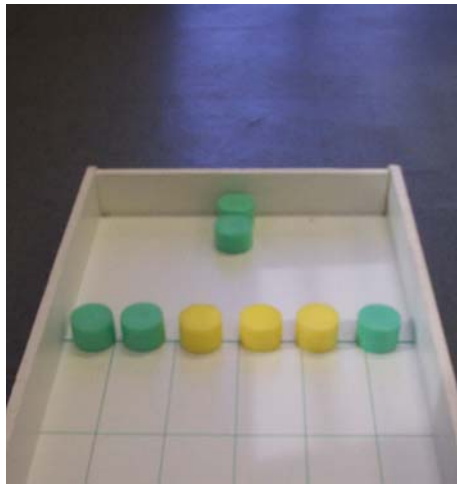
Malgré les difficultés à modéliser et paramétrer les différents modèles, nous avons réussi à implémenter un programme qui suit correctement un mur.

Les programmes « continu » et « discret » répondent exactement à nos attentes et suivent le mur, imperturbables. Ils permettent aussi au robot d'éviter les petits obstacles ou de repérer les déformations du mur. Le programme avec observateur reste encore à améliorer : il permet au robot de suivre le mur mais a tendance à entrer en oscillation dès qu'il rencontre des obstacles ou que le mur disparaît.

L'analyse des trois modèles ainsi qu'une optimisation de nos algorithmes nous ont permis de mettre au point un petit programme qui permet au robot Khepera de :

- Suivre les murs
- Suivre les angles (intérieurs comme extérieur)
- Contourner les gros obstacles

Ce programme n'était pas demandé, c'est pourquoi nous ne l'avons pas détaillé dans notre rapport.



**Figure 3 : Parcours du combattant**

*« Notre vie ne sera plus jamais la même maintenant que nous l'avons vue avec les yeux d'un Khepera ».*

## Codes sources

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <signal.h>
#include "../include/serial_io.h"
#include "../include/I.v.0.0.h"

#define RAYONROBOT 26.0
#define TAILLETABSAVE 2000
#define APPROXECARTTYPE 20
#define DUREECOURSE 10000
#define PI 3.1415926
#define GAIN 2
#define VITESSECALIB 3
#define DISTCALIB 25

// parametres du modele continu
/* premier essai
pour V=30 et T=0.3
#define KXCONT 0.3704
#define KTHETACONT -6.6667
*/
/* deuxième essai
pour V=30 et T=0.5
#define KXCONT 0.1333
#define KTHETACONT -4
*/
/* troisieme essai
pour V=30 et T=0.4
#define KXCONT 0.2083
#define KTHETACONT -5
*/
#define KXCONT 0.1333
#define KTHETACONT -4
#define K 3

// parametres du modele discret
#define TS 0.5
#define T 0.3
#define V 2.4;
#define KXDISC 0.3704
#define KTHETADISC -6.6667

// Parametres du modele avec observateur
#define KOBS 0.001
#define TSOBS 0.1

// Parametres de calibrage
#define ECARTTYPE 0.35
#define FLUXMAX 700
#define ANGLEMAX 0.35

serial_t serial_1;
u_int16_t ValeursCapteurs[8];
int nbmesure;
int numcapt;
int etat;

struct DonneesCalibrage
{
    float AmplitudeMax;
    float AngleMax;
    float EcartType;
    float A;
};
```

```

/*****
* Fonction appelee lors du CTRL-C
*****/
void stopHandler(int x){
    printf("\nAAAARGHHH Je meurs !!!\n\n");
    D(&serial_1, 0, 0);
    L( &serial_1 , 0 , 0 );
    L( &serial_1 , 1 , 0 );
    exit(0);
}

/*****
* Fonction permettant d'allumer ou d'eteindre les leds du khepera
* @param int : la valeur du flux reçue par le capteur
*****/
void sature(int n){
    if ( n <= 5 ) L( &serial_1 , 0 , 1 ); // led trop loin allumee
    else if(n == 1023) L( &serial_1 , 1 , 1 ); // led trop pres allumee
    else
        { // les deux leds clignotent
            L( &serial_1 , 0 , 1 );
            L( &serial_1 , 0 , 0 );
            L( &serial_1 , 1 , 1 );
            L( &serial_1 , 1 , 0 );
        }
}

