doc kraken

Clément Besnier

May 2018

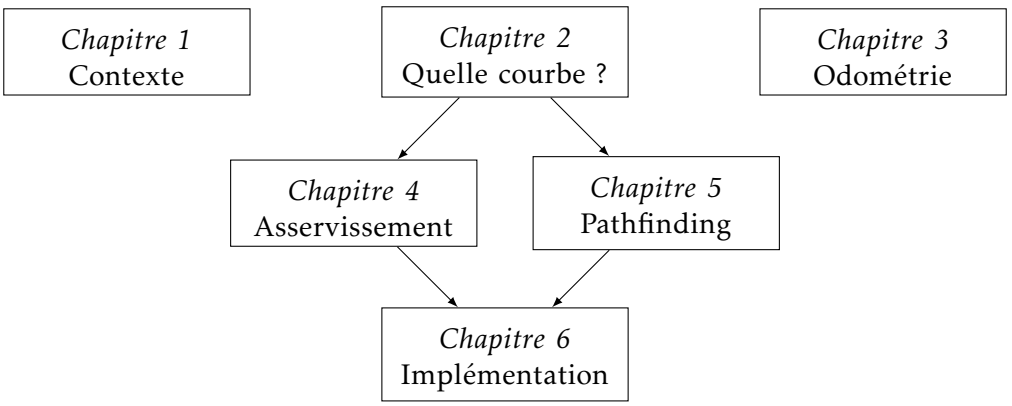
Contents

1 Introduction

In this doc, we only considere a robot which is:

- non-holonomic, which cinematically behaves like a car
- with
- with proximity sensors
- autonomous and does not need to cooperate

If you want to read only one chapter, beware the dependencies between sections



2 Mathematical background

2.1 Path

The easiest way to move with a robot is to make rotations and translations.

2.2 Curved Path

Definition:

Criteria:

- ExpressivityLeur expressivité, c'est-à-dire à quel point les courbes du modèle pourront se rapprocher de n'importe quelle courbe;
- Constraints on the robot:
- Math:

2.3 Circular arc

2.4 Bezier Curve

https://en.wikipedia.org/wiki/B%C3%A9zier_curve

2.5 Clothoids

Also called Euler spiral https://en.wikipedia.org/wiki/Euler_spiral, clothoids have wonderful properties so that we use them for all our motorways and railways, where straight lines join [Baa82], for fonts and design.

Let's assume we are driving a car with a constant speed. Then we are turning the wheel. That's all, the trajectory of the car is describing a clothoid! What is great with clothoids is that the derivative of the curvature of clothoids is a constant and this is good for cars.

Remarque : From now, $'$ means the deriation according to curvilinear abscissa s.

If $\theta(s)$ is the orientation of the robot at s, the curvature at s is :

$$\kappa(s) = \theta'(s)$$

The greater the absolute value of the curvature, the faster we turn.

En gros, plus la valeur absolue de la courbure est grande, plus on tourne vite. De plus, si on approche la trajectoire du robot en s par un cercle (appelé cercle osculateur), alors le rayon $R(s)$ de ce cercle est appelé rayon de courbure[1] et vaut[2] :

$$R(s) = \frac{1}{\kappa(s)}$$

[1] plus précisément, c'est le rayon algébrique, qui peut être négatif

[2] notez, cette définition n'est pas utilisée par la suite. C'est juste que savoir que la courbure est l'inverse du rayon du courbure, dans la vie, c'est important

Its parametric equation is:

$$p(t) = a \int_0^t e^{iu^2}\, du \qquad (*)$$

A clothoid is thus a curve such that $\kappa(s)' = C$.

A unitary clothoid ($C = 1$) may be drawn thanks to its power series:

$$\begin{cases} x &= s - \frac{s^5}{1\cdot 2\cdot 5} + \frac{s^9}{1\cdot 2\cdot 3\cdot 4\cdot 9} - \ldots = \sum_{n=0}^{+\infty} \frac{(-1)^n \cdot s^{4n+1}}{(2n)!\cdot(4n+1)} \\ y &= \frac{s^3}{1\cdot 3} - \frac{s^7}{1\cdot 2\cdot 3\cdot 7} + \frac{s^{11}}{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 11} - \ldots = \sum_{n=0}^{+\infty} \frac{(-1)^n \cdot s^{4n+3}}{(2n+1)!\cdot(4n+3)} \end{cases} \qquad (**)$$

Ces calculs de série peuvent être effectués une fois pour toutes, les clothoïdes étant toutes similaires entre elle (sauf dans le cas particulier de la ligne droite et des cercles, c'est-à-dire quand $\kappa' = 0$)

Par contre, il y a certains calculs qui devront être approchés, notamment la distance d'un point à la courbe[3]. The distance between a point and the curve.

