TP1 - Contrôle de robots unicycle et bicycle

Icare SAKR et Yan CHEN

Novembre 2020

1 Résumé du cours

Le cours présente d'abord le modèle cinématique qui lie le mouvement du robot par rapport au mouvement des actionneurs. Ainsi ce modèle exprime le mouvement du repère du robot par rapport au repère de référence. pour contrainte d'holonomie et de simplicité, il convient de bien placer les repères (repère de Frenet par exemple). Ce modèle est utilie à la stabilisation de trajectoire, le suivi de chemin et la stabilisation des configurations fixes. Pour atteindre ces objectifs avec précision, le cours présente le correcteur PID qui permet par boucle fermée de fournir une commande au robot qui soit proportionnelle à l'erreur commise (bien définie), à sa dérivée, et à son intégrale respéctivement par des coefficients (K_p, K_d, K_i) bien fixés (ex : Méthode de Ziegler-Nichols) pour répondre aux contraites de temps de réponse, de stabilité et d'erreur statique.

2 TP

2.1 Description

Dans ce TP, nous avons implémenter des contrôleurs PID de type P (proportionnels) pour le controle de robots unicycle et bicyclette.

Pour l'unicycle, nous allons présenter deux controlleurs P, un pour amener le robot à une position détérminée, et l'autre pour l'orientation.

Pour la bicyclette, nous allons d'abord implémenter un controleur proportionnel pour àmener le robot à un point donné, ensuite un controlleur pour amener le robot à une position (point + orientation), et finalement un controleur proportionnel permettant le suivi d'un chemin.

2.2 Contrôle de modèle unicycle

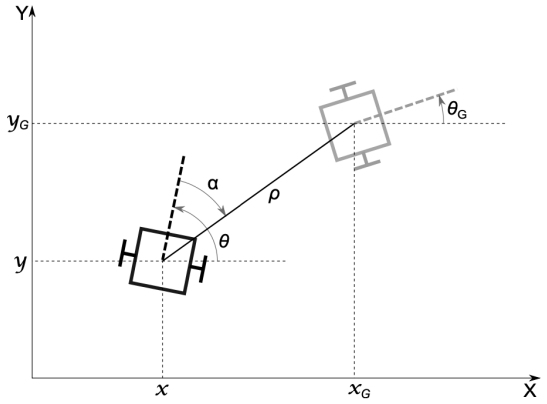


FIGURE 1 – Contrôle d'un unicycle vers une pose

Le modèle d'unicycle a une vitesse de translation limitée $\|v\| < 1m.s^{-1}$ et une vitesse de rotation limitée $\|\omega\| < \pi rad.s^{-1}$. De plus la accélération est elle aussi limitée $\|\frac{dv}{dt}\| < 10m.s^{-2}, \|\frac{d\omega}{dt}\| < 10rad.s^{-1}$.

Dans la figure 1, (x, y) et (x_G, y_G) sont les coordonnées actuelles et cible de centre robot. On définit la distance entre les deux positions est ρ, l'angle d'orientation actuelle et cible de robot est respectivement θ et θ_G. L'angle entre l'orientation de robot et ρ est α.

On définit règle de mouvement par rapport au-dessus.

✏ Loin du but ($\rho > 0.05$), la vitesse de translation et de rotation sont calculées par les équations suivantes.

$$\begin{aligned} \rho &= \sqrt{(x_G - x)^2 + (y_G - y)^2} \\ \alpha &= \arctan \frac{(y_G - y)}{x_G - x} - \theta \\ v &= K_\rho * \rho \\ \omega &= K_\alpha * \alpha \\ v &= 0, \quad si|\alpha| > \alpha_{max} \end{aligned} \tag{1}$$

✏ Lorsque le robot est proche du but ($\rho < 0.05$), le contrôleur utilisera la variable $\beta(\beta = \theta_G - \theta)$ pour calculer la vitesse de rotation :

$$\omega = K_\beta * \beta \tag{2}$$

Dans cette section, on va contrôler le robot unicycle pour arriver rapidement la pose. On optimise principalement les paramètres K_α, K_β et K_ρ. La vitesse de rotation est lié à α et K_α, si la paramètre K_α est très grand(par exemple $K_\alpha =$

1000), il va générer la oscillation d'angle sur trajectoire dans la figure 2. La même théorie, si la paramètre ρ est très grand(par exemple $\rho = 150$), il va générer petite oscillation de distance autour point cible, et le robot ne peut pas coïncider avec le point cible dans la figure 3. Ensuit, l'oscillation d'angle autour de point cible s'associe à la paramètres K_β, plus grand de valeur, plus de nombres d'oscillation. Donc la performance de mouvement de robot est sensible à les paramètres K_ρ, K_α, K_β.

