Compte rendu : Trajectoires

Tom Déporte, Ulrich Camille

Novembre 2021

Dans ce compte-rendu nous respecterons la notation suivante:

- n → nombre de Noeuds [t,x] dans un spline
- x → position
- y → vitesse
- t → temps
- S_i → Spline au point de passage i
- $S_i(t)$ → Position à t résultant du polynome S_i

1 Trajectoires en dimension 1

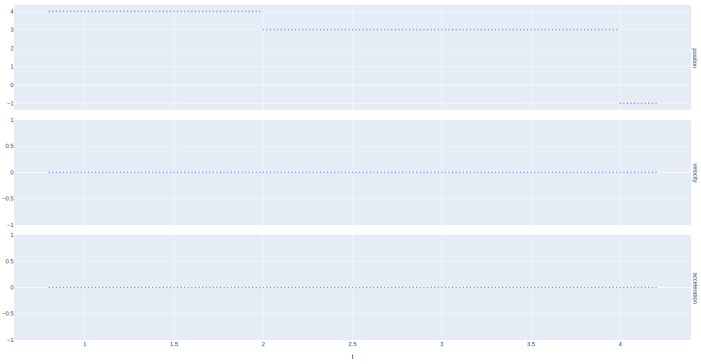
1.1 Définition de la classe *Spline*

1.2 Spline constant

La dernière valeur spécifiée est utilisée jusqu'au prochain point de passage:
$S_i(t) = x_i$
Cette fonction n'a pas ou très peu d'application car la position doit en générale toujours être continue pour un robot

Ci-dessous un résultat de test sur la spline suivante:

- Start → 0.0
- Noeuds [t,x] → [[0,2],[1,4],[2,3],[4,-1]]



1.3 Spline linéaire

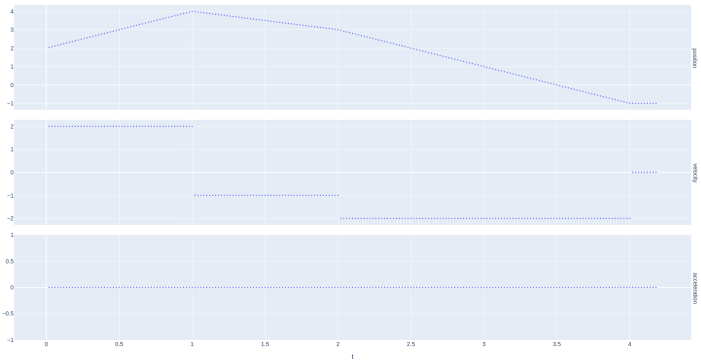
L'implémentation d'un sline linéaire est déja plus réalisable pour un robot, cependant a vitesse n'étant pas continue, on peut avoir des pics d'accélaration qui tendent vers l'infini ce qui n'est pas possible matériellement.

On résout pour chaque i le système suivant:

$S_i(t_i) = x_i$
$S_i(t_{i+1}) = x_i + 1$

Ci-dessous un résultat de test sur la spline suivante:

- Start $\rightarrow$ 0.0
- Noeuds [t,x] $\rightarrow$ [[0,2],[1,4],[2,3],[4,-1]]



1.4 Spline cubique

1.4.1 Spline cubique locales

1.4.1.1 Splines cubiques avec dérivées nulles

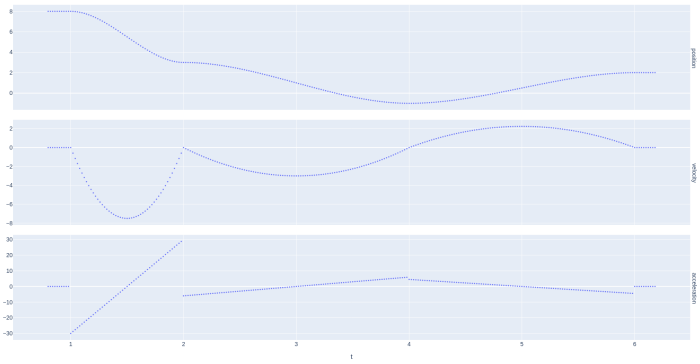
Le spline cubique avec dérivées nulles est une première approche qui prend en compte les dérivées première et seconde à l'arrivée, ce qui nous d'avoir une courbe plus lissé.

On résout pour chaque i le système suivant:

$S_i(t_i) = x_i$
$S_i(t_{i+1}) = x_i + 1$
$\dot{S}_i(t_i) = 0$
$\dot{S}_i(t_{i+1}) = 0$

Ci-dessous un résultat de test sur la spline suivante:

- Start $\rightarrow$ 0.0
- Noeuds [t,x] $\rightarrow$ [[0,2],[1,8],[2,3],[4,-1],[6,2]]



1.4.1.2 Splines cubiques à large voisinage

Le spline cubique à large voisinage prend pour contraintes les positions au temps précédent, présent, suivant et suivant du suivant.

