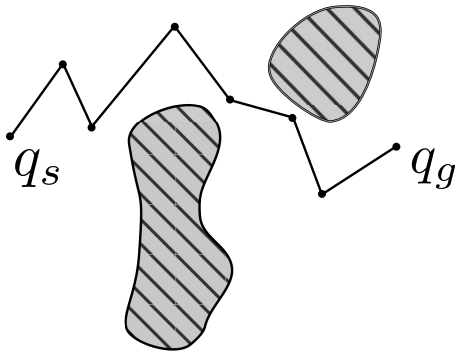
CS 294-115 Algorithmic Human-Robot Interaction **Fall 2016**

Lecture 5: Trajectory Optimization

Scribes: Esther Rolf, Matthew Matl

5.1 Overview

Last lecture, we saw that randomized planners can get us from a start configuration to a goal configuration while overcoming some of the issues associated with high dimensionality. In practice, RRT runs quickly (milliseconds to seconds in a 7 d.o.f. robot). However, there are still several drawbacks of RRT. It is only probabilistically complete, which is a rather weak assertion, as we discussed last time. RRT is also not optimal, and as we see in practice, it is *very* non-optimal. For example, consider the figure below, which shows an example path that RRT might generate in C-space.



This path in C-space will zig-zag randomly, and the corresponding motion in world-space can be very jerky and unintuitive, which is a problem when robots are working with humans. One way to address this is to use a post-processing step called *shortcutting*, which samples pairs of points on the path returned by RRT and tries to connect them with a straight line in C-space. If the connection will not cause a collision, it is added to the path and redundant vertices are removed so that the overall path is shorter. This takes a long time, can still result in paths that are in the wrong homotopy class, can still result in jerky and counterintutive paths.

5.2 Optimal Motion Planning

The idea behind optimal motion planning (optimal randomized sampling based planners) is to leverage the benefits of randomized sampling (namely, reaching a feasible solution quickly), with asymptotic optimality. Here, asymptotic optimally means that as the number of iterations goes to infinity, the resulting trajectory converges to an optimal one.

An example of an optimal motion planning algorithm is RRT*, introduced by [Karman, Frazzoli '2010]. RRT* is based on unidirectional RRT with two modifications:

1. Parent selection. When a new point is sampled, rather than making it a child of the closest node in the graph so far, we connect it to the point that would produce the shortest path from the starting point.
2. Rewiring Step. Once a new point x_s is sampled and added to the graph, consider whether we can improve the existing graph using this new point. For every node x_v in the vicinity of the new point, check if that point can be reached faster by an alternate path using x_s. If so, make x_s the parent of x_v, and delete all redundant edges so that the graph is again a tree.

RRT* is an asymptotically optimal algorithm. However, the convergence to the optimal solution is slow, and the algorithm itself can be slow due to all the extra graph modification steps. While RRT may be more feasible for use in practice, RRT* is an important theoretical result.

5.3 Trajectory Optimization

When robots are working in an environment with people, it's important that they move in an understandable way, and in a way that humans can anticipate. Trajectory optimization can get us closer to that goal. The idea behind trajectory optimization is to start with a simple path, and let the obstacles guide you away from collision while optimizing for efficiency/smoothness or other costs. The resulting algorithm is:

- probabilisitically complete
- fast
- locally optimal

The problem statement of trajectory optimization is to find a trajectory ξ which maps time to C-space configurations:

$$\xi : [0, T] \to C$$

Here $\xi \in \Xi$ is a particular trajectory among a set of possible trajectories Ξ. A trajectory ξ is a function of $t \in [0, T]$ which produces a C-space configuration $q \in C$. Thus, a trajectory tells the robot where to be for each point in time.

There are many possible trajectories in the set Ξ. To distinguish between them, we define a *cost functional* (also called an objective functional) $\mathcal{U}$, which maps trajectories onto non-negative scalars[1] :

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

Our goal is to optimize $\mathcal{U}$, resulting in an optimal trajectory, where optimal is defined according to the factors encoded in $\mathcal{U}$. Some potential factors encoded in $\mathcal{U}$ include:

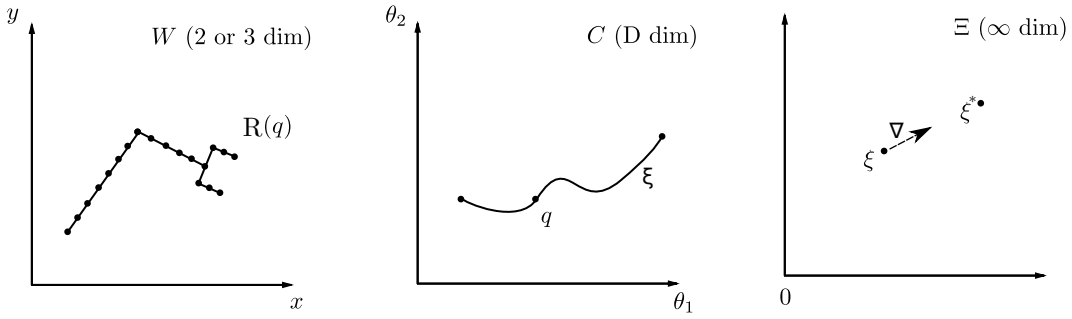
- length or efficiency of the resulting path (in C-space or world-space)
- obstacle avoidance
- smoothness
- caution/ slow-down near obstacles
- energy minimization

[1] $\mathcal{U}$ maps functions to scalars, and thus is a *functional*.

- "naturalness"
- information gathering
- predictability
- legibility / intent expression

Before we define the technical optimization problem, consider the worlds in which each of these elements live. A robot's pose, $R(q)$, in world space $W(\mathbb{R}^3)$ is a single point, q, in C-space $C(D)$, where D is robot's number of degrees of freedom. A trajectory, ξ, is a timed path through C-space and is a single point in trajectory-space, Ξ. Ξ is a space of functions and is thus infinite-dimensional.