Les essais ont montré que une valeur optimale du benchmark est obtenue pour $K_\rho = 15$, $K_\alpha = 10$, $K_\beta = 20$ et $\alpha_{max} = 1.2$, celui-ci étant de 1978.0476. Comme indiquée ci-dessous figure 4 et figure 5. Le robot peut arriver le point cible le plus rapidement, et aucune oscillation.

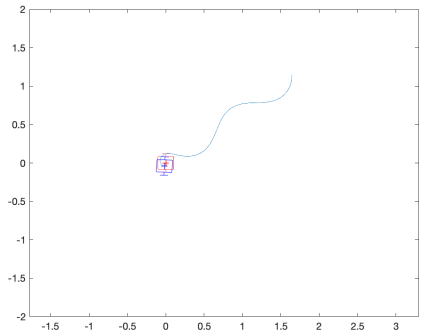


FIGURE 2 – Trajectoire de robot unicycle avec $\alpha = 1000$

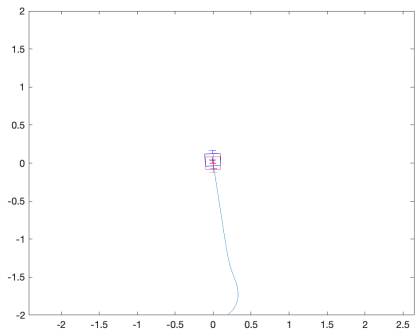


FIGURE 3 – Trajectoire de robot unicycle avec $\rho = 150$

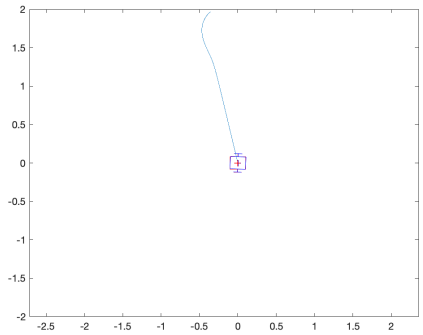


FIGURE 4 – Trajectoire de robot unicycle

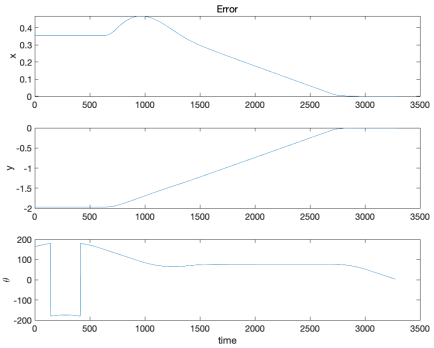


FIGURE 5 – Errors de variables de robot

Code :

```
function [ u ] = UnicycleToPoseControl( xTrue,xGoal )
%Computes a control to reach a pose for unicycle
%   xTrue is the robot current pose : [ x y theta ]'
```

```
%   xGoal is the goal point
%   u is the control : [v omega]'

Krho=15;
Kalpha=10;
Kbeta=20;
error=xGoal-xTrue;
rho=norm(error(1:2));
alpha = AngleWrap(atan2(error(2),error(1))-xTrue(3));
u(1) = Krho*rho;
if abs(alpha)>1.2
        u(1)=0;
end
u(2) = Kalpha*alpha;
if rho<0.05
    u(2) = Kbeta*error(3);
end
end
```

