CS 333: Safe and Interactive Robotics — Fall 2017

Lecture 2: Motion Planning, Trajectory Optimization

Scribes: Maxime Bouton, Keven Wang, Mingyu Wang

2.1 Motion Planning: Sampling Based Methods

The sampling based methods require:

- A collision checker: that checks if point in configuration space q collides with any obstacles C_{obs}:

$$\gamma(q) = \begin{cases} 1 & if\ q \in C_{obs} \\ 0 & otherwise \end{cases}$$

- A simple planner: that is fast, but not complete nor optimal.

$$B(q_1, q_2) = \begin{cases} A\ path\ from q_1\ to\ q_2 \\ Failure \end{cases}$$

Sampling based methods are probabilistically complete. In the case we don't know the entire configuration space, we plan in horizon where we know the collision checker function.

2.1.1 Probabilistic Roadmaps (PRM)

In the PRM graph construction phase, random points are sampled from the configuration space, with only points lying in the free configuration space C_{free} kept. A local planner connects the sampled points to neighbors. A graph search algorithm attempts to find a path between start point q_s and end point q_g in the configuration space. The above steps are repeated until a path is found between q_s and q_g.

Algorithm:

```
Given $q_s, q_g$
while $q_s, q_g$ not in the same connected component do
    Sample M points in configuration space
    Remove points that collides with obstacles using collision checker $\gamma$
    Connect new points to existing points using fast planner B
end
```

2.1.2 Rapidly-exploring Random Trees (RRT)

Rapidly exploring random trees algorithm is similar to Probabilistic Roadmap, except that it only samples one point in C_{free} at a time. The sampling is done with bias toward area not yet sampled in the space. The sampled point is then connected to connected groups C_s or C_g using simple planner. Initially $C_g =$

$q_g, C_s = q_s$. One variant of RRT algorithm, Bi-directional RRT, attempts to build tree from both start and end of the configuration space. RRT is fast and scalable, but is not optimal given sampled nodes.

Algorithm:

```
$C_g = q_g$
$C_s = q_s$
**while** *$q_s, q_g$ not in the same connected component* **do**
    Sample one point in $C_{free}$ space
    Connect point to closest point in $C_g$ or $C_s$ using simple planner
**end**
```

2.1.3 Rapidly-exploring Random Trees Star (RRT*)

RRT* is the same as vanilla RRT, except that it include post-processing step after each iteration:

- Parent selection (figure 2.1): examine neighboring nodes (in radius according to some hyper parameter), if becomes parent, would result in shorter path
- Rewiring (figure 2.2): drop existing edge, if going through new node results in shorter path

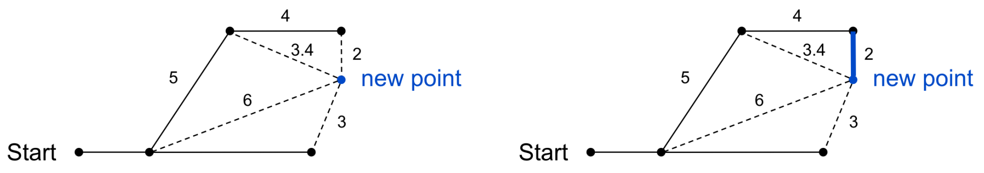


Figure 2.1: Parent selection step

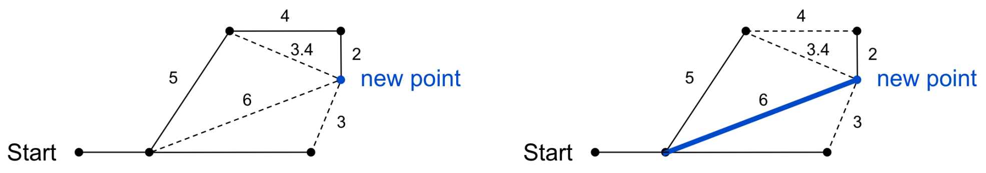


Figure 2.2: Rewiring step

RRT* is optimal given sample nodes.

