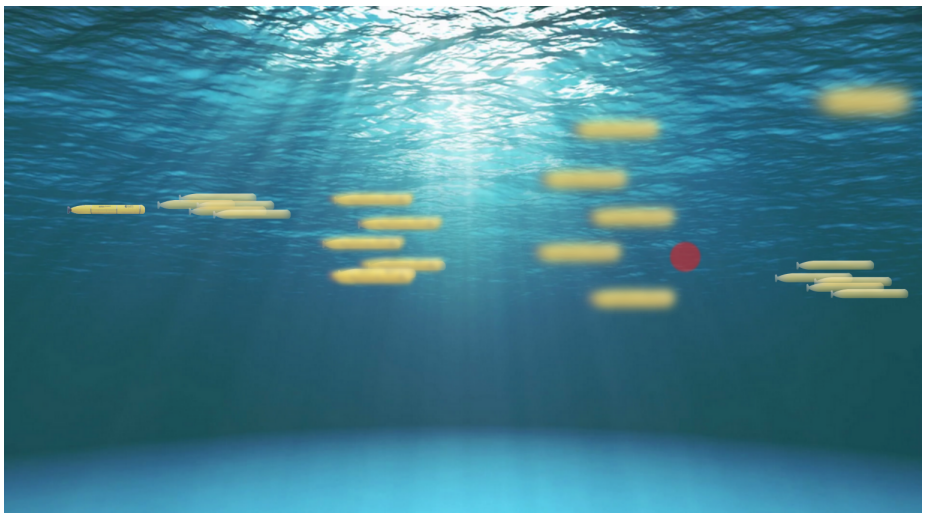
BE UV 5.7 – *Predictor*

28/11/2018

Lien vers le site web



Suivi de Champ de Vecteurs & Linéarisation par Bouclage

Joris Tillet
Responsable : Luc Jaulin

Objectifs

- Comprendre la méthode de linéarisation par bouclage (*feedback linearization*) pour le contrôle commande,
- Se familiariser avec des notions de géométrie différentielle pour monter en niveau d'abstraction,
- Utiliser la bibliothèque Python de calcul formel `Sympy`,
- Comprendre l'intérêt du suivi de champ de vecteurs.

Table des matières

1 Linéarisation par Bouclage

1.1 Rappel : Exemple simple

On choisit de commander une voiture de Dubins. Le système est décrit par les équations suivantes :

$$\begin{cases} \begin{pmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{pmatrix} = \begin{pmatrix} \cos(x_3) \\ \sin(x_3) \\ u \end{pmatrix} \\ y = x_3 \end{cases}, \tag{1}$$

avec (x_1, x_2) la position de la voiture et x_3 son cap.

On mesure donc son cap, et on désire maintenant le réguler. Pour cela, on calcule les dérivées successives de y jusqu'à faire apparaître u. Dans notre exemple, il suffit de le dériver une fois, puisque $\dot{y} = \dot{x_3} = u$.

Si on souhaite que le cap converge vers 0, il faut donc choisir la relation entre y et $\dot{y}$ tel que y tende vers 0. On prend donc l'équation différentielle suivante :

$$\dot{y} + y = 0.$$

On a donc $u = \dot{y} = -y = -x_3$. La commande $u = -x_3$ placée en entrée du système fera donc converger la sortie $y = x_3$ vers 0 de manière exponentielle [1] [2].

Plus généralement, on souhaite que notre voiture suive le cap décrit par la fonction w. Notre erreur s'exprime donc comme $e = w - y$, et c'est l'erreur que l'on veut faire converger vers 0 cette fois-ci. On réécrit notre équation différentielle en e :

$$\dot{e} + e = 0.$$

Or $\dot{e} = \dot{w} - \dot{y}$ donc $\dot{w} - \dot{y} + w - y = 0$. Et comme $\dot{y} = u$ et $y = x_3$, on a finalement notre commande :

$$u = \dot{w} + w - x_3.$$

On retrouve donc bien notre commande $u(t) = -x_3(t)$ calculée ci-dessus, si on choisit $w(t) = 0$ pour tout t. Si maintenant on veut suivre un cap en sinus, on obtient la commande $u(t) = \cos(t) + \sin(t) - x_3(t)$.

1.2 Méthode générale

On considère un système non linéaire décrit par les équations :

$$\begin{cases} \dot{x} = & f(x, u) \\ y = & g(x) \end{cases} \tag{2}$$

On calcule les dérivées successives de y jusqu'à faire apparaître u. Le nombre de fois que l'on a dû dériver y s'appelle le *degré relatif*, que l'on note n. On peut donc exprimer u en fonction de $y^{(n)}$.

Ensuite, pour faire converger y vers une fonction w, on cherche à faire converger la fonction d'erreur $e(t) = y(t) - w(t)$ vers 0. Pour cela, on force e à être solution de l'équation différentielle en $e^{(n-k)}$ pour k allant de 1 à n, en plaçant tous les pôles à -1. De cette façon, on garantit que e convergera vers 0 de manière exponentielle. Pour le placement des pôles, on pourra utiliser le triangle de Pascal, présenté Figure 1.