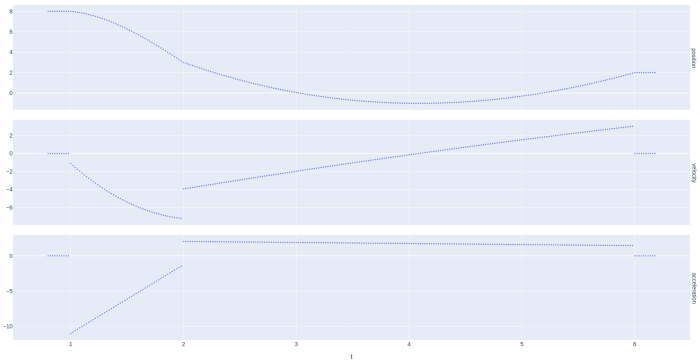
On résout pour chaque i le système suivant:

$S_i(t_{i-1}) = x_{i-1}$
$S_i(t_i) = x_i$
$S_i(t_{i+1}) = x_{i+1}$
$S_i(t_{i-2}) = x_{i-2}$

Ci-dessous un résultat de test sur la spline suivante:

- Start $\rightarrow$ 0.0

- Noeuds [t,x] → [[0,2],[1,8],[2,3],[4,-1],[6,2]]



1.4.1.3 Splines cubiques custom

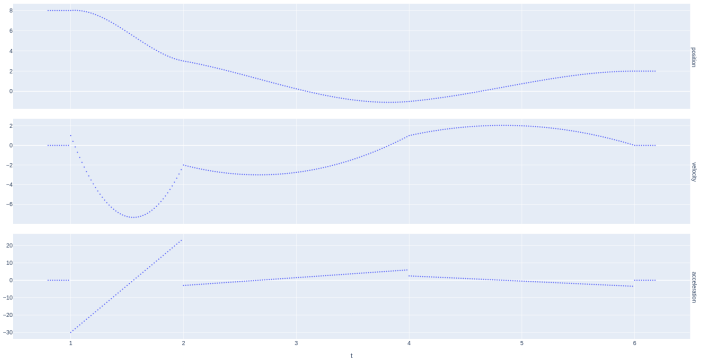
Pour ce spline on prend pour contraintes les positions et la vélocité (que l'on notera y) au temps précédent et présent.

On résout pour chaque i le système suivant:

$S_i(t_i) = x_i$
$S_i(t_{i+1}) = x_{i+1}$
$\dot{S}_i(t_i) = y_i$
$\dot{S}_i(t_{i+1}) = yi + 1$

Ci-dessous un résultat de test sur la spline suivante:

- Start → 0.0
- Noeuds [t,x,y] → [[0,2,0],[1,8,1],[2,3,-2],[4,-1,1],[6,2,0]]



1.4.2 Spline cubique globales

Pour chaque Splines cubiques globale on résout un seul système composé de 4n - 2 contraintes:

2n contraintes de position:

Pour $i \in [0, n_{-1}]$, $S_i(t_i) = x_i$
Pour $i \in [0, n_{-1}]$, $S_i(t_{i+1}) = x_{i+1}$

2n - 2 contraintes de vitesse:

Pour $i \in [0, n_{-1}]$, $S_i(t_i) = x_i$
Pour $i \in [0, n_{-1}]$, $S_i(t_{i+1}) = x_{i+1}$

Il est possible d'ajouter deux contraintes pour résoudre des problèmes particuliers.

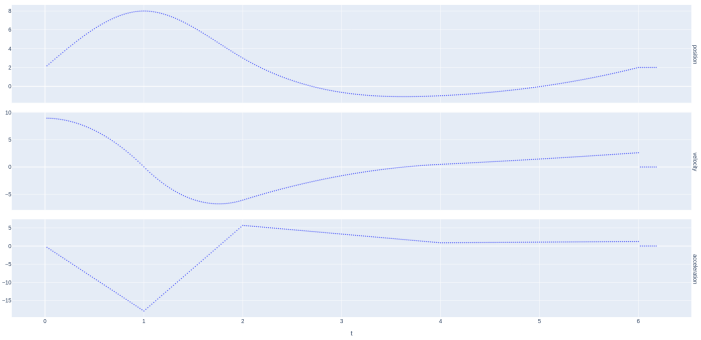
1.4.2.1 Splines cubiques naturelles

Pour une spline cubique naturelle, on ajoute les contraintes suivantes :

$\dot{S}_0(t_0) = 0$
$\ddot{S}_{n-1}(t_n) = 0$

Ci-dessous un résultat de test sur la spline suivante:

- Start $\rightarrow$ 0.0
- Noeuds [t,x] $\rightarrow$ [[0,2],[1,8],[2,3],[4,-1],[6,2]]



