Gestion du calcul de l’erreur dans le cas d’une trajectoire de type B-Spline

Vecteur Robot ()

Vecteur normal ()

Vecteur error ()

Vecteur Vitesse ()

ThetaError = (Produit scalaire dans lequel

Cela permet d’éviter qu’on considère dans l’erreur à la fois l’erreur liée à l’orientation (décalage) et que cela introduise une accélération dans le robot pour tenter de rattraper cette erreur qui ne peut se rattraper que par changement de trajectoire

AlphaError = Angle( - \*

Il faut faire attention au fait que le rattrapage d’une erreur statique ne peut se faire qu’en roulant, car si on a une telle erreur à la fin, le robot risque de se tourner pour essayer de corriger l’erreur, et dans ce cas, on va avoir une orientation du robot complètement anormale par rapport

Ce qui donne tout simplement

AlphaError = Angle( -

Cas d’une erreur sans décalage

On voit dans ce cas, que le robot va accélérer car E et N sont quasiment colinéaires. Il ne faudrait pas que le robot soit tellement en retard que E et N soit orthogonal, mais dans ce cas c’est l’orientation qui prendrait le relai avant de « rattraper la colinéarité ».

Au niveau de l’orientation, on voit que le robot est mal orienté (différence entre R et N).

Mais comme il a aussi du retard, le produit extérieur de V et de E va également compter dans le rattrapage.

Stratégie pour gérer le problème de la continuité des trajectoires

Cas où doit accélérer

T3

T2

T1

On prend juste en compte le fait qu’on a déjà « gagné un peu de temps sur la phase d’accélération

Dans cet exemple, la 1ère phase consiste à « décélérer ! »

Gestion de l’arrêt brutal alors qu’on est en mouvement (gestion de la décélération à faire).

**Gestion en cas de trapeze**

time

speed

accTime

initialSpeed

decTime

endSpeed

On voit que l’intégrale de la vitesse (rose + bleu + vert) = nextPosition

Que se passe-t-il si lorsqu’on détermine que bleu (calculé par bleu = nextPosition – rose – vert) est négatif ?

On a

Dans ce cas, on souhaite avoir le type de courbe suivant

**Gestion en cas de triangle**

speed

time

speedMax

accTime

initialSpeed

decTime

endSpeed

L’aire de la partie bleue est égale à :

Donc

La distance *NextPosition* correspond à l’aire de l’ajout des triangles rose, vert bleu

Ce qui donne

2 \* bleu – 2 \* np (0)

Soit

(1)

Mais on a aussi la relation

D’où

En développant et en réduisant

En remplaçant (at + dt) par A

Puis

Puis

On obtient :

A \* smax \* smax + B \* smax + C = 0, qui renvoie

Or d’après (0), on voit que l’expression que B vaut 0, ce qui simplifie et nous permet de résoudre l’équation

**Test de la formule simplifiée :**

* at =dt, es = 0, is = 0 ;
* np / at

Comme s / at = acceleration, on retrouve :

BSplines de Test

Problème : Gérer au mieux le calcul d’une courbe de bézier