/*****
* Initialisation des parametres du robot
*****/
void initialisation (void)
{
    serial_1.dbg_serial= 1;
    serial_1.trc_serial= 1;
    serial_init_env(&serial_1); /* utilisation du port serie 1 */

    G( &serial_1 , 0 , 0 ); /* roues en position 0 */
    D( &serial_1 , 0 , 0 ); /* vitesse des roues nulle */
    nbmesure=0;
}

/*****
* Calcule l'ecart type en radians
* @param float* : le tableau des angles mesures
* @param u_int16* : le tableau des flux mesures
* @param float : valeur du flux maximum
* @param float : valeur de l'angle pour le flux maximum
* @return float : l'ecart type
*****/
float calculEcartType ( float* angle , u_int16* flux , float amplitude , float anglemax )
{
    float amp_60 = amplitude*0.60;
    float angle_60=0;
    int i = 0;

    while(flux[i] < amp_60) i++; // Parcours jusqu'à 60% du flux max
    angle_60 = angle[i];
    while(flux[i] < amplitude)i++; // Pour etre sur de passer la bosse
    while(flux[i] > amp_60)i++; // Parcours jusqu'à 60% du flux max

    return (angle[i] - angle_60) / 2;
}

```

```

/*****
* Calcule la distance au mur en mm
* @param struct DonneesCalibrage : les parametres calcules
* @param float* : angle mesure (radians)
* @param float : flux mesure
* @return float : la distance en mm
*****/
float calculDistance(struct DonneesCalibrage *parametres, float angle, u_int16 flux)
{
    float r;
    float deltaAngle;

    // deltaAngle est l'ecart en radians par rapport a l'angle de flux max (modulo PI)
    deltaAngle = angle - parametres->AngleMax;
    if (deltaAngle > PI) deltaAngle = 2 * PI - deltaAngle;

    // Pour aider le khepera a retrouver le mur sans trop de difficultes
    if ( flux < 20 ) {
        flux = 20;
        if ( etat != 1 ) {
            printf("Ben il est ou le mur??? Je pars a sa recherche...\n");
            etat = 1;
        }
    }
    // Pour aider le khepera a eviter le mur
    else if ( flux >= 1010 ){
        if( etat != 2 ){
            printf("Aie je suis trop pres !!!\n");
            etat = 2;
        }
    }
    else
        if( etat != 0 ) {
            printf("Ouf ca va mieux...\n");
            etat = 0;
        }

    // Traitements normaux
    if ( flux >= 1000 ) r = 10;
    else{
        r = 10 * sqrt(
            parametres->A / parametres->AmplitudeMax
            + parametres->AmplitudeMax
            / flux
            * exp( - deltaAngle * deltaAngle
                / ( 2 * parametres->EcartType * parametres->EcartType)
            )
        );
    }
    return r;
}

/*****
* Calcule le flux correspondant a une distance desiree
* ( pour une position du khepera parallele au mur )
* @param struct DonneesCalibrage : les parametres calcules
* @param float : distance desiree en mm
* @return float : le flux correspondant
*****/
float calculFlux( struct DonneesCalibrage *parametres, float distanceDesiree )
{
    float r;
    r = ( parametres->A / ( distanceDesiree * distanceDesiree / 100 ) )
        * exp (
            - ( parametres->AngleMax * parametres->AngleMax ) / ( 2 * parametres->EcartType * parametres-
>EcartType )
        );
    return r;
}

```

```

/*****
* Calcule l'angle
* @return float : l'angle
*****/
float calculAngle(void)
{
    float res;
    int coef;
    int32 p_right, p_left;

    // Donnees des roues
    H(&serial_1, &p_left, &p_right);

    // Calcul de l'angle
    res = ( ( p_right * 0.08 ) - ( p_left * 0.08 ) ) / ( 2 * RAYONROBOT );

    // Modulo 2 PI
    coef = (int) (res / (2 * PI));
    res = res - ( coef * 2 * PI);
    // Fin Modulo 2 PI

    return res;
}

/*****
* Rafraichit les valeurs recues par les 8 capteurs
*****/
void refreshValeurCapteurs(void){
    N(&serial_1,
      ValeursCapteurs,
      ValeursCapteurs+1,
      ValeursCapteurs+2,
      ValeursCapteurs+3,
      ValeursCapteurs+4,
      ValeursCapteurs+5,
      ValeursCapteurs+6,
      ValeursCapteurs+7);
    sature(ValeursCapteurs[numcapt]);
    return;
}

