TP2 : Espace produit, pénalisation de courbure [1].

Ce TP est consacré aux méthodes de Fast-Marching dans des espaces de configuration de la forme $\Omega = U \times A$, où U est l'espace "physique" et A est un espace de paramètres abstrait.

Une application que nous mettons en avant est le calcul de chemins γ minimisant globalement une énergie faisant intervenir la courbure

$$\min_{T,\gamma} \int_0^T c(\gamma(t), \gamma'(t)) \mathcal{C}(\gamma''(t)) \mathrm{d}t, \qquad \gamma(0), \gamma'(0), \gamma(T), \gamma'(T) \text{ donnés}, \tag{1}$$

où le temps final T est libre, et où $\gamma : [0, T] \to U \subset \mathbb{R}^2$ est paramétré à vitesse euclidienne unité. Pour cela on pose $\Omega = U \times \mathbb{S}^1$ l'ensemble des positions et orientations dans le plan, et on utilise une métrique dégénérée de la forme suivante. Pour $(\mathbf{x}, \theta) \in U \times \mathbb{S}^1$ et $(\dot{\mathbf{x}}, \dot{\theta}) \in \mathbb{R}^2 \times \mathbb{R}$ un vecteur tangent

$$\mathcal{F}_{(\mathbf{x},\theta)}(\dot{\mathbf{x}}, \dot{\theta}) = c(\mathbf{x}, \theta) \mathcal{C}(\dot{\theta}) \qquad \text{si } \dot{\mathbf{x}} = (\cos\theta, \sin\theta), \tag{2}$$

étendue par 1-homogénéité si $\mathbf{x} = \lambda(\cos\theta, \sin\theta)$, $\lambda > 0$, et par $+\infty$ ailleurs.

1. Justifier, formellement, que le problème (1) peut se résoudre en calculant un chemin minimal pour la métrique (2).
2. On souhaite autoriser dans (1) la possibilité de passer en marche arrière. Comment modifier (2) en ce sens ?

Les données décrivant nos EDP seront stockées dans un dictionnaire, `hfmIn`, dont les clefs et valeurs seront ici notées sous la forme `'clef':valeur`. Toutes les listes et tableaux renseignés de cette manière doivent être au format `numpy.array`. On importera les librairies suivantes :

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.image
import HFMUtils as utils # Wrapper pour la librairie HFMpy + divers
```

⚠ *Un redémarrage du kernel du notebook est nécessaire à chaque changement de modèle.*

1 Planification de mouvement / Pénalisation de courbure

1.1 Dans un environnement vide

Création du domaine.

1. Initialiser un dictionnaire `hfmIn` décrivant le domaine physique, par exemple [2]

   ```
   hfmIn = utils.MakeHFMRect([-1,-1],[1,1],gridScale=0.02)
   ```

2. Préciser le nombre d'orientations utilisées pour la discrétisation du domaine angulaire $\mathbb{S}^1$, typiquement entre 60 et 120, et le format des tableaux

   ```
   'dims':np.append(hfmIn['dims'],60), 'arrayOrdering':'YXZ_RowMajor',
   ```

1. Dernière version : `dl.dropbox.com/s/5xkayaok4n0tgew/TP2.pdf`
2. Syntaxe : coin inférieur, coin supérieur, et en option l'échelle `gridScale` ou bien `dimx` le nombre de points horizontalement. Diminuer gridScale améliore la précision au prix d'un plus long calcul.

Choix du modèle. Trois modèles classiques sont disponibles, dûs à Reeds-Shepp, Euler-Mumford (courbes elasticae) et Dubins [3]. Ils correspondent respectivement à une croissance linéaire, quadratique, et infinie, du coût de déplacement en fonction de la courbure du chemin : $\mathcal{C}(\dot\theta) = \mathcal{C}_*(\xi\dot\theta)$ où $\xi > 0$ est homogène à un rayon de courbure, et où $\mathcal{C}_*(\dot\varphi)$ est donné par (resp.)

$$\sqrt{1+\dot\varphi^2}, \qquad 1+\dot\varphi^2, \qquad \begin{cases} 1 & \text{si } |\dot\varphi| \leq 1, \\ +\infty & \text{sinon.} \end{cases} \tag{3}$$

La dépendance $c(\mathbf{x}, \theta)$ aux position et orientation courante (2), est définie librement via la clef `'cost'`.

3. Définir complètement métrique, par exemple (pour un coût uniforme $c \equiv 1$ sur Ω)

   `'model':'Dubins2', 'xi':0.4, 'cost':1,`

4. Choisir un/des point source `'seed'`, et les extrémités des géodésiques `'tips'` dans $\Omega \subset \mathbb{R}^2 \times \mathbb{S}^1$. Lancer le calcul et afficher les résultats.

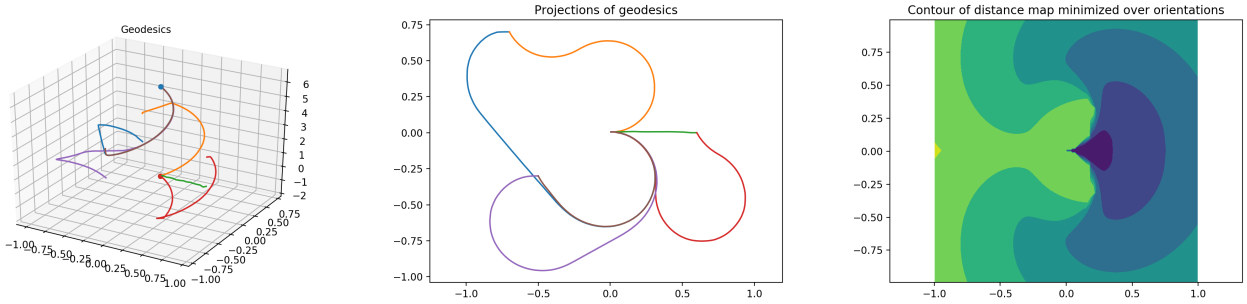

Figure : Quelques chemins minimaux pour le modèle de Dubins, leur projection dans le plan, et les contours de $u(\mathbf{x}) := \min\{U(\mathbf{x}, \theta); \theta \in \mathbb{S}^1\}$ où U est la fonction distance, solution d'une équation eikonale.

5. Quelle propriété qualitative ont les chemins minimaux de Dubins ? (Ci dessus.) La solution de l'équation eikonale est-elle continue ?
6. Dans le cas du modèle *Reeds-Shepp forward*, les orientations prescrites aux extrémités sont-elles satisfaites ? Que se passe t-il ? (Regarder les géodésiques en 3D ou utiliser un quiver plot de leurs tangentes.)
7. Essayer le modèle `'ReedsShepp'` qui dispose d'une marche arrière. Les chemins minimaux, et leurs projections planaires, sont-ils réguliers ? Quel est l'effet de l'option `'projective':1` ? (Elle remplace l'espace angulaire $\mathbb{S}^1$ par $\mathbb{P}^1$.)

Script complet : `dl.dropbox.com/s/kzpegdjapy659y6/TP2_MotionPlanning_Empty.py`

Modification du coût. Créer un système de paramètres `X,Y,T = utils.GetGrid(hfmIn)`

8. (Région privilégiée) Essayer un coût non-uniforme en espace, comme `'cost':1.+(Y>=0)`.

Pour conclure cette partie, nous proposons de modéliser un voilier essayant de remonter au vent.

9. (Face au vent) Choisir un domaine allongé, par ex $[0,5] \times [0,1]$, et placer `'seeds'` et `'tips'` à des extrémités opposées.
10. On utilisera le modèle `'ReedsSheppForward2'` et la vitesse [4] angulaire $s(\theta) = 2 - \cos(2\theta) -$

3. Définir la clef 'model' comme 'ReedsSheppForward2', 'Elastica2', ou 'Dubins2'
4. Vitesse et fonction coût sont inversement proportionnels.

$0.4\cos(3\theta)$. L'afficher `plt.plot(s(T)*np.cos(T),s(T)*np.sin(T))`. D'où vient le vent ?

11. Quelle est l'allure des chemins minimaux face au vent, et dos au vent ? Expliquer.

Script complet : `dl.dropbox.com/s/76m9u06bigi7jg9/TP2_MotionPlanning_Wind.py`

1.2 Visite au musée

Construction du domaine

12. Importer ls plan [5], de sorte que les murs soient représentés par des 1 et l'espace libre par 0.

```
walls = 1.-matplotlib.image.imread("PompidouCrop.png")
```

13. Créer un dictionnaire `hfmIn` et renseigner sa taille et le format des tableaux

```
'dims':np.array([walls.shape[1],walls.shape[0],60]), 'arrayOrdering':'YXZ_RowMajor',
```

Indiquer la présence de murs `'walls':walls`, et choisir l'échelle.

14. Afficher le plan, avec l'option `%matplotlib notebook`, pour identifier les coordonnées de la sortie et d'un certain nombre de points d'intérêt [6].

Choix du modèle

15. Choisir le modèle comme en (3), et le rayon de courbure ξ en tenant compte de l'échelle et de la taille de l'image (par ex. `'xi':1` pour `'gridScale':0.1`).
16. Choisir la source du `'seeds'` et les points de sortie `'tips'`. Considérer éventuellement des points *non-orientés*, définis dans les champs `'seeds_Unoriented'` et `'tips_Unoriented'`. Géodésiques correspondant à ces derniers : `utils.GetGeodesics(hfmOut,'Unoriented')`.

Script complet : `dl.dropbox.com/s/n5zb4j3908gd8co/TP2_MotionPlanning_Walls.py`

1.3 Véhicule déséquilibré

La définition du coût de la courbure se fait en termes de $\mathcal{C}_*$, à choisir parmi (3), et d'une variable intermédiaire $\dot{\varphi}$ jusqu'ici proportionnelle à la vitesse angulaire. Il est possible d'utiliser une expression plus générale :

$$\dot{\varphi} = \xi(\mathbf{x},\theta)(\dot{\theta} - \kappa(\mathbf{x},\theta))$$

Seul le modèle de Dubins est actuellement distribués dans cette variante [7] : `'model':'DubinsExt2'`.

17. (Embourbement local) Définir un véhicule ayant un rayon de braquage double dans une partie de l'image.
18. (Déséquilibrage) Définir un véhicule ayant un rayon de braquage double pour les virages droits plutôt que gauches. Un véhicule ne pouvant tourner que d'un côté.

Script complet : `dl.dropbox.com/s/h18pyn03ph1eqfk/TP2_MotionPlanning_Unbalanced.py`

5. Ici un plan partiel du centre Pompidou à Paris
6. Par ex. `[[1.7,6.1,1.]]` et `[[13,15],[19,14],[12,12],[16,5],[7,2],[3,13]]`, pour `'gridScale':0.1`.
7. Les autres modèles, Reeds-Shepp, Elastica, sont codés mais pas inclus dans votre package conda.