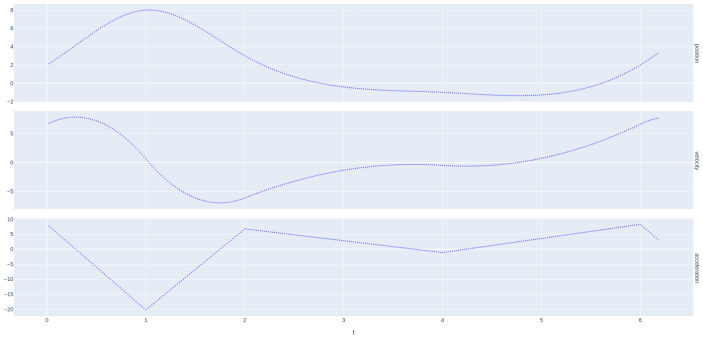
1.4.2.2 Splines cubiques périodiques

Pour une spline cubique périodique, on ajoute les contraintes suivantes afin que le mouvement ne s'arrête pas entre chaque répétition:

$\dot{S}_0(t_0) = \dot{S}_{n-1}(t_n)$
$\ddot{S}_0(t_0) = \ddot{S}_{n-1}(t_n)$

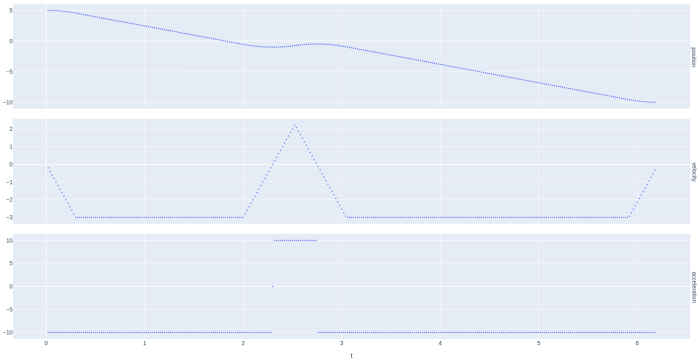
Ci-dessous un résultat de test sur la spline suivante:

- Start → 0.0
- Noeuds [t,x] → [[0,2],[1,8],[2,3],[4,-1],[6,2]]



1.5 Loi trapézoïdale en vitesse

La loi de trapèze semble correct car nous avons bien un trapèze en vitesse entre les deux premiers point donc avec un temps caratéristique alpha qui fait monter progressivement la valeur de la vitesse jusqu'à son maxima (ici vMax = 3). En ce qui concerne le point suivant. la position suit une force cubique par intégration du trapèze et l'accélération est une constante.



Sur les figures ci-dessous on peut voir deux extraction de la vitesse sur deux intervalle différent, dans la première figure on obtient bien un trapèze car D est assez grand comparé à la condition.
Cependant, entre le deuxième et troisième noeud donc l'échantillon de vitesse de la deuxième figure, la distance n'est pas assez long pour aller jusqu'à la vitesse maximale c'est pourquoi on obtient ce triangle.

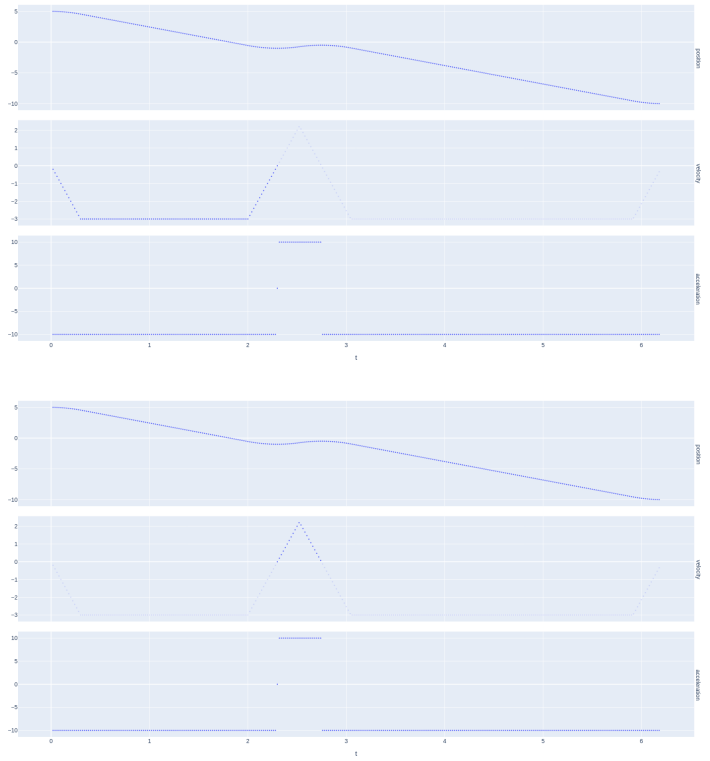


Figure 1: Découpage vitesse selon deux intervalles différents

2 Trajectoires multidimensionnelles

Lors de cette partie, nous avons implémenté les méthodes et constructeur RobotTrajectory qui permettra de créer une commande multidimensionnelle et applicable à un robot.