/*****
* Calibrage du robot
* Le robot effectue un tour complet sur lui meme.
* Il effectue une campagne de mesures et calcule ensuite l'amplitude max
* mesure, l'angle a cette amplitude, et l'ecart type.
* @param float : la distance de calibrage
* @return struct DonneesCalibrage : les parametres calcules
*****/
struct DonneesCalibrage calibrage ( float distance )
{
    int16 vright, vleft;
    int16 v_right, v_left;
    int32 p_right, p_left;
    int32 pright, pleft;
    int k = 0;

    struct DonneesCalibrage donnees;

    u_int16 tabflux[TAILLETABSAVE];
    float tabangle[TAILLETABSAVE];
    float angle = 0 ;
    float angleprec;
    int saturation = 0;

    vright = VITESSECALIB;
    vleft = -VITESSECALIB;
    pright = 0;
    pleft = 0;

    // Un tour complet
    D(&serial_1, vleft, vright);
    while( angle >= angleprec )
    {
        angleprec = angle;

```

```

        angle = calculAngle();
        refreshValeurCapteurs();

        /* filtre sur la mesure du flux */
        if (k>1) ValeursCapteurs[numcapt]=0.70*ValeursCapteurs[numcapt]+0.20*tabflux[k-
1]+0.10*tabflux[k-2];

        tabflux[k] = ValeursCapteurs[numcapt];
        tabangle[k] = angle;
        if (tabflux[k] > 1023)
        {
            saturation = 1;
            break;
        }

        if (donnees.AmplitudeMax < ValeursCapteurs[numcapt])
        {
            donnees.AmplitudeMax = ValeursCapteurs[numcapt];
            donnees.AngleMax = angle;
        }
        nbmesure++;
        k ++;
    }
    D(&serial_1, 0, 0);

    if(saturation) {
        printf("Erreur: saturation du capteur, le robot est trop pres de l'obstacle\n\t->calibration
interrompue\n");
        exit(1);
    }

    // Parametres du modele
    donnees.EcartType=calculEcartType(tabangle,tabflux,donnees.AmplitudeMax,donnees.AngleMax);
    donnees.A = donnees.AmplitudeMax * distance * distance;

    return donnees;
}

/*****
* Fait suivre le mur au robot (en continu ou discret)
* @param struct DonneesCalibrage : les parametres du modele
* @param float : la distance a suivre
* @param int : le mode de calcul ( 1-> continu , 2 -> discret )
*****/
void SuiviMur( struct DonneesCalibrage *parametres , float distDesiree , int mode )
{
    float deltaW;
    float deltaX;
    float deltaV;
    float deltaTheta;
    float angle;
    float initvitesse = 30;

    float vg;
    float vd;
    int16 vg_int;
    int16 vd_int;
    int dodo;

    if( mode == 1) dodo = 0;
    else dodo = TS * 100000;

    G(&serial_1,0,0); /* roues en position 0 */
    D(&serial_1, 1 , 1 );

    for ( ; ; )
    {
        usleep(dodo);

        angle = calculAngle();
        refreshValeurCapteurs();

        // Calculs
        // -> variation de distance par rapport a la distance desiree
        deltaX = calculDistance( parametres , angle , ValeursCapteurs[numcapt] ) - distDesiree;

        // -> Variation de l'angle par rapport a l'angle max

```

```

deltaTheta = angle;

// -> Parametre W
if( mode == 1) deltaW = KXCONT * deltaX + KTHETACONT * deltaTheta;
else deltaW = KXDISC * deltaX + KTHETADISC * deltaTheta;

// Application de la loi de commande aux roues
vg = initvitesse - RAYONROBOT * deltaW;
vg = vg * 0.8 / 10;
vg_int = floor(vg);
vd = initvitesse + RAYONROBOT * deltaW;
vd = vd * 0.8 / 10;
vd_int = floor(vd);

//printf("deltaX = %f deltaW = %f deltaV = %f vd = %d vg = %d
",deltaX,deltaW,deltaV,vd_int,vg_int);
//printf("Angle = %f Flux = %d Dist = %f \n", angle , ValeursCapteurs[numcapt] ,
(distDesiree+deltaX));

