ROB316 - TD1

BRAMBILLA Davide Luigi - GOMES DA SILVA Rafael

2 Décembre 2019

1 Le cours

Le cours présente le **modèle cinématique** qui lie les différents parties du robot au travers de relations d'équivalence. Il permet donc d'exprimer le mouvement d'un *repère*, celui du robot, par rapport au *repère de référence*. Parmi les applications des modèles cinématiques on trouve la **stabilisation de la trajectoire**, le **suivi de chemin**, la **stabilisation des configurations fixes**. Enfin le correcteur **PID** a été expliqué: au travers un boucle fermé il est possible d'utiliser la valeur de l'*erreur*, définie comme la différence entre le comportement atteint et le comportement espéré, sa dérivé et son intégral pour corriger le comportement du robot. En particulier, chacun des trois valeurs sera multiplié par une constante $(k_p, \kappa_d, \kappa_i)$ qui servira à faire une pondération entre les trois effets: *proportionnel*, *dérivé* et *intégral*.

2 TP

Dans le TP, il nous est demandé de implémenter des contrôleurs *proportionnels* afin de conduire des robots unicycle et bicyclette vers un point désiré avec une orientation désirée.

Pour un *robot unicycle* nous implémenterons un contrôleur de type P pour l'amener vers une position précis avec un orientation précis.

Pour un *robot bicyclette* nous avons implementé trois différents contrôleurs:

- le premier pour amener le robot vers une position désirée
- le deuxième pour amener le robot vers la position désirée avec l'orientation désiré
- le troisième pour faire suivre au robot une trajectoire prédéfinie

2.1 Q1 - Contrôle Uni-cycle

Dans cette section, nous devons implémenter deux contrôleurs proportionnels: le premier afin de conduire le robot sur la position du but et un deuxième pour l'ajustement de son orientation.
Nous allons utiliser les variables ρ et α, comme indiqué dans le sujet de TD, afin de pouvoir constituer le contrôle.
Notre contrôleur aura le comportement suivant: si le robot se trouve loin de la valeur désirée il y aura soit une correction sur sa position soit sur son orientation sauf dans le cas l'orientation est plus grande que $1.2rad$ où il y aura une correction exclusivement sur l'orientation.
Si le robot se trouve proche de la position désiré on donnera plus d'importance à la correction de sa direction.

Influence des coefficients

Pour ce que concerne les coefficients, k_ρ donnera, en comparaison avec k_α, l'importance qui on donne au contrôle sur la position par rapport au contrôle sur l'orientation quand on se trouve loin de la position désirée $(\rho > 0.05)$.

Enfin, k_β va choisir la rapidité avec laquelle l'objet, une fois arrivé en proximité $(\rho < 0.05)$ de la position finale, ajustera son orientation.

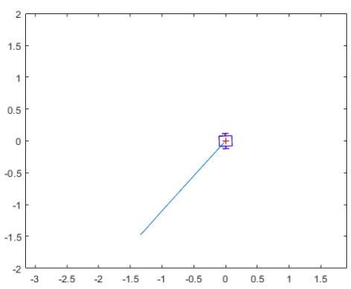
Des valeurs trop petites rendront le systeme lent pour arriver à la convergence et des valeurs des coefficients trop grandes pourront amener à des oscillations et les valeurs de benchmark pour des valeurs très grands ne sont pas satisfaisantes.

Le valeurs obtenues

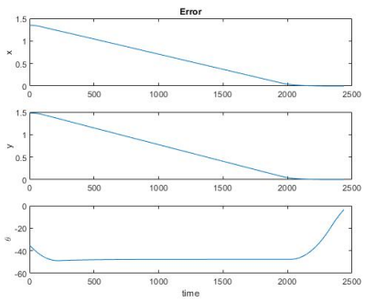
Ci-dessus j'ai reporté les valeurs de Benchmark obtenues avec différents valeurs des coefficients.

