INSTITUT POLYTECHNIQUE DE PARIS
ENSTA PARIS

CSC_5RO16_TA, Planification et Contrôle

TP3, Côntrole PID

by
Guilherme NUNES TROFINO

supervised by
Julien ALEXANDRE DIT SANDRETTO
David FILLIAT

ROBOTIQUE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET COMMUNICATION

Paris, FR
15 janvier 2025

Table des matières

1. Méthodologie

Dans ce Travaux Pratique, la composante proportionnelle du contrôle PID sera étudiée en détails, appliquée aux modèles de robots unicycle et bicyclette. Afin de détérminer les constantes nécessaires pour chaque algorithme, des essais statistiques ont été réalisés afin d'identifier des valeurs appropriées pour ces constantes.

1.1. Exploration

Dans un premier temps, les scripts fournis ont été modifiés afin de pourvoir être utilisés en boucle, permettant ainsi de tester plusieurs combinaisons de variables.

1.1.1. Algorithme `Control`

L'algorithme de contrôle a également été ajusté pour accepter toutes les constantes nécessaires à son fonctionnement, comme illustré ci-dessous :

```
function [ u ] = UnicycleToPoseControl( xTrue, xGoal, alpha_maximum, K_rho, K_alpha, K_beta )
```

Listing 1 : `UnicycleToPoseControl.m` algorithme

> **Remarque.** Aucune autre changement n'a été apporté au code.

1.1.2. Algorithme `Benchmark`

L'algorithme de benchmark a été modifié pour recevoir toutes les constantes nécessaires au fonctionnement de l'algorithme de contrôle et retournait les performances mesurées des exécutions, comme illustré ci-dessous :

```
function [ mean_performance ] = UniclycleToPoseBenchmark( alpha_maximum, K_rho, K_alpha, K_beta )
```

Listing 2 : `UnicycleToPoseBenchmark.m` algorithme

> **Remarque.** Aucune autre changement n'a été apporté au code.

1.1.3. Algorithme `Explore`

Après ces modifications, un nouveau fichier a été créé pour exécuter plusieurs essais avec l'algorithme, comme montré ci-dessous :

```
function UnicycleExplore
    for alpha_maximum = pi/4 : pi/4 : pi
        for K_rho = 5 : 10 : 25
            for K_alpha = 5 : 10 : 25
                for K_beta = 5 : 10 : 25
                    mean_performance = UnicycleToPoseBenchmark(...
                        alpha_maximum,...
                        K_rho,...
                        K_alpha,...
                        K_beta...
                    );
                    disp([...
                        num2str(alpha_maximum), ',',...
                        num2str(K_rho), ',',...
                        num2str(K_alpha), ',',...
                        num2str(K_beta), ',',...
                        num2str(mean_performance)...
                    ]);
                endfor
            endfor
        endfor
    endfor
end;
```

Listing 3 : `UnicycleExplore.m` algorithme

Les résultats de chaque exécution ont été affichés dans le terminal au format CSV afin de faciliter leur analyse.

1.2. Analyse

Une fois l'exécution terminée, les résultats ont été sauvegardés au format CSCV et analysés. Si un résultat satisfaisant était trouvé, il était retenu comme solution. Sinon, de nouvelles itérations étaient réalisées pour tenter d'identifier des valeurs appropriées.

Les intervalles d'analyse ont commencé de manière restreinte, puis ont été élargis progressivement.

> **Remarque.** Les fonctions de l'unicycle sont présentées ici comme exemple, mais le processus appliqué est identique pour les autres modèles.

2. Question 1 - Unicycle, Contrôle Proportionnel Position

2.1. Description

Pour cette question, un contrôleur **proportionnel** a été implémenté pour un unicycle : pour la position et l'orientation. Ce contrôleur suit les équations présentées ci-dessous :

1. **Position** :

$$v = \begin{cases} K_\rho \times \underbrace{\sqrt[2]{(x_G - x)^2 + (y_G - y)^2}}_{\rho}, & \text{if} \quad |\alpha| \leq \alpha_{\max} \\ 0, & \text{if} \quad |\alpha| > \alpha_{\max} \end{cases} \tag{2.1}$$