2.3 Contrôle de modèle bicyclette

2.3.1 Contrôle vers un point

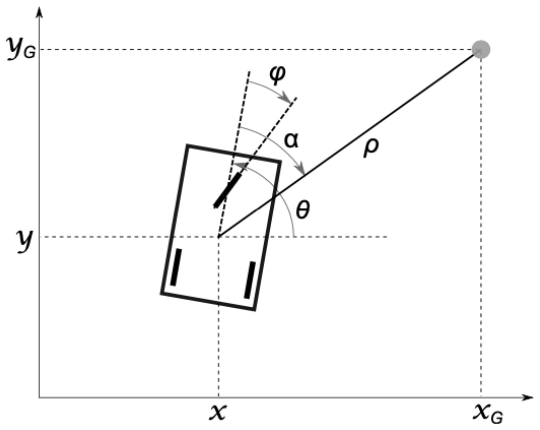


FIGURE 6 – Contrôle d'une modèle bicyclette vers un point

Dans cette section, on va contrôler le robot bicyclette pour arriver rapidement le point n'importe quoi l'orientation. On optimise principalement les paramètres K_α et K_ρ. Si la paramètre K_α est très grand (par exemple, $K_\alpha = 20$, $K_\rho = 100$), le robot va mouver en cycle autour le point cible dans quelques situations dans la figure 7 et la figure 8. Lorsque le robot arrive autour le point, la vitesse de translation est petite, si l'angle α et K_α sont grands, la vitesse de rotation est très grand, le robot n'est pas contrôlé bien. Donc il va faire une mouvement en cycle.

Les essais ont montré que une valeur optimale du benchmark est obtenue pour $K_\rho = 100$ et $K_\alpha = 4$, celui-ci étant de 1342.9524.

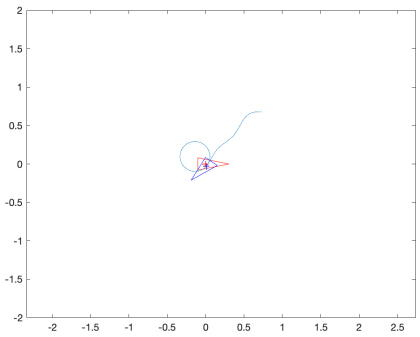


FIGURE 7 – Trajectoire bicyclette krho = 20, kalpha = 20

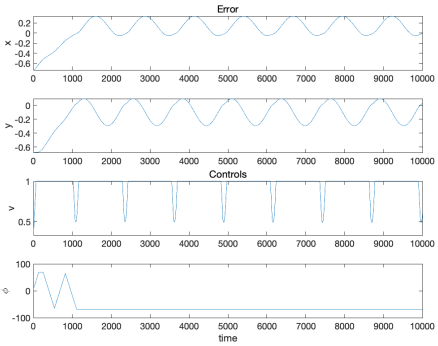


FIGURE 8 – Errors de variables krho = 20, kalpha = 20

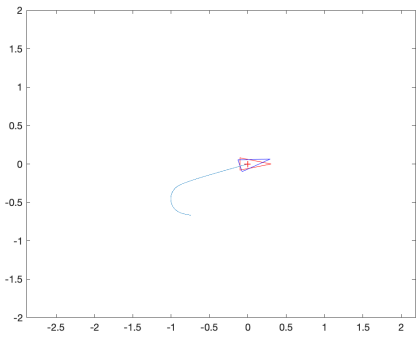


FIGURE 9 – Trajectoire bicyclette krho = 100, kalpha = 4

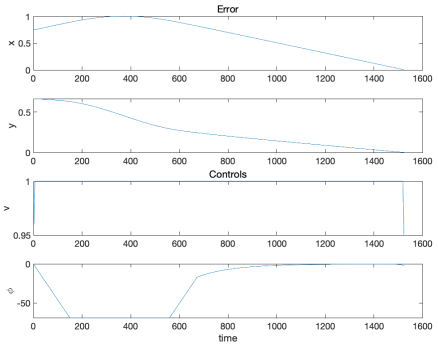


FIGURE 10 – Errors de variables krho = 100, kalpha = 4

Code :

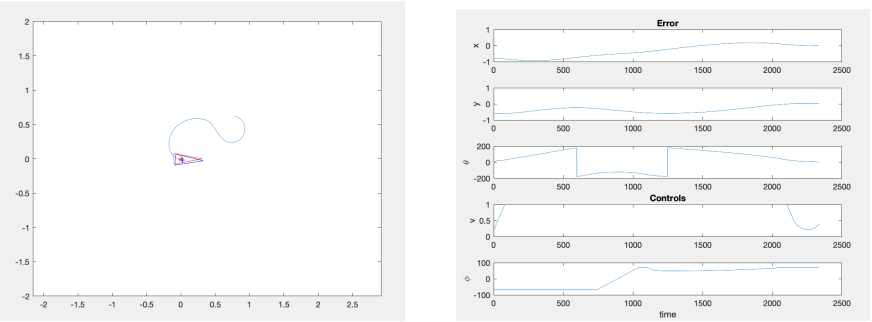
```
function [ u ] = BicycleToPointControl( xTrue,xGoal )
%Computes a control to reach a pose for bicycle
%   xTrue is the robot current pose : [ x y theta ]'
%   xGoal is the goal point
%   u is the control : [v phi]'

Krho = 100;
Kalpha = 4;
error = xGoal - xTrue;
rho = norm(error(1:2));
alpha = AngleWrap(atan2(error(2), error(1)) - xTrue(3));

u(1) = Krho * rho;
u(2) = Kalpha * alpha;

end
```