De plus, si on fait intervenir la vitesse angulaire $\omega(s)$ et la vitesse instantanée $v(s)$ du robot, on a la relation suivante :

$$\omega(s) = v(s) \times \theta'(s) = v(s) \times \kappa(s)$$

Mais on n'est pas obligé de suivre une clothoïde à vitesse constante. Il suffit d'avoir :

$$\kappa(s)' = \left(\frac{\omega(s)}{v(s)}\right)' = C$$

soit[4]

$$\frac{\omega'(s) \times v(s) - v'(s) \times \omega(s)}{v(s)^2} = C$$

Au final, les clothoïdes sont bien pratiques, mécaniquement. Nous avons donc choisi une succession d'arcs de clothoïdes comme trajectoire courbe pour ces raisons. Mathématiquement, c'est globalement facile (on verra ça en détail dans un chapitre ultérieur).

3

https://en.wikipedia.org/wiki/Rotary_encoder

On appelle *codeur* un capteur rotatif qui permet de connaître les déplacements du robot. Il y en a deux, un pour chaque roue de propulsion. Ils sont soit intégrés aux moteurs, soit montés sur des roues folles. Ce deuxième cas, un peu plus lourd à mettre en œuvre mécaniquement, à l'avantage de détecter les patinages (quand les roues tournent mais glissent, ce qui fait que le robot ne bouge pas).

A partir des informations provenant de ces codeurs, on peut estimer la position du robot.

Il y a plusieurs moyens d'approcher la position du robot à partir du mouvement des codeurs en utilisant des approximations. On peut voir deux principales approximations.

3.1 Linear Approximation

On suppose que le robot suit des lignes brisées. C'est un calcul simple à mettre en place qui suffit le plus souvent… sauf en cas de trajectoire courbe.

3.2 Circular Approximation

Une approximation plus complexe (dont le cas linéaire est un cas particulier) mais qui est bien plus adaptée aux trajectoires courbes. En plus, c'est une approximation qui fait naturellement intervenir le calcul de la courbure que suit le robot ; et sans trop vouloir *spoiler*, ça nous sera utile.

Le lecteur intéressé est invité à lire le document de l'équipe RCVA sur le sujet[RCV10][5]. Il est complet et je n'ai rien à y ajouter.

[3] on en reparlera après

[4] même si en pratique, rassurez-vous, on n'utilisera pas cette équation

[5] `http://www.rcva.fr/images/stories/site/cours/Odometrie2010/ODOMETRIE_2010.pdf`

4 Pathfinding

Here is presented the core of this docs: finding the shortest path between a start point to a goal point. To begin with, let's introduce an abstract definition of our aim: Let a weighted graph G = (E,V), with E the set of edges and V the set of vertices (or nodes), and a function δ, such that $\delta(e) \in \mathfrak{Re}_+$. V represents the possible positions of the robot, E the possible paths between positions and δ the cost of an edge (because moving costs time!). The start node is $n_S \in E$ and the goal node $n_G \in E$. Now that we have everything to discuss how to find a p

4.1 Dijkstra's algorithm

All nodes, except n_S, are assumed to be distant from n_S of +inf.

4.2 A*

A* is an **informed search** algorithm. It means that it knows the complete graph

The node evaluation is done as following: f(n) = g(n) + h(n), where f(n) is the estimation of the total cost, g(n) is th cost from n_S to n and h(n) is the cost from n to n_G. the node with the lowest cost is choen.

There is a fundamental constraint that h must obey: it must be consistent, i.e. $\forall (x,y) \in E^2, h(x) \geq d(x,y) + h(y)$.

4.3 D*-lite

[KL02]

https://en.wikipedia.org/wiki/D*

4.4 What is really done by Kraken

5 Control theory

https://pdfs.semanticscholar.org/f22f/52367026da0be2e6291a10c8eb40a5617d4d.pdf

https://arxiv.org/pdf/1610.06534.pdf

Où on se rend compte qu'on n'est plus au Kansas

6 Suivi de chemin de C. Samson

6.1 L'équation

Le régulateur PID est un système d'asservissement qui s'appuie sur trois grandeurs : l'erreur, la dérivée de l'erreur et son intégrale. Il est très utilisé dans l'industrie et la robotique n'y fait pas exception.

Le problème avec le PID, c'est qu'il faut être asservi à une certaine valeur. Or, nous, on veut s'asservir à une *trajectoire*, qui est constituée d'une infinité de positions

Introduisons quelques notations:

- R la position du robot
- R' le point de la courbe le plus proche de R
- θ l'orientation du robot
- θ_{des} cette orientation en R'
- $\kappa(R')$ la courbure de $\mathcal{C}$ en R'

- $d = d(R, R')$ l'erreur de position, algébrique (positive si le robot est à gauche de la courbe, négative sinon[6])
- $\theta_e = \theta - \theta_{des}$ l'erreur d'orientation, algébrique aussi
- κ_c la consigne de courbure à donner au robot

Alors on a[7]:

$$\kappa_c = \kappa(R') - k_1 d - k_2 \theta_e \qquad (\bigstar)$$

où $k_1 = \xi^2$ et $k_2 = 2\zeta\xi$ avec ξ la pulsation propre du système et ζ son amortissement.

The complete formula is available at[MS93, Sam95] ; la formule ci-dessus est une version approchée au premier ordre issue de [Gau99].

Ainsi, on dispose à tout instant de la consigne en courbure à fournir au robot pour qu'il suive la trajectoire. Il suffit alors de fixer une vitesse v (celle qu'on veut) pour en déduire la consigne en vitesse angulaire $\omega_c = v \times \kappa_c$.

A noter que d'autres régulateurs existent, comme[KC99]. Celui de Samson est adapté à la clothoïde car on peut très facilement connaître la courbure $\kappa(R')$ et l'orientation $\theta_{des}(R')$ en un point de la trajectoire. Seul le paramètre d est un peu compliqué à calculer.

6.2 Mais tout n'est pas si simple

Et concrètement, comment fait-on ? Il y a pas mal de paramètres qui entrent en jeu. Tout d'abord, pour déterminer la consigne du robot, il faut deux paramètres : κ_c (la courbure consigne) et v_c (la vitesse consigne). Grâce au régulateur de Samson, on a κ_c. J'ai dit plus haut que v_c est libre. Mais dans les faits, la vitesse est limitée par la courbure : si la courbure est grande (on tourne beaucoup), il ne faudra pas aller trop vite sinon la force centrifuge sera trop grande et le robot pourra soit lever le côté intérieur soit déraper vers l'extérieur du virage (selon la mécanique du robot).

Ainsi, la vitesse dépend de la courbure. Il y a alors deux possibilités : soit on se met à une vitesse qui ne pose pas ce genre de problème (en supposant que la courbure ne sera jamais trop grande, quitte à la caper), soit on ajuste la vitesse. Dans la suite de ce chapitre, on va supposer le second cas (le premier ne posant pas problème).

Pour la vitesse à appliquer au robot, il y a deux possibilités :

- soit la vitesse actuelle est toujours compatible avec κ_c, auquel cas on ne change rien ;
- soit il faut ralentir.

S'il faut ralentir, on se retrouve dans un scénario un peu compliqué : et si la consigne en courbure augmente si vite que la consigne en vitesse ne peut pas suivre, du fait de la décélération limitée[8] ?

7 Algorithme d'asservissement

On suppose disposer d'un asservissement en vitesse à chaque roue (je vous laisse adapter si ce n'est pas le cas) qui utilise les consignes v_{gc} (consigne en vitesse de la roue gauche) et v_{dc} (consigne en vitesse de la roue droite). D'ailleurs, ces asservissements sont liés, car les contraintes mécaniques des moteurs font qu'il y a trois bornes sur acc_g (accélération de la roue gauche) et acc_d (accélération de la roue droite): $|acc_g| < C_1$, $|acc_d| < C_1$, $|acc_g + acc_d| < C_2$ et $|acc_g - acc_d| < C_3$. Contrairement aux déplacements en ligne droite et aux rotations sans translation où $|v_{gc}| \approx |v_{dc}|$), $|v_{gc}|$ et $|v_{dc}|$ peuvent être différentes dans le cas des trajectoires courbes. Je ferme la parenthèse.

On suppose enfin aller en marche avant. Toutes les vitesses sont donc supposées positives. L'algorithme s'adapte bien au cas de la marche arrière.

Définissons alors quelques nouvelles notations :

[6] par rapport à la direction dans laquelle le robot avance. La formule change légèrement si le robot suit une trajectoire courbe en marche arrière

[7] s'il y a une seule formule à retenir, c'est celle-là

[8] en supposant, bien sûr, qu'on puisse arriver dans un tel cas limite.

- v_r la vitesse réelle (mesurée) du robot;
- v_{max} la vitesse maximale du robot(il pourra aller moins vite s'il y est forcé);
- $v_{lim}(\kappa)$ la vitesse limite (maximale) en fonction de la courbure;
- $\kappa_{lim}(v)$ la courbure limite (maximale) en fonction de la vitesse[9];
- κ_{max} la courbure maximale que le robot peut suivre;
- κ_s la consigne en courbure provenant de (★);
- κ_c la consigne effective qu'on va suivre;
- v_c la consigne en vitesse linéaire;
- ω_c la consigne en vitesse angulaire;
- C la moitié de la distance entre les deux roues de propulsion.