À partir des trajectoires/splines codé auparavant, on peut donner des trajectoires selon tout les axes au robot.

Avant on donnait seulement la cibles final au robot, maintenant la cible donnée change au cours du temps afin de converger vers la solution proposer sur chaque dimension par le spline/trajectoire.

Un des problèmes rencontrer est tout d'abord faire la différence entre cibles dans le monde articulaire ou cibles dans le monde opérationnel. En effet, prenons l'exemple de la consigne TrapezoidaleVelocity. On peut soit vouloir que chaque axe avance avec une vitesse trapézoïdale et donc des actionneurs qui vont devoir "suivre" de force, on a donc ici une cible opérationnelle. Inversement, une cible dans l'espace articulaire donnera des moteurs qui bouge avec une vitesse trapézoïdale (que celui-ci soit en rotation ou translation) et donc les autres axes n'auront pas de chemin ou manière particulière d'atteindre la cible mais au bout du compte elle l'atteindra.

C'est pourquoi nous allons réutiliser les outils créer dans les TP précédents, tel que la Jacobienne, la MGD, La MGI,...

Position :

Dans le cas d'une cible à l'espace opérationnelle :

- Le getValue donnera directement la position dans l'espace opérationnelle,
- la MGI permettra de passer de l'espace opérationnelle à l'espace articulaire.

Dans le cas d'une cible à l'espace articulaire:

- Le getValue donnera directement la position dans l'espace articulaire,
- la MGD permettra de passer de l'espace articulaire à l'espace opérationnelle .

Vitese : Ici, il sera nécéssaire d'introduire l'équation ci-dessous

$$Jq' = o' \tag{1}$$

- Le getValue donnera directement la vitesse dans l'espace opérationnelle,

- l'équation permettra de passer de l'espace opérationnelle à l'espace articulaire.

Dans le cas d'une cible à l'espace articulaire:

- Le getValue donnera directement la vitesse dans l'espace articulaire,
- l'équation permettra de passer de l'espace articulaire à l'espace opérationnelle .

Accélération : Et enfin, l'accélération se servira aussi de l'équation (1) à une dérivé près ce qui donne l'équation suivante

$$J'q' + Jq'' = o'' \tag{2}$$

- Le getValue donnera directement l'accélération dans l'espace opérationnelle,
- 'équation permettra de passer de l'espace opérationnelle à l'espace articulaire.

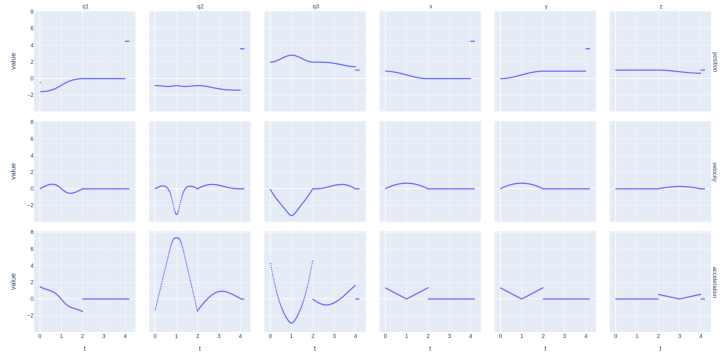
Dans le cas d'une cible à l'espace articulaire:

- Le getValue donnera directement l'accélération dans l'espace articulaire,
- l'équation permettra de passer de l'espace articulaire à l'espace opérationnelle .

le calcul de la dérivé Jacobienne n'est pas anodine. la calcul de la Jacobienne et sa dérivée a été rajouter dans robot.py sur chaque robot mais aussi des nouvelles fonctions dans homogeneous_transform.py ont été rajouté nottament : dd_rot_x,y,z.
dd_rot_x,y,z retourne une matrice de transformation avec une rotation dérivé deux fois

```
def dd_rot_x(alpha):
    """Return the 4x4 homogeneous transform corresponding to a rotation of
    alpha around x
    """
    c = np.cos(alpha)
    s = np.sin(alpha)
    return -np.array([[0, 0, 0, 0],
                      [0, c, -s, 0],
                      [0, s, c, 0],
                      [0, 0, 0, 1]], dtype=np.double)
```

Voici quelques logs pour rendre compte du comportement théorique et souhaité du robot avec les paramtres du fichier "rrr_zero_derivative_operational.json" : cible opérationelle :



cible articulaire

: