Laborator 3 (SPER)

Generarea de traiectorii în plan

28 februarie 2022

Cuprins

compilat la: 28/02/2022, 22:58

Scopul lucrării

Folosind modelul matematic al mașinii Dubins, construim traiectorii Dubins pentru atingerea unei configurații dorite.
Se prezintă de asemenea și noțiunea de reprezentare plată cu parametrizare prin funcții Bezier, pentru rezolvarea ulterioară a unei probleme de optimizare.

1 Noțiuni teoretice

Considerăm "mașina Dubins" ce nu poate aluneca lateral ("sideslip") și ce poate merge doar înainte [1]:

$$\begin{cases} \dot{x} &= u_V \cos\theta, \\ \dot{y} &= u_V \sin\theta, \\ \dot{\phi} &= \frac{u_V}{L} \tan u_\phi, \end{cases} \tag{1}$$

Mașina este condusă prin intrările $u_V \in [-1, 1]$ și $u_\theta \in (-\pi/2, \pi/2)$. Starea este dată de poziție (x și y) și orientare (θ).

Suntem interesați de trasarea unei traiectorii între două configurații arbitrare (configurație = combinație de poziție și orientare a vehiculului):

$$(x(t_i), y(t_i), \theta(t_i)) \qquad \longrightarrow \qquad (x(t_f), y(t_f), \theta(t_f)) \tag{2}$$

Se arată că pentru orice pereche de configurații sunt doar 6 posibilități optime de trasare a traiectoriei Dubins:

$$\{LRL, RLR, LSL, LSR, RSL, RSR\}, \tag{3}$$

unde, L – left, R – right și S – straight. Prin urmare, o traiectorie între două configurații succesive este unic definită de cuvântul ales din (3) și de lungimea fiecărei componente (unghi pentru L, R sau distanță pentru S).

Considerând $z = \begin{bmatrix} x & y \end{bmatrix}^\top$ ca ieșire plată pentru sistemul (1), rescriem ecuațiile (1) în funcție de aceasta:

$$\begin{cases} u_V &= \sqrt{\dot{z}_1^2 + \dot{z}_2^2} \\ u_\phi &= \arctan\left(\dfrac{L\dot{\phi}}{u_V}\right) = \arctan\left(L\dfrac{\ddot{z}_2\dot{z}_1 - \dot{z}_2\ddot{z}_1}{\left(\dot{z}_1^2 + \dot{z}_2^2\right)^{\frac{3}{2}}}\right) \\ \theta &= \arctan\left(\dfrac{\dot{z}_2}{\dot{z}_1}\right) \end{cases} \tag{4}$$

Prin urmare, putem reduce o problemă tipică de planificare a mișcării, ce presupune de exemplu:

- trecerea printr-o colecție de puncte intermediare: $z(t_j) = w_j$,
- minimizarea energiei traiectoriei: $\int_{t_i}^{t_f} \|\dot{z}(t)\|^2 dt$,

la o problemă de optimizare ce implică doar variabila $z(t)$.

Pentru o implementare practică, este necesar să exprimăm $z(t)$ ca o combinație de funcții bază, ponderate prin puncte de control:

$$z(t) = \sum_{i=0}^{n} P_i B_{i,n}(t). \tag{5}$$

Aceasta ne permite să reducem problema menționată anterior la una ce folosește ca variabile doar punctele de control $P_0 \dots P_n$ (în loc să încercăm obținerea în mod direct a lui $z(t) \leftarrow$ mult mai dificil și de obicei imposibil):

$$\min_{z(t)} \int_{t_i}^{t_f} \|\dot{z}(t)\|^2 dt \qquad \leftarrow \text{cost} \rightarrow \quad \min_{P_i} \int_{t_i}^{t_f} \left\| \sum_{i=0}^{n} P_i \dot{B}_{i,n}(t) \right\|^2 dt \tag{6a}$$

$$\text{s.t. } z(t_j) = w_j, \forall j \qquad \leftarrow \text{constrângere puncte} \rightarrow \quad s.t. \quad \sum_{i=0}^{n} P_i B_{i,n}(t_j) = w_j, \forall j. \tag{6b}$$

Deși teoretic se poate alege orice familie de funcții bază, în continuare lucrăm cu funcții Bezier[1] (au multe proprietăți interesante și sunt relativ simple):

$$B_{i,n}(t) = \binom{n}{i} (1-t)^{n-i} t^i, \quad \forall t \in [0,1], \tag{7}$$

ceea ce ne permite în plus, să exprimăm și $\dot{z}(t)$ ca o combinație de funcții Bezier (de ordin $n-1$ de data aceasta) și puncte de control:

$$\dot{z}(t) = n \sum_{i=0}^{n-1} (P_{i+1} - P_i) B_{i,n-1}(t). \tag{8}$$

Folosind (8) putem acum scrie problema de optimizare strict în funcție de punctele de control și de termeni numerici proveniți din evaluarea funcțiilor Bezier:

$$\min_{P_i} \sum_{i=0}^{n-1} \sum_{k=0}^{n-1} (P_{i+1} - P_i)^\top (P_{k+1} - P_k) \cdot n^2 \int_0^1 B_{i,n-1}(t) B_{k,n-1}(t) dt \tag{9a}$$

$$s.t. \sum_{i=1}^{n} P_i B_{i,n}(t_j) = w_j, \forall j. \tag{9b}$$

Deși nu este detaliat aici, același tip de raționament (re-formularea în funcție de punctele de control) se poate face și pentru alte tipuri de constrângeri: ocolire de obstacole, asigurarea vizibilității, etc.