La relation entre courbure et vitesse est:

$$v_{lim}(\kappa) = \min\left(\sqrt{\frac{gd}{\kappa z}}, \sqrt{\frac{k_a g}{\kappa}}\right)$$

où d est la demi-distance entre les faces extérieures des roues de propulsion, z l'altitude du centre de gravité du robot, k_a le coefficient d'adhérence des roues de propulsion sur la table. Il y a deux facteurs : le premier correspond au point de décollement et le second au point de patinage. Selon le robot, ce sera l'un ou l'autre qui sera limitant.

Notre algorithme d'asservissement est le suivant[10] :

Algorithm 1: Asservissement à la courbe $\mathcal{C}$

Input: v_{max} la vitesse maximale, $\mathcal{C}$ la courbe à suivre

Loop

- $R' \leftarrow Proj_{\mathcal{C}}(R)$ // on projette R sur $\mathcal{C}$
- $\kappa_s \leftarrow EquSamson(\kappa(R'), d(R,R'), \theta_{robot} - \theta(R'))$ // calcul de κ_s avec (★)
- $\kappa_c \leftarrow \min(\kappa_{lim}(v_r), \kappa_s, \kappa_{max})$ // on s'approche au mieux de la consigne idéale
- $v_c \leftarrow \min(v_{lim}(\kappa_s), v_{max})$ // on limite la consigne en vitesse si besoin est
- $\omega_c \leftarrow \kappa_c \times v_r$ // la consigne angulaire
- $v_{gc} \leftarrow v_c - C \times \omega_c$ // consigne en vitesse moteur gauche
- $v_{dc} \leftarrow v_c + C \times \omega_c$ // onsigne en vitesse moteur droit
- $AsserVitesse(v_{gc}, v_{dc})$ // PID classique qui commande les moteurs

Grâce aux propriétés de la clothoïde, l'évolution de toutes ces grandeurs est continue. De plus, on sait que la dérivée de $\kappa(R')$ est bornée, et donc que celle de κ_c le sera probablement aussi.

L'algorithme ci-dessus permet d'être asservi à une trajectoire mais ne permet pas de commencer ou de finir la trajectoire. En effet, il y a bien un moment où il faudra s'arrêter...[11] Une manière d'asservir un robot est d'utiliser un trapèze de vitesse. Le mouvement est alors décomposé en trois parties : accélération constante, vitesse constante et enfin décélération constante.

On va supposer disposer d'un régulateur PID sur la distance entre le robot et sa destination qui nous fournira une consigne v_k pour la vitesse linéaire. Il suffira alors d'utiliser l'algorithme ci-dessus en calculant à chaque itération $v_{max} = \min(v_k, v_t)$, où v_t est la vitesse maximale que le robot ne pourra jamais dépasser, celle du sommet du trapèze.

[9] si $v = v_{lim}(\kappa)$, alors $\kappa = \kappa_{lim}(v)$ (et inversement). Pour le dire plus mathématiquement, v_{lim} est la bijection réciproque de κ_{lim}. De plus, ces deux fonctions sont décroissantes (plus on tourne, moins on doit aller vite ; plus on va vite, moins on peut tourner)

[10] Protip : tous les meilleurs algos contiennent une boucle infinie

[11] je parle d'arrêts *conventionnels*. Se prendre un bord de table n'est pas une solution viable pour s'arrêter.

8 Planification

9 Conclusion

Où y a plus grand chose à dire...

Au final, on a un robot asservi correctement une trajectoire courbe parce qu'elle est mécaniquement adaptée, qui peut calculer son chemin et surtout réagir rapidement à la moindre variation de son environnement.

Il y a encore des points à améliorer, et ce document sera modifié. N'hésitez pas à me contacter si vous avez des questions ou des idées d'amélioration. Si ce sujet vous intéresse et que vous souhaitez en savoir plus, je vous encourage fortement à consulter la bibliographie à la page suivante.

Et n'oubliez pas...

Les docs de TechTheTroll, c'est comme les AX-12. C'est fait pour tourner !

PF

References

[Baa82] KG Baass. Use of clothoid templates in highway design. Technical report, 1982.

[Gau99] Eric Gauthier. *Utilisation des réseaux de neurones artificiels pour la commande d'un véhicule autonome*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 1999.

[KC99] Kyoung Chul Koh and Hyung Suck Cho. A smooth path tracking algorithm for wheeled mobile robots with dynamic constraints. *Journal of Intelligent and Robotic Systems*, 1999.

[KL02] Sven Koenig and Maxim Likhachev. D^* lite. *AAAI/IAAI*, 28, 2002.

[MS93] Alain Micaelli and Claude Samson. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. Technical report, 1993.

[RCV10] RCVA. Odométrie. 2010.

[Sam95] Claude Samson. Control of chained systems application to path following and time-varying point-stabilization of mobile robots. *Automatic Control, IEEE Transactions on*, 1995.