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 1 & & & \\ & & 1 & & 2 & & 1 & & \\ & 1 & & 3 & & 3 & & 1 & \\ 1 & & 4 & & 6 & & 4 & & 1 \end{array}$$

FIGURE 1 – Triangle de Pascal

Exemple pour $n = 3$: On choisit l'équation différentielle $\dddot{e} + 3\ddot{e} + 3\dot{e} + e = 0$. On a donc d'une part $\dddot{e} = -3\ddot{e} - 3\dot{e} - e$ qui ne dépend pas de u, mais seulement de l'état x et de w, et d'autre part, $\dddot{e}$ en fonction de u (puisqu'on a supposé le degré relatif égal à 3). On a donc une expression pour u qui est une commande qui fera converger e vers 0 si on la met en entrée du système.

1. La solution de l'équation différentielle est $y(t) = y(0)\exp(-t)$.
2. On remarque que la commande trouvée ici est une simple commande proportionnelle.

2 Géométrie différentielle – Dérivées de Lie

2.1 Lien avec la linéarisation par bouclage

Pour mieux comprendre ce qu'il se passe mathématiquement, on peut reprendre de façon plus formelle le système d'équations 2. Lorsque l'on dérive y, on obtient :

$$\begin{aligned}\frac{\partial y}{\partial t} &= \frac{\partial x}{\partial t} \cdot \frac{\partial g(x)}{\partial t} \\ &= f(x,u) \cdot \frac{\partial g(x)}{\partial t} \\ &= \mathcal{L}_f\left(g(x)\right).\end{aligned}$$

Ce qui s'appelle une dérivée de Lie, ou *Lie derivative* en anglais. Il existe donc un outil mathématique pour faire ce que l'on fait. Dans la suite, on va essayer de mieux comprendre cet outil afin de l'utiliser dans du calcul formel et ainsi gagner du temps [3].

2.2 Définitions et notations

Les définitions suivantes ont volontairement été simplifiées et ne sont donc pas toutes mathématiquement rigoureuses. Les notes de bas de page les complètent, mais utilisent des notions mathématiques non expliquées ici.

Variété : C'est un espace topologique qui ressemble localement à un espace euclidien [4]. En anglais, cela s'appelle : ***manifold***.

Variété différentielle : Objet de base de la géométrie différentielle dans lequel on se place. C'est une variété où il existe des **systèmes de coordonnées** locales [5] et où il est possible d'effectuer les opérations du calcul différentiel et intégral. En anglais, cela s'appelle : *differentiable manifold*.

Champ de vecteurs : Application f qui à un point x associe un vecteur [6]. La fonction d'évolution f d'un système ($\dot{x}(t) = f(x(t), u(t))$) est donc un champ de vecteur de la même dimension que le vecteur d'état.

Flot : Notion associée à la notion de champ de vecteurs. Un flot peut être vu comme la trajectoire d'un point suivant un champ de vecteurs à tout instant.

Dérivée de Lie : Elle peut être vue comme le taux de variation (dérivée) du flot en un point. On note $\mathcal{L}_f(x)$ la dérivée de Lie de x par rapport au champ de vecteur f, où x est une fonction du temps (on ne dérive pas un point). On a donc : $\mathcal{L}_f(x) = f(x) \cdot \frac{\partial x}{\partial t}$.

3. et surtout éviter des erreurs de calcul.
4. Pour tout point x appartenant à une variété de dimension n, x a un voisinage qui est homéomorphique à l'espace Euclidien de dimension n.
5. définis par les homéomorphismes locaux.
6. avec x appartenant à un ouvert d'un espace de Banach E, et $f(x)$ appartenant à E.

2.3 Lien avec Sympy

`Sympy` est une bibliothèque Python servant à faire du calcul formel. Elle possède un module de géométrie différentielle avec lequel on peut calculer des dérivées de Lie.

On importe donc tous les outils depuis ce module :

```
from sympy.diffgeom import *
```

Et on a alors accès aux différentes notions présentées auparavant :

- `M = Manifold('M', 3)` pour définir une variété de dimension 3,
- `P = Patch('P', M)` pour définir une sous-variété différentielle de `M`,
- `coord = CoordSystem('coord', P, ['x_1', 'x_2', 'x_3'])` pour définir un système de coordonnées associé à `P`,
- `dx = LieDerivative(f, x)` pour calculer la dérivée de Lie de `x` par rapport au champ de vecteurs `f`.

3 Exemple de suivi de champ de vecteurs

Attention : à partir de maintenant, on manipule deux champs de vecteurs complètement différents. Dans la section précédente était présenté le champ de vecteurs associé à la fonction d'évolution f, qui est de la dimension du vecteur d'état. Maintenant, on va chercher à contrôler un robot pour qu'il suive un champ de vecteurs, qui lui sera en 2D (éventuellement 3D), et sera indépendant du modèle d'état du robot.

3.1 Intérêt du suivi de champ de vecteurs

Suivre un champ de vecteurs plutôt qu'une trajectoire présente l'avantage de ne pas prendre en compte le temps. En effet, quelque soit la situation dans laquelle notre robot se trouve, s'il converge rapidement vers le champ de vecteurs puis qu'il le suit, on peut garantir sa trajectoire.