[1] Termenul binomial este dat de formula: $\binom{n}{i} = \frac{n!}{i!(n-i)!}$.

2 Implementarea unei traiectorii Dubins în plan

Există numeroase librării ce generează traiectorii Dubins, spre exemplu:

- https://pypi.org/project/dubins/
- https://uuvsimulator.github.io/packages/uuv_simulator/docs/features/jupyter_notebooks/2d_dubins_path/

Totuși, adesea apar probleme de dependințe, de exemplu instalarea unor librării ce sunt dependente de platforma utilizată (Linux/Windows). O soluție (nu neaparat eficientă!) pentru a lucra totuși cu o librărie ce funcționează doar Linux este să combinați Visual Code cu o instalație WSL (https://code.visualstudio.com/docs/remote/wsl). În acest caz, veți avea nevoie și de o aplicație "xserver" pentru a vizualiza rezultatele. Urmăriți, spre exemplu pașii din https://techcommunity.microsoft.com/t5/windows-dev-appconsult/running-wsl-gui-apps-on-windows-10/ba-p/1493242 pentru o astfel de soluție.

În continuare, folosim codul din https://github.com/fgabbert/dubins_py care are avantajul major că este "simplu": nu apar dependințe, poate fi folosit indiferent de platformă.

Pentru ilustrare, acesta este fragmentul de cod ce calculează parametrii unei traiectorii LSL:

```
def dubinsLSL(alpha, beta, d):
    tmp0      = d + math.sin(alpha) - math.sin(beta)
    tmp1      = math.atan2((math.cos(beta)-math.cos(alpha)),tmp0)
    p_squared = 2 + d*d - (2*math.cos(alpha-beta)) + (2*d*(math.sin(
    alpha)-math.sin(beta)))
    if p_squared<0:
        print('No LSL Path')
        p=-1
        q=-1
        t=-1
    else:
        t         = (tmp1-alpha) % (2*math.pi)
        p         = math.sqrt(p_squared)
        q         = (beta - tmp1) % (2*math.pi)
    return t, p, q
```

Implementarea se bazează pe articolul [2] iar parametrii ce apar în fragmentul de cod respectă relația:

$$L_q(S_p(L_t(0,0,\alpha))) = (d,0,\beta),$$

ceea ce spune că pornind de la o configurație $(0,0,\alpha)$ se ajunge la $(d,0,\beta) \leftarrow$ se fac diverse schimbări de coordonate și presupuneri pentru a ajunge la aceste forme particulare. Implementarea din cod se bazează pe rezolvarea ecuațiilor:

$$\begin{aligned} p\cos(\alpha+t) - \sin\alpha + \sin\beta &= d, \\ p\sin(\alpha+t) + \cos\alpha - \cos\beta &= 0, \\ \alpha + t + q &= \beta \,[\mathrm{mod} 2\pi]. \end{aligned}$$

3 Exerciții propuse

Exerciţiul 1. Pentru codul din secțiunea anterioară:

i) Implementați un traseu cu mai multe puncte intermediare.

ii) Având o listă de puncte, implementați un mecanism de parcurgere pentru a găsi traseul de lungime minimă (compus ca o traiectorie Dubins).

Indicație: căutați/implementați un algoritm care rezolvă problema "comisvoiajorului" (traveling salesman problem)

Exerciţiul 2 (Tema 1 – 15p). Pentru o problemă de optimizare definită ca în (9), se cer următoarele:

i) Pentru o colecție dată de puncte de control, implementați și ilustrați traiectoria parametrizată de acestea, ($z(t)$, definită ca în (5)); **[5p]**

ii) Alegeți-vă o colecție de puncte intermediare și momente de timp asociate (w_j, t_j). Implementați, folosind Casadi, problema de optimizare din (9). Pentru colecția de puncte de control obținută, ilustrați $z(t)$; **[5p]**

iii) Ilustrați comenzile $u_V(t), u_\phi(t)$, folosind $z(t)$ obținut la pasul anterior, pe baza relațiilor din (4). **[5p]**

Bibliografie

[1] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[2] Andrei M Shkel **and** Vladimir Lumelsky. "Classification of the Dubins set". **in***Robotics and Autonomous Systems*: 34.4 (2001), **pages** 179–202.