Algorithm:

```
$C_g = q_g$
$C_s = q_s$
**while** *$q_s, q_g$ not in the same connected component* **do**
    Sample one point in $C_{free}$ space
    Connect point to closest point in $C_g$ or $C_s$ using simple planner
    Parent selection
    Rewiring
**end**
```

2.2 Trajectory Optimization

Trajectory is a function which maps time to C-space configurations

$$\xi : [0, T] \to \mathcal{C}, \xi \in \Xi,$$

where Ξ is a set of all possible trajectories.

To distinguish different trajectories, we further define a cost functional $\mathcal{U}$ that maps any trajectory to a non-negative value:

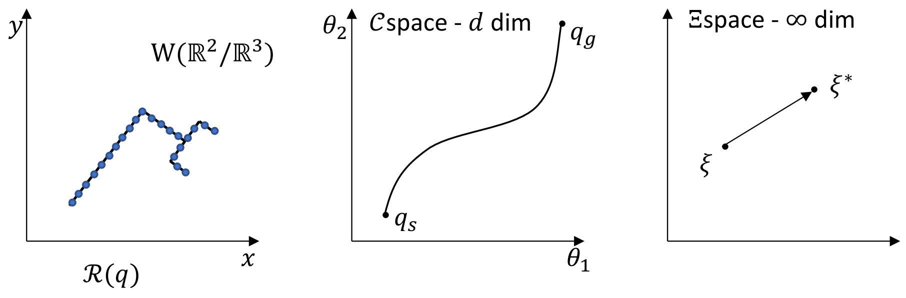
$$\mathcal{U} : \Xi \to \mathbb{R}^+$$

Our goal is to optimize $\mathcal{U}$.

Some example factors to consider when constructing the cost functionals are:

- path length
- efficiency
- obstacle avoidance
- uncertainty reduction
- predictability
- legibility/ intent expression
- human comfort
- naturalness

To better understand the trajectory optimization problem, let's take a look at the worlds where the robot, the robot's configuration and the trajectory live in. The robot configuration q, is a point in $\mathcal{C}$-space with dimension d, where d is the degree of freedom of the robot. $R(q)$ maps the configuration of a robot to a set of points where the robot occupies in the world space (typically $\mathbb{R}^2$ or $\mathbb{R}^3$). A trajectory, ξ, is a timed path in robot's configuration space. Ξ is a space of trajectory functions and thus is ∞-dimensional.


y
x
$\mathrm{W}(\mathbb{R}^2/\mathbb{R}^3)$
$\mathcal{R}(q)$
θ_2
$\mathcal{C}$space - d dim
q_g
q_s
θ_1
Ξspace - ∞ dim
ξ^*
ξ

Trajectory Optimization is the process of finding an optimal trajectory ξ^*:

$$\xi^* = \arg\min_{\xi \in \Xi} \mathcal{U}[\xi]$$

$$\begin{aligned} s.t. \quad & \xi(0) = q_s \\ & \xi(T) = q_g \end{aligned}$$

One method to solve this trajectory optimization problem is Gradient Descent:

$$\xi_{i+1} \leftarrow \xi_i - \frac{1}{\alpha}\nabla_\xi \mathcal{U}(\xi_i)$$

To get an intuition why this works, consider we have a trajectory ξ_i and want to minimize $\mathcal{U}[\xi]$:

$$\xi_{i+1} = \arg\min_{\xi\in\Xi}\{\mathcal{U}[\xi_i] + \nabla_{\xi_i}\mathcal{U}^T(\xi - \xi_i) + \frac{1}{2}\alpha\|\xi - \xi_i\|^2\}$$

The first two terms approximate $\mathcal{U}[\xi]$ by first order Taylor expansion, and the third term penalizes if ξ is far away from ξ_i. Since the right-hand side is a convex function, we take the gradient of the right-hand side and set it to zero

$$0 + \nabla_{\xi_i}\mathcal{U} + \alpha(\xi - \xi_i) = 0$$

Solve the above equation, we get

$$\xi = \xi_i - \frac{1}{\alpha}\nabla_\xi \mathcal{U}(\xi_i),$$

which is the same equation as Gradient Descent.