2. **Orientation** :

$$\omega = \begin{cases} K_\alpha \times \underbrace{\left(\arctan\left(\frac{y_G - y}{x_G - x}\right) - \theta\right)}_{\alpha}, & \text{if} \quad \rho > 0.05 \\ K_\beta \times \underbrace{(\theta_G - \theta)}_{\beta}, & \text{if} \quad \rho \leq 0.05 \end{cases} \tag{2.2}$$

2.2. Algorithme

Ensuite, l'algorithme suivant a été implémenté pour intégrer ces contrôleurs :

```
function [ u ] = UnicycleToPoseControl( xTrue, xGoal, alpha_maximum, K_rho, K_alpha, K_beta )
  %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]'

  x = xTrue(1);
  y = xTrue(2);
  theta = xTrue(3);

  x_delta = xGoal(1) - x;
  y_delta = xGoal(2) - y;

  rho = sqrt( (x_delta)^2 + (y_delta)^2 );
  alpha = AngleWrap( atan2(y_delta, x_delta) - theta );

  v = K_rho * rho;

  if abs(alpha) > alpha_maximum
    v = 0;
  endif

  if rho > 0.05
    omega = K_alpha * alpha;
  else
    omega = K_beta * AngleWrap( xGoal(3) - theta );
  endif

  u = [v, omega];
end
```

Listing 4 : UnicycleToPoseControl.m

2.3. Résultats

Après une série d'essais statistiques, les meilleures performances observées étaient de **1936.1429**, obtenues avec les paramètres suivants : `alpha_maximum = 0.7854`, `K_rho = 15`, `K_alpha = 5` et `K_beta = 25`.

Ces valeurs optimisées ont permis d'atteindre un compromis efficace entre précision et rapidité du contrôle.

3. Question 2 - Bicyclette, Contrôle Proportionnel Point

3.1. Description

Pour cette question, un contrôleur **proportionnels** a été implémenté pour un bicycle : pour la position. Ce contrôleur suit les équations présentées ci-dessous :

1. **Position** :

$$v = K_\rho \times \underbrace{\sqrt[2]{(x_G - x)^2 + (y_G - y)^2}}_{\rho} \tag{3.1}$$

2. **Orientation** :

$$\phi = K_\alpha \times \underbrace{\left(\arctan\left(\frac{y_G - y}{x_G - x}\right) - \theta\right)}_{\alpha} \tag{3.2}$$

3.2. Algorithme

Ensuite l'algorithme suivant a été implémenté pour intégrer ce contrôleur :

```
function [ u ] = BicycleToPointControl( xTrue, xGoal, K_rho, K_alpha )
  %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]'

  x = xTrue(1);
  y = xTrue(2);
  theta = xTrue(3);

  x_delta = xGoal(1) - x;
  y_delta = xGoal(2) - y;

  rho = sqrt( (x_delta)^2 + (y_delta)^2 );
  alpha = AngleWrap( atan2(y_delta, x_delta) - theta );

  v = K_rho * rho;
  phi = K_alpha * alpha;

  u = [v, phi];
end
```

Listing 5 : BicycleToPointControl.m

3.3. Résultats

Après une série d'essais statistiques, les meilleurs performances obsrvées étaient **1349.619**, obtenues avec les paramètres suivants : `K_rho = 25` et `K_alpha = 5`.

Ces valeurs optimisées ont permis d'atteindre un compromis efficace entre récision et rapidité du contrôle.

4. Question 3 - Bicyclette, Contrôle Proportionnel Position

4.1. Description

Pour cette question, un contrôleur **proportionnel** a été implémenté pour un bicycle : pour la position et l'orientation. Ces contrôleur suit les équations présentées ci-dessous :

1. **Position** :

$$v = K_\rho \times \underbrace{\sqrt[2]{(x_G - x)^2 + (y_G - y)^2}}_{\rho} \tag{4.1}$$

2. **Orientation** :

$$\phi = K_\alpha \times \underbrace{\left(\arctan\left(\frac{y_G - y}{x_G - x}\right) - \theta\right)}_{\alpha} + K_\beta \times \underbrace{\theta_G - \left(\arctan\left(\frac{y_G - y}{x_G - x}\right)\right)}_{\beta<0} \tag{4.2}$$

4.2. Algorithme

Ensuite, l'algorithme suivant a été implémenté pour intégrer ces contrôleurs :

```
function [ u ] = BicycleToPoseControl( xTrue, xGoal, K_rho, K_alpha, K_beta )
  %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]'

  x = xTrue(1);
  y = xTrue(2);
  theta = xTrue(3);

  x_delta = xGoal(1) - x;
  y_delta = xGoal(2) - y;

  rho = sqrt( (x_delta)^2 + (y_delta)^2 );
  alpha = AngleWrap( atan2(y_delta, x_delta) - theta );
  beta = AngleWrap( xGoal(3) - atan2(y_delta, x_delta) );

  v = K_rho * rho;
  phi = K_alpha * alpha + K_beta * beta;

  u = [v, phi];
end
```

Listing 6 : BicycleToPoseControl.m

4.3. Résultats

Après une série d'essais statistiques, les meilleures performances observées étaient de **1735.8571**, obtenues avec les paramètres suivants : `K_rho = 25`, `K_alpha = 5` et `K_beta = -3`.

Ces valeurs optimisées ont permis d'atteindre un compromis efficace entre récision et rapidité du contrôle.

5. Question 4 - Bicyclette, Contrôle Proportionnel Chemin

5.1. Description

Pour cette question, un contrôleur **proportionnel** a été implémenté pour un bicycle : pour la position. Ces contrôleur suit les équations présentées ci-dessous :

1. **Position** :

$$v = K_\rho \times \underbrace{\sqrt[2]{(x_G - x)^2 + (y_G - y)^2}}_{\rho} \tag{5.1}$$

2. **Orientation** :

$$\phi = K_\alpha \times \underbrace{\left(\arctan\left(\frac{y_G - y}{x_G - x}\right) - \theta\right)}_{\alpha} \tag{5.2}$$

5.2. Algorithme

Ensuite, l'algorithme suivant a été implémenté pour intégrer ces contrôleurs :

```
function [ u ] = BicycleToPathControl( xTrue, Path, K_rho, K_alpha, rho_threshold )
%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]'

    x = xTrue(1);
    y = xTrue(2);
    theta = xTrue(3);

    % initialize persistent
    persistent CURRENT_VERTICE;
    persistent CURRENT_POINT;
    persistent PATH_POINTS;

    if isempty(CURRENT_VERTICE)
        CURRENT_VERTICE = 1;
    endif

    if CURRENT_VERTICE == size(Path, 2) || isempty(PATH_POINTS)
        CURRENT_VERTICE = 1;

        x_points = linspace(Path(1, CURRENT_VERTICE), Path(1, CURRENT_VERTICE + 1), 20);
        y_points = linspace(Path(2, CURRENT_VERTICE), Path(2, CURRENT_VERTICE + 1), 20);

        PATH_POINTS = [[ x_points; y_points ]];
    endif

    if isempty(CURRENT_POINT)
        CURRENT_POINT = 1;
    endif

    % update target points
    if CURRENT_POINT <= size(PATH_POINTS, 2)
        current_target = PATH_POINTS(:, CURRENT_POINT);
        x_g = current_target(1);
        y_g = current_target(2);

        % reached current target?
        if sqrt((x - x_g)^2 + (y - y_g)^2) < rho_threshold
            CURRENT_POINT = CURRENT_POINT + 1;
        endif
    else
        if (CURRENT_VERTICE + 1) == size(Path, 2)
            x_g = Path(1, end);
            y_g = Path(2, end);
        else
            CURRENT_POINT = 1;
            CURRENT_VERTICE = CURRENT_VERTICE + 1;

            x_points = linspace(Path(1, CURRENT_VERTICE), Path(1, CURRENT_VERTICE + 1), 20);
```

```
53              y_points = linspace(Path(2, CURRENT_VERTICE), Path(2, CURRENT_VERTICE + 1), 20);
54
55              PATH_POINTS = [[ x_points; y_points ]];
56
57              current_target = PATH_POINTS(:, CURRENT_POINT);
58
59              x_g = current_target(1);
60              y_g = current_target(2);
61          endif
62      endif
63
64      % command
65      rho = sqrt((y_g - y)^2 + (x_g - x)^2);
66      alpha = AngleWrap( atan2(y_g - y, x_g - x) - theta );
67
68      v = K_rho * rho;
69      phi = K_alpha * alpha;
70
71      u = [ v phi ];
72  end
```

Listing 7 : BicycleToPathControl.m

5.3. Résultats

Après une série d'essais statistiques, les meilleures performances observées étaient de **398.9695**, obtenues avec les paramètres suivants : `rho_threshold = 0.3`, `K_rho = 6` et `K_alpha = 6`. Le chemin est montré ci-dessous :

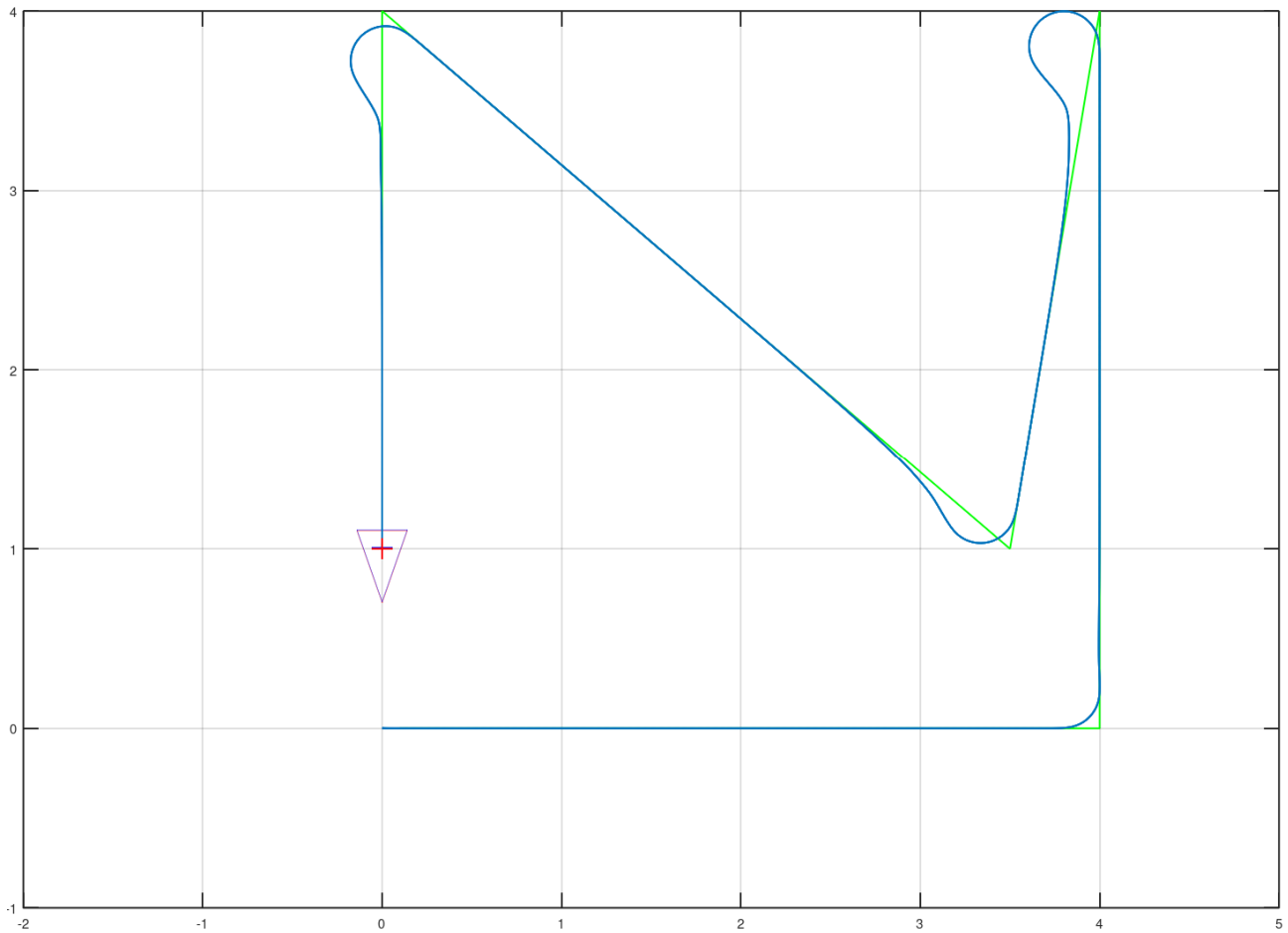

FIGURE 5.1 : Chemin Finale

Ces valeurs optimisées ont permis d'atteindre un compromis efficace entre récision et rapidité du contrôle.