    D(&serial_1, vg_int, vd_int);
}
D(&serial_1,0,0);
}

/*****
* Fait suivre le mur au robot (en discret avec observateur sans odometre)
* @param struct DonneesCalibrage : les parametres du modele
* @param float : la distance a suivre
*****/
void SuiviMurObservateur( struct DonneesCalibrage *parametres , float distDesiree )
{
    float deltaW;
    float deltaX;
    float deltaTheta;
    float deltaFlux;
    float fluxEtoile;
    float initvitesse = 30;
    float delataFluxChapeau;
    float deltaFluxChapeau;

    float vg;
    float vd;
    int16 vg_int;
    int16 vd_int;
    float distMesuree;

    G(&serial_1 , 0 , 0 ); /* roues en position 0 */
    D(&serial_1, 0 , 0 ); /* vitesse des roues nulle */
    D(&serial_1, 1 , 1 );

    // Calibrage du modele avec observateur
    refreshValeurCapteurs();
    fluxEtoile = calculFlux( parametres , distDesiree ); // Calcul du flux pour le khepera dans les
conditions desirées (bonne distance et bonne direction)
    deltaX = 0;
    deltaTheta = 0;

    for ( ; ; )
    {
        //usleep(TSOBS * 100000);

        refreshValeurCapteurs();

        // Calculs
        // -> Variation du flux
        deltaFlux = ValeursCapteurs[numcapt] - fluxEtoile;

        // -> variation de la distance
        distMesuree = calculDistance( parametres , deltaTheta , ValeursCapteurs[numcapt] );
        deltaX = distMesuree - distDesiree;

        // -> Variation du flux estime
        deltaFluxChapeau = ( parametres->A / ( distMesuree * distMesuree / 100 ) )
* exp(-
( deltaTheta - parametres->AngleMax ) * ( deltaTheta - parametres->AngleMax )
/

```

```

        (2 * parametres->EcartType * parametres->EcartType)
    );

    // -> Commande
    deltaW = KXCONT * deltaX + KTHETACONT * deltaTheta;

    // Application de la loi de commande aux roues
    vg = initvitesse - RAYONROBOT * deltaW;
    vg = vg * 0.8 / 10;
    vg_int = floor(vg);
    vd = initvitesse + RAYONROBOT * deltaW;
    vd = vd * 0.8 / 10;
    vd_int = floor(vd);

    //printf("deltaX = %f deltaW = %f deltaFluxChapeau = %f vd = %d vg = %d deltaFlux = %f
",deltaX,deltaW,deltaFluxChapeau,vd_int,vg_int,deltaFlux);
    //printf("DeltaTheta = %f Flux = %d DeltaDist = %f \n\n", deltaTheta , ValeursCapteurs[numcapt] ,
deltaX);

    // -> Variation de l'angle pour la boucle suivante
    deltaTheta += TSOBS * deltaW;
    deltaTheta += KOBS * (deltaFlux - deltaFluxChapeau);

    D(&serial_1, vg_int, vd_int);
}
D(&serial_1,0,0);
}

/*****
* Fonction principale
*****/
int main (char argc, char *argv[])
{
    int mode;
    float distDesiree;
    struct DonneesCalibrage donnees;

    if (argc != 2)
    {
        printf("Utilisation : ./khepera <modele>\n");
        printf("Modele :\n");
        printf("\t1 -> continu \n");
        printf("\t2 -> discret \n");
        printf("\t3 -> continu avec observateur, un seul capteur actif\n");
        return 1;
    }
    sscanf(*(argv+1), "%d", &mode);

    // Parametres d'utilisation
    numcapt = 0;

    // Initialisation du robot
    printf("Initialisation...\n");
    initialisation();

    // Catche le signal STOP
    signal(SIGINT, stopHandler);

    // Parametres du modele
    donnees.EcartType=ECARTTYPE;
    donnees.AmplitudeMax=FLUXMAX;
    donnees.AngleMax=ANGLEMAX;
    donnees.A = donnees.AmplitudeMax * 2.5 * 2.5; // 2.5 : distance de calibrage

    // Initialisation des distances
    distDesiree = 30;

    if( mode == 1 || mode == 2 ) SuiviMur( &donnees , distDesiree , mode );
    else if ( mode == 3 ) SuiviMurObservateur( &donnees , distDesiree );

    return 0;
}

```