En revanche, si on impose un suivi de trajectoire, si le robot ralentit ou accélère à un moment donné, il risque de couper les virages ou de faire demi-tour.

D'autre part, le suivi d'un champ de vecteurs permet d'anticiper les virages et donc de prendre en compte la courbure d'une trajectoire. Ce suivi est donc plus précis en général, si on ne souhaite pas tenir compte de la vitesse.

3.2 Méthode

Pour suivre un champ de vecteurs, un robot doit avoir, à tout instant, son vecteur vitesse colinéaire au vecteur v, où v est le vecteur du champ de vecteurs au point où se situe le robot.

Autrement dit, notre erreur e va s'exprimer comme

$$e = \psi - \Theta, \tag{3}$$

où ψ représente l'angle du vecteur v, et Θ l'angle du vecteur vitesse du robot.

C'est à dire :

$$\begin{cases} \psi = & \arctan 2(b, a) \\ \Theta = & \arctan 2(\dot{y}, \dot{x}) \end{cases}, \tag{4}$$

Avec (a, b) le champ de vecteurs en fonction de (x, y) la position du robot. Suivant le modèle utilisé, on a souvent l'égalite $\Theta = \theta$, où θ est le cap du robot.

Puis on peut utiliser la méthode de linéarisation par bouclage vue auparavant, avec notre nouvelle erreur. On pourra donc garantir qu'en quelques secondes, le robot aura son vecteur vitesse colinéaire au champ de vecteur, et si ce dernier à bien été défini, on suivra précisément le flot en fonction des conditions initiales.

4 Travail à faire

Je vous conseille d'utiliser *git* pour travailler. Une fois les deux questions de la section 4.1 terminées, faites un *tag* de la version de votre fichier *line_following_sympy.py*, ou gardez-en une copie que vous ne toucherez plus.

Puis, pour les questions suivantes, repartez de ce fichier et faites des tags (ou des copies) dès que vous avez terminé une question.

À la fin, envoyez-moi votre archive git ou une archive avec tout votre travail à joris.tillet@ensta-bretagne.org.

4.1 Voiture de Dubins suivant une ligne

Le but est de simuler une voiture de Dubins et de trouver la bonne commande pour qu'elle suive la ligne $y = 0$ grâce à un champ de vecteurs. On pourra prendre par exemple :

$$\begin{cases} a = & 1 \\ b = & -y \end{cases}.$$

1) Vous devez faire l'exercice "à la main", sans utiliser `Sympy`. Vous pouvez partir du fichier *line_following.py* pour vous aider.

2) Utilisez la bibliothèque `Sympy` pour faire la même chose avec du calcul symbolique, en vous basant sur le fichier *line_following_sympy.py*.

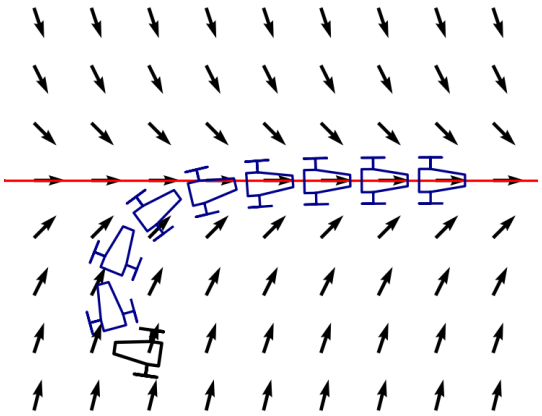


FIGURE 2 – Objectif de la simulation

4.2 Avec d'autres champs de vecteurs

On simule à nouveau la voiture de Dubins, mais on veut cette fois-ci que le robot suive des champs de vecteurs différents, donnés ci-dessous. Utilisez uniquement la bibliothèque `Sympy` pour faire ces exercices (si on essaye de les faire sans, on se rendra rapidement compte de l'intérêt du calcul formel).

L'objectif est de suivre :

1) une sinusoïde, avec

$$\begin{cases} a = & 1 \\ b = & \cos(x) - y + \sin(x) \end{cases},$$

2) le cycle limite de Van der Pol, en dessinant la trace du robot pour vérifier qu'il suit bien le cycle :

$$\begin{cases} a = & y \\ b = & -(0.01x^2 - 1)y - x \end{cases}.$$

4.3 En changeant le modèle de robot

On désire maintenant non plus contrôler la voiture de Dubins, mais la remorque qu'elle tracte. Vous pouvez faire cet exercice sur la sinusoïde ou bien sur le cycle limite de Van der Pol.

On pourra modéliser la remorque en rajoutant l'angle θ_t à notre vecteur d'état, avec θ_t le cap de la remorque. On a donc $\theta - \theta_t$ qui représente l'angle entre la voiture et la remorque. On pourra prendre : $\dot{\theta}_t = \frac{1}{L_t} \sin(\theta - \theta_t)$, avec L_t la longueur entre la voiture et la remorque.