α_{max}	k_ρ	k_α	k_β	valeur de Benchmark
1.2	5	5	5	7316.381
1.2	5	5	10	7205.809
1.2	10	5	5	2585.524
1.2	10	10	20	2023.285
1.2	10	5	10	1909.904

Nous avons noté que les valeurs avec les résultats meilleurs étaient lesquels qui donnaient, au début, moins d'importance au correction sur l'orientation par rapport à la correction sur la position. Parmi les valeurs essaies, la meilleure valeur pour le benchmark nous la obtenons avec $\alpha_{max} = 1.2$, $k_\rho = 10$, $k_\alpha = 5$ et $k_\beta = 10$. Cela nous fait avoir une trajectoire très rapide vers le point finale où le robot se dirigera rapidement vers le point désiré où, en suite, modifiera son orientation.



(a) La trajectoire suivi



(b) Erreur sur les états

Figure 1

Fichier 1: UnicycleToPoseControl.m

```
Kr = 10;
Ka =  5;
Kb = 10;
alfamax = 1.2;

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

u(1) = Kr * rho;
if abs(alfa) > alfamax
   u(1) = 0;
end

if rho > 0.05
    u(2) = Ka * alfa;
else
    u(2) = Kb * beta;
end
```

2.2 Q2 - Contrôle de Bicyclette vers un point

Pour choisir le réglage correct pour les gains k_α et k_ρ, nous faisons varier ces gains entre 0 et 40, tout en vérifiant les effets de chaque gain sur la réponse du système.

Influence des coefficients

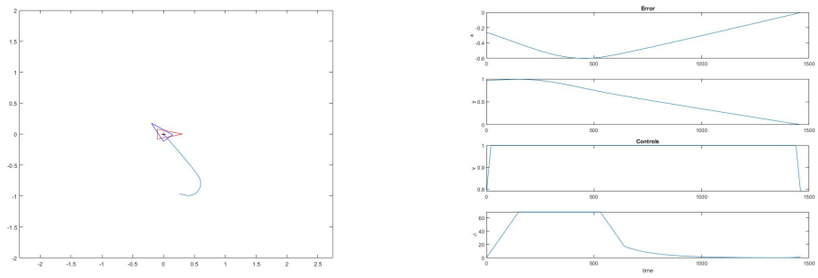
Le gain k_α est lié à la orientation de la bicyclette. Des valeurs très élevées de k_α entraînent une oscillation dans la trajectoire suivie par la bicyclette, car ce gain est directement lié à la correction de l'erreur de trajectoire angulaire, et chaque petit variation est amplifié par ce gain. En revanche, une valeur de k_α petite retarde la convergence vers le point souhaité car la contribution de l'action de contrôle du régulateur sera moins importante.

Le gain k_ρ est lié à l'erreur de distance entre la position de la bicyclette et laquelle du point souhaité. Pour les petites valeurs, le robot bicyclette prend plus de temps pour arriver au point: en utilisant le *BiclycleToPointBenchmark*, nous avons obtenu 5757.9048 avec $k_\rho = 1$ et 1300 avec $k_\rho = 100$). Une remarque importante est que la modification des valeurs de ce gain ne provoque pas d'oscillation dans le système, mais d'autre part, lorsque la valeur est trop petite, la bicyclette atteint une vitesse nulle avant d'atteindre le point souhaité.

Par conséquent, compte tenu des effets des deux gains, l'ajustement choisi a été effectué pour avoir la valeur minimale possible pour k_α afin que le système n'oscille pas ($\boldsymbol{k_\alpha = 2.3}$) et une valeur de k_ρ suffisamment grande pour empêcher la bicyclette d'avoir une vitesse nulle avant de terminer l'objectif ($\boldsymbol{k_\rho = 30}$). En utilisant le *BiclycleToPointBenchmark*, nous avons obtenu **1302.1905** pour cette réglage. Ainsi, le code ci-dessous montre l'implémentation du régulateur fait pour le contrôle de bicyclette vers un point.

Fichier 2: BicycleToPointControl.m

```
Kr = 30;
Ka =  2.3;

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

u(1) = Kr * rho;
u(2) = Ka * alfa;
```



(a) La trajectoire suivi

(b) Érrerus sur les états et commandes

Figure 2

2.3 Q3 - Contrôle de bicyclette vers une position avec une certaine orientation

Dans cette section, le contrôleur devra assurer le robot bicyclette d'aller vers un point précis et avoir une orientation précise.

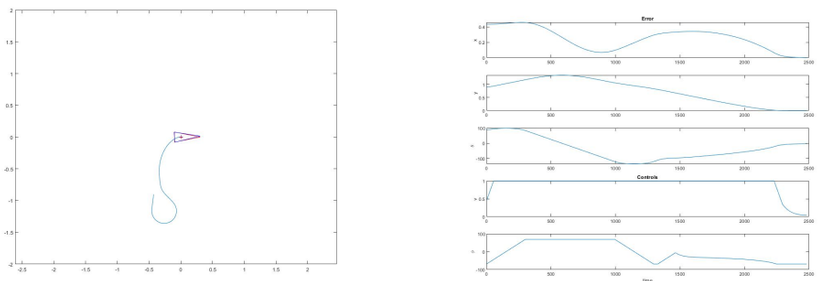
Pour faire cela, nous avons implémenté le contrôle définie dans le sujet du TD et nous avons réglé les différents coefficients présents afin d'obtenir la valeur désirée.

Influence des coefficients

Dans ce cas, le coefficient k_ρ donne importance au contrôle sur la position.

Dans ce cas, le contrôle sur l'orientation est donné par la comparaison entre la valeur de α, qui donne un information sur l'angle courant, et la valeur de β qui donne un information par rapport à l'orientation que on veut avoir à la fin.
En particulier le rapport entre k_α et k_β (qui devront avoir deux signes opposés) permettra de donner plus où moins d'importance au contrôle sur l'orientation du système.

Les valeurs qui nous ont permis d'obtenir le meilleur benchmark ont été $k_\rho = 15$, $k_\alpha = 10$ et $k_{beta} = -5$ qui nous ont fait obtenir une valeur de 1739.



(a) La trajectoire suivi

(b) Erreur sur les états et commandes

Figure 3

Fichier 3: BicycleToPoseControl.m

```
Kr = 15;
Ka =  10;
Kb = -5;

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

```
u(1) = Kr * rho;
u(2) = Ka * alfa + Kb * beta;
```

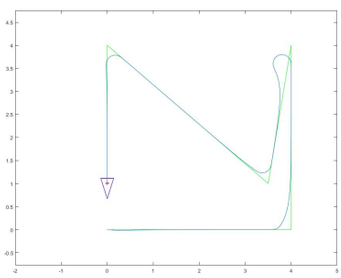
2.4 Q4 - Contrôle de bicyclette selon un chemin

Dans cette dernière question, nous traitons une application des notions utilisées dans la question Q2, où une bicyclette doit suivre le chemin souhaité. Pour cela, nous déterminons le point de départ de la bicyclette et à chaque pas de temps un nouveau point objectif est déterminé afin d'avoir toujours une distance entre la bicyclette et la trajectoire souhaitée inférieure à 0.5 . Autrement dit, le but de la loi de contrôle est de réduire la distance entre le robot et le chemin à suivre et l'erreur de l'orientation. Dans le code nous avons utilisés les variables *id* et *xGoal* du type *"persistent"* pour traiter les variables de la fonction *BicycleToPathControl* de façon itérative: comme cela nous pourrons conserver en mémoire les valeurs de ces variables entre différents appels à la fonction.

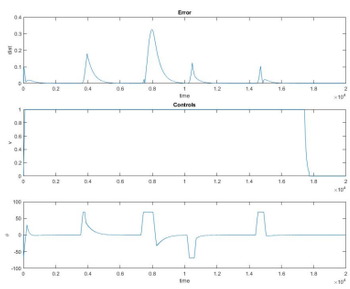
Pour la réglage des gains, nous avons utilisé la même méthode suivi dans la question Q2, e nous avons pu constaté les mêmes remarques concernant les variations des gains et les effets liés a ces changements, et nous avons aussi pu vérifier que les valeurs de gains optimales dans ce cas étaient les mêmes utilisés précédemment ($\boldsymbol{k_\alpha = 2.3}$ et $\boldsymbol{k_\rho = 30}$).

Fichier 4: BicycleToPathControl.m

```
persistent id xGoal;

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

error = Path(:,id) - xTrue;
rho = norm(error(1:2));

if rho < 0.5
    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

Kr = 30;
Ka =  2.3;

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

u(1) = Kr * rho;
u(2) = Ka * alfa;
```



(a) La trajectoire suivi



(b) Erreur sur les états et commandes

Figure 4