2.3.2 Contrôle vers une position

Dans cette section, nous allons implémenter et optimiser un contrôleur proportionnel, pour permettre au robot d'atteindre une position avec une orientation bien déterminée. Pour réaliser cela, nous allons envoyer la commande sur l'angle phi de la roue avant comme la somme pondérée de l'erreur α (sur la ligne qui lie le robot et le point désiré), et l'erreur β (commise par la ligne qui relie le robot et le point final, par rapport à l'angle final désiré). Le rapport entre k_α et k_β (qui devront avoir deux signes opposées) permettra de donner plus où moins d'importance au contrôle sur l'orientation désirée du système. Une valeur grande de k_α signifie que le robot atteindra rapidement la direction de la ligne qui relie son centre au point désiré, et une valeur grande de $|k_\beta|$ signifie que la direction de cette ligne atteindra rapidement l'angle désiré.

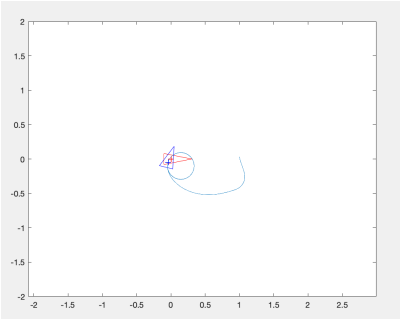
Les essais ont montré que une valeur optimale du benchmark est obtenue pour $k_\rho = 15$, $k_\alpha = 5$ et $k_\beta = -2.63$, celui-ci étant de 1657.57.

Les valeurs de k_α et k_β sont fortement liés et doivent être assez proches en valeur absolue car si par exemple on augmente k_α beaucoup plus que k_β le robot atteindra rapidement le point désiré mais n'aura pas le temps d'atteindre l'orientation désirée, et le système devient instable (oscillant) (voir figure ci-dessous). En augmentant k_ρ à k_β et k_α fixes, on observe une amélioration de performance mais pas à partir d'un certaine valeur, car l'accélération et la vitesse sont limités.

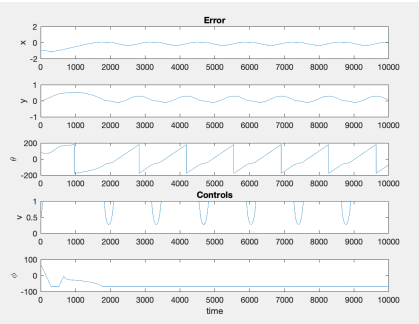


(a) Trajectoire (b) Erreurs et commandes

FIGURE 11 – Résultats trajectoires et erreurs, cas optimal ($k_\beta = 5 \sim k_\alpha$ = -2.63)



(a) Trajectoire cas instable



(b) erreurs cas instable

FIGURE 12 – Résultats trajectoires et erreurs, cas instable ($k_\beta = 9$ » k_α = -2.63)

Code :

```
function [ u ] = BicycleToPoseControl( xTrue,xGoal )
%Computes a control to reach a pose for bicycle
%   xTrue is the robot current pose : [ x y theta ]'
%   xGoal is the goal point
%   u is the control : [v phi]'

Krho = 10;
Kalpha = 5;
Kbeta = -2.63;


rho = sqrt((xGoal(1)-xTrue(1))^2+(xGoal(2)-xTrue(2))^2);

alpha = AngleWrap(atan2(xGoal(2)-xTrue(2), xGoal(1)-xTrue(1)) - xTrue(3));

beta = AngleWrap(xGoal(3) - atan2(xGoal(2)-xTrue(2), xGoal(1)-xTrue(1)));

v = Krho*rho;
phi = Kalpha*alpha + Kbeta*beta;

u = [v phi];


end
```