Calculus Review:

In trajectory optimization, we care about vector calculus and multivariable calculus because the trajectory and the the cost can be represented as vector valued function and multivariable functions respectively.

$$\xi : [0, T] \longrightarrow \mathcal{C}$$

$$\xi(t) = q = \begin{bmatrix} q_1(t) \\ \vdots \\ q_d(t) \end{bmatrix}$$

where d is the dimension of the configuration space $\mathcal{C}$. ξ is represented as a vector valued function. If we discretize time we can also write:

$$\xi = \begin{bmatrix} q_1 \\ \vdots \\ q_N \end{bmatrix}$$

where N is the number of time steps. Then $\mathcal{U}(\xi) = \mathcal{U}(q_1, \ldots, q_N)$, it is a multivariable function.

Note: If the time is not discretized, $\mathcal{U}$ is a different object called a functional ("function of functions").

Derivatives and Gradients:

For single variable functions $f : \mathbb{R} \to \mathbb{R}$ the derivative is defined as follows:

$$f'(x) = \lim_{\epsilon\to 0}\frac{f(x+\epsilon) - f(x)}{\epsilon}$$

Example:

$$f(x) = 2x$$

$$f'(x) = \lim_{\epsilon\to 0}\frac{2(x+\epsilon) - 2x}{\epsilon} = 2$$

If f is a multivariable function: $f(x_1, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}$, we define the partial derivatives with respect to each variable x_i to be:

$$\frac{\partial f(x_1, \ldots, x_n)}{\partial x_i} = \lim_{\epsilon \to 0} \frac{f(x_1, \ldots, x_i + \epsilon, \ldots, x_n) - f(x_1, \ldots, x_n)}{\epsilon}$$

The gradient of f is then given by $\nabla_{x_1,\ldots,x_n} f(x_1, \ldots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$

Example:

$$\begin{aligned} &f(x,y) = 2x + y + xy \\ &\quad \frac{\partial f}{\partial x} = 2 + y, \quad \frac{\partial f}{\partial y} = 1 + x \\ &\quad \nabla_{x,y} f = \begin{bmatrix} 2+y \\ 1+x \end{bmatrix} \end{aligned}$$

For vector valued function $f : \mathcal{R} \to \mathcal{R}^n$, the derivative is just the derivative of the components:

$$f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{bmatrix} \qquad f'(x) = \begin{bmatrix} f_1'(x) \\ \vdots \\ f_n'(x) \end{bmatrix}$$

When a function is multivariable and vector valued: $f : \mathcal{R}^n \to \mathcal{R}^m$ we define the Jacobian (of size $m \times n$):

$$\frac{\mathrm{d}f}{\mathrm{d}(x_1, \ldots, x_n)} = J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Note: For linear functions f such that $f(x_1, \ldots, x_n) = Ax$ where $x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ Then A is the jacobian of f evaluated at x.

Example:

$$\begin{aligned} f(x,y) &= \begin{bmatrix} 2x + y \\ y \end{bmatrix} \\ J &= \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

Hilbert Space:

The space Ξ of all the trajectories is a Hilbert Space. It is a complete vector space with an inner product defined over it. The inner product can be euclidean:

$$\xi_1, \xi_2 \in \Xi, \quad \langle \xi_1, \xi_2 \rangle = \int_0^T \xi_1(t)^\top \xi_2(t) \mathrm{d}t$$

An inner product verifies the following properties:

- Symmetry: $\langle \xi_1, \xi_2 \rangle = \langle \xi_2, \xi_1 \rangle$
- Linearity:

$$\begin{aligned} \langle \xi_1 + \xi_2, \xi_3 \rangle &= \langle \xi_1, \xi_3 \rangle + \langle \xi_2, \xi_3 \rangle \\ \langle a\xi_1, \xi_2 \rangle &= a\langle \xi_1, \xi_2 \rangle \text{ where } a \in \mathbb{R} \end{aligned}$$

- Positive Definite:

$$\forall \xi \ \langle \xi, \xi \rangle \geq 0 \tag{2.1}$$
$$\langle \xi, \xi \rangle = 0 \text{ if and only if } \xi = 0 \tag{2.2}$$