Lab 4 – Bezier trajectories

REPLAN team

Tuesday 18th March, 2025

Contents

compiled at: 18/03/2025, 08:51

The idea

The notion of flat representation with parameterization through Bezier functions is also presented, for the subsequent solution of an optimization problem.

1 Theoretical background

We consider the simplified "Dubins car" which cannot slip laterally ("sideslip") and which can go only forward [1]:

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

The car is driven by $u_V \in \{-1, 1\}$ and $u_\theta \in (-\phi_{\max}, \phi_{\max})$. The state vector is given by position (x and y) and orientation (θ).

Taking $z = \begin{bmatrix} x & y \end{bmatrix}^\top$ as the flat output for system (1), we rewrite the remaining state (θ) and the control actions (u_V, u_θ) in function of it:

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

Consequently, we can reduce a typical movement planning problem, which involves, for example:

- passing through a list of way-points: $z(t_j) = w_j$,
- minimizing the energy along the path: $\int_{t_i}^{t_f} \|\dot{z}(t)\|^2 dt$,

to a problem which involves only variable $z(t)$.

For a practical implementation, we need to express $z(t)$ as a combination of basis functions, weighted by control points:

$$z(t) = \sum_{i=0}^{n} P_i B_{i,n}(t). \tag{3}$$

This allows us to reduce the above problem to one that uses as variables only the control points $P_0 \ldots P_n$ (instead of trying to get directly from $z(t)$ ← much more difficult and usually impossible):

$$\min_{z(t)} \int_{t_i}^{t_f} \|\dot{z}(t)\|^2 dt \qquad \leftarrow \text{cost} \rightarrow \qquad \min_{P_i} \int_{t_i}^{t_f} \left\| \sum_{i=0}^{n} P_i \dot{B}_{i,n}(t) \right\|^2 dt \tag{4a}$$

$$\text{s.t. } z(t_j) = w_j, \forall j \qquad \leftarrow \text{way-points} \rightarrow \qquad s.t. \quad \sum_{i=0}^{n} P_i B_{i,n}(t_j) = w_j, \forall j. \tag{4b}$$

Although theoretically any family of basis functions can be chosen, we prefer to work with Bezier functions[1] (they have many interesting properties and are relatively simple to manipulate):

$$B_{i,n}(t) = \binom{n}{i}(1-t)^{n-i}t^i, \quad \forall t \in [0,1], \tag{5}$$

which also allows us to express $\dot{z}(t), \ddot{z}(t)$ as a combination of Bezier functions (of order $n-1$, and respectively $n-2$, this time) and control points:

$$\dot{z}(t) = n \sum_{i=0}^{n-1} (P_{i+1} - P_i) B_{i,n-1}(t), \tag{6a}$$

$$\ddot{z}(t) = n(n-1) \sum_{i=0}^{n-2} (P_{i+2} - 2P_{i+1} + P_i) B_{i,n-2}(t). \tag{6b}$$

Using (6) we can now write the optimization problem strictly according to the control points and numerical terms derived from the evaluation of the Bezier functions:

$$\min_{P_i} \sum_{i=0}^{n-1} \sum_{k=0}^{n-1} (P_{i+1} - P_i)^\top (P_{k+1} - P_k) \cdot n^2 \int_0^1 B_{i,n-1}(t) B_{k,n-1}(t) dt \tag{7a}$$

$$s.t. \sum_{i=1}^{n} P_i B_{i,n}(t_j) = w_j, \forall j. \tag{7b}$$

Both $B_{i,n}(t_j)$ and $n^2 \int_0^1 B_{i,n-1}(t) B_{k,n-1}(t) dt$ are 'numbers' that can be easily computed at runtime or offline. Thus, the optimization problem (7) only depends on the control points $\{P_i\}$.

Although it is not detailed here, the same type of reasoning (re-formulating according to the checkpoints) can be done for other types of constraints: obstacle avoidance, ensuring visibility, etc.

[1] The binomial term is given by the formula: $\binom{n}{i} = \frac{n!}{i!(n-i)!}$.

2 Implementation

For a given list of waypoints and time instants at which we pass through them.

```
W = np.array([
    [0, 0],
    [1, 2],
    [3, 1],
    [4, 2],
    [5, 1]
 ])
tw = [0, 0.4, 0.6, 0.8, 1]
```

Next, we define and solve the optimization problem (7).

```
#  the number of Bezier curves >= the number of waypoints
n = 5
# instantiate the Opti class
solver = ca.Opti()
# define the variables
P = solver.variable(2, n+1)
epsilon = solver.variable(1)

# define a function that computes the energy along the path and use it
    to define the cost
solver.minimize(path_energy(P) + 100 * epsilon)

# pass through the waypoints
for j in range(m):
    Bs = [B(i, n, tw[j]) for i in range(n+1)]
    solver.subject_to(P @ Bs == W[j, :])
# limit the values for the control points
solver.subject_to(epsilon>=0)
for i in range(n+1):
    solver.subject_to(P[:, i] <= epsilon)
    solver.subject_to(P[:, i] >= -epsilon)

# attach a solver
solver.solver('ipopt')

# and solve the optimization problem
sol = solver.solve()
```

Once the problem is solved, we retrieve the control points and plot the results.

```
P_opt = sol.value(P)
t_values = np.linspace(0, 1, 100)
trajectory = np.array([z(t, P_opt) for t in t_values])
plot_bezier_curve(trajectory, P_opt, W, title='Bezier curve passing
    through list of waypoints')
```

3 Proposed exercises

Using and modifying the available code, solve the following exercises.

Exercise 1 (Homework 1 – 15p). For an optimization problem defined as in (7), the following is required:

i) For a given collection of m way-points, plot the total path length as a function of $n+1 \geq m$, the number of control points; **[5p]**

ii) Relax the constraint of way-point passing to a a neighborhood constraint

$$|z(t_j) - w_j| \leq \delta$$

and solve the modified (7). Select $\bar{\delta}$ such that the path length corresponding to $\delta = 0$ is at most 10% longer than the one corresponding to $\delta = \bar{\delta}$; **[5p]**

iii) Append to the constraints a condition which penalizes the velocity ($\|v(t)\| \leq \rho$) and solve (7), thus modified. Find the interval $\rho_{min} \leq \rho \leq \rho_{max}$ for which the problem has a solution. **[5p]**

Exercise 2. Solve iteratively the optimization problem (7) where the time instants t_j at which the trajectory has to pass through waypoints W_j, are themselves decision variables.
Indication: One idea is to use PSO - particle swarm optimization to solve the optimization problem (see `https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/`).

Exercise 3. Change (7) to penalize in the cost, the total control effort:

$$\int_0^1 \|u_V(t)\|^2 + \|u_\theta(t)\|^2 dt.$$

If this cannot be done exactly, what strategies do you propose?

References

[1] Steven LaValle. *Planning Algorithms / Motion Planning*. Cambridge university press, 2006. URL: `https://lavalle.pl/planning/` (visited on 02/04/2025).