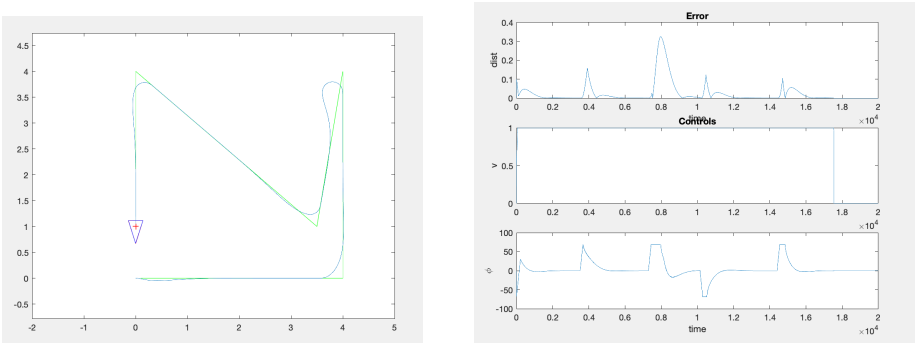
2.3.3 Contrôle selon un chemin

Finalement, nous avons implémenter un contrôleur P permettant à la bicyclette de suive une trajectoire déterminée. Pour cela, l'algorithme consiste à prendre à chaque itération de temps un point but sur la trajectoire qui soit situé à une distance $\rho = 0.5$ du robot (voir code ci dessous). A partir de ce point, on écrit un contrôleur proportionnel comme en 2.3.1, pour atteindre ce point. Içi l'orientation objective n'est pas prise en compte) et ceci car on a un problème de non-holonomie

et donc on ne peut pas avancer sur la courbe en étant perpendiculaire à la courbe par exemple (il faut des roues mecanum par exemple et 4 actionneurs ce qui n'est pas le cas). Si on avait pris dans notre cas aussi une contrainte d'orientation, le robot fera des allées retours autour du point pour essayer d'atteindre l'angle désiré et le système sera instable.

Pour l'algorithme, nous avons utilisé des variable persistantes (globales) *id* et *xGoal* permettant de mémoriser entre deux appels de la fonction de contrôle les états du point final de la sous-trajectoire et le point but (comme précédemment déterminé).

Les choix de k_ρ et k_α sont déterminés comme dans 2.3.1, et des valeurs $k_\rho = 100$, $k_\alpha = 2.3$, permettent d'obtenir une erreur (moyenne des distances entre le robot et le point le plus proche de la trajectoire.) de ~ 429.43.



(a) Trajectoire (b) Erreurs et commandes

FIGURE 13 – Résultats suivi de trajectoire

Nous remarquons qu'au changement de sous trajectoire, nous obtenons des pics d'erreurs dues à l'éloignement du robot de la trajectoire pour pouvoir changer de direction. Intuitivement, le choix de ρ (0.5 dans notre cas) est lié à l'angle maximal que peut effectuer la roue avant du robot pour ne pas trop sortir de la trajectoire à chaque fois qu'on veut changer de sous trajectoire.

Code :

```
function [ u ] = BicycleToPathControl( xTrue, Path )
%Computes a control to follow a path for bicycle
%   xTrue is the robot current pose : [ x y theta ]'
%   Path is set of points defining the path : [ x1 x2 ... ;
%                                               y1 y2 ...]
%   u is the control : [v phi]'

persistent id xGoal;

if xTrue == [0;0;0]
    id = 1;
    xGoal = Path(:,1);
end
```

```
%check if we are close to the end of the subpath
error = Path(:,id) - xTrue;
rho = norm(error(1:2));

if rho < 0.5 %Error between xTrue end the end point of the subpath
    xGoal = Path(:, id);
    id = id + 1;
    id = min(id, size(Path,2));

else
    diff = Path(:, id) - Path(:, id-1);
    diff = diff/norm(diff);

    error = xGoal - xTrue;
    goalDist = norm(error(1:2));

    while goalDist <  0.5
        xGoal = xGoal + 0.01*diff;
        error = xGoal - xTrue;
        goalDist = norm(error(1:2));
    end
end

Krho = 100;
Kalpha = 2.3;

error = xGoal - xTrue;
rho = norm(error(1:2));
alpha = AngleWrap(atan2(error(2), error(1)) - xTrue(3));

u(1) = Krho * rho;
u(2) = Kalpha * alpha;


end
```