In our optimization problem, we will seek to find an optimal $\xi \in \Xi$. We will be using gradient descent on $\mathcal{U}i$.



Definition

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

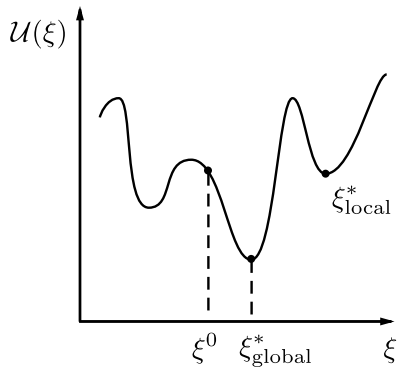
$$\begin{aligned} & \xi^* = \arg\min_{\xi \in \Xi} \mathcal{U}[\xi] \\ \text{s.t.} \quad & \xi(0) = q_s \\ & \xi(T) = q_g \end{aligned}$$

One method for solving this optimization problem is gradient descent (GD), which iteratively updates the guess for an optimal ξ:

$$\xi_{t+1} = \xi_t - \frac{1}{\alpha} \nabla_\xi \mathcal{U}(\xi_t)$$

It is important to note that gradient descent returns a local minimizer, not a global minimizer of the cost functional. We can minimize globally for some $\mathcal{U}$ (convex), but not for general $\mathcal{U}$.

Unfortunately, most objective functions we will want to use are non-convex. Because we are only guaranteed a local minimum, we might threshold an acceptable $\mathcal{U}^* = \mathcal{U}(\xi^*)$, and if the $\mathcal{U}^*$ returned by GD is above this threshold, resample using a different initialization ξ_0, or introduce randomization in more sophisticated ways (e.g. Hamiltonian Monte Carlo).



5.4 Calculus Review

5.4.1 Motivation

In this section, we review the basics of single-variable, multivariate, and vector calculus. Before doing so, however, it is useful to think about how calculus can be used in trajectory optimization. Consider a trajectory $\xi : [0, T] \to C$ for a configuration space with d dimensions. This trajectory is a function that maps from some time $t \in [0, T]$ to a vector q of coordinates in C-space:

$$\xi(t) : t \mapsto q, \; q = \begin{bmatrix} \xi_1(t) \\ \xi_2(t) \\ \vdots \\ \xi_d(t) \end{bmatrix}$$

Because of this, we say that ξ is a *vector-valued function* – a function operates on one or more variables and maps them to vectors. In order to analyze these sorts of functions, we need to understand *vector calculus*.

Furthermore, if we discretize time with N timesteps, then ξ becomes a vector of configuration vectors:

$$\xi = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{bmatrix}$$

Then, our cost function for trajectory optimization, $\mathcal{U} : \Xi \to \mathbb{R}^+$ maps from some discrete trajectory, which is simply a vector, to a single real number. This mapping from many variables to a single value makes $\mathcal{U}$ a *multivariate function*, and we need *multivariable calculus* in order to analyze this function properly.

Finally, if we do not discretize trajectories, $\mathcal{U}$ is a functional, and we need *calculus of variations*.

5.4.2 Derivative

Definition

Let $f(x) : \mathbb{R} \to \mathbb{R}$ be a function of a single real variable. The *derivative* of $f(x)$ is a function of that same variable that, when evaluated at a given input point, produces the rate of change of f at that point. The derivative of $f(x)$, which is written as $f'(x)$, can be defined as

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

Example

Let $f(x) = 2x$. We can compute the derivative as follows.

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

In this case, the derivative is a constant. This is not always the case.

Let $f(x) = x^2$. Omitting all steps for brevity, we can show that

$$f'(x) = 2x$$

In this case, the derivative is a function of x.

5.4.3 Partial Derivative

Definition

A *partial derivative* is an extension of the concept of a derivative to a multivariate function. Let $f(x_1, x_2, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}$ be a function of n real variables. Partial derivatives of f are always defined with respect to only one of f's variables. Specifically, the partial derivative of f with respect to x_i is a multivariate function of f's variables that, when evaluated at a given input point, produces the rate of change of f in the direction of x_i.

In order to calculate the partial derivative of f with respect to x_i, we assume that all variables but x_i are held constant. This reduces f to a function of only x_i, and we can calculate the partial derivative, $\frac{\partial f}{\partial x_i}$, as the simple derivative of this new function using a limit:

$$\frac{\partial f}{\partial x_i}(x_1, x_2, \ldots, x_n) = \lim_{\epsilon \to 0} \frac{f(x_1, x_2, \ldots, x_{i-1}, x_i + \epsilon, x_{i+1}, \ldots, x_n) - f(x_1, x_2, \ldots, x_n)}{\epsilon}$$

Example

Let $f(x, y) = 2x + y + xy$. We can calculate the derivative of f with respect to x as follows:

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

Similarly,

$$\frac{\partial f}{\partial y} = 1 + x$$

5.4.4 Gradient

Definition

A gradient is a generalization of the concept of partial derivatives. Specifically, let $f(x_1, x_2, \ldots, x_n)$ be a function of n real variables. Then, the gradient is simply a vector whose elements are the n partial derivatives of f with respect to each of its variables:

$$\nabla_{x_1,x_2,\ldots,x_n} f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

The gradient is a function that maps from f's variables to a vector. Thus, we can describe the gradient as a multivariate vector-valued function. When the gradient is evaluated at a given input point, it produces a vector in $\mathbb{R}^n$ which points in the direction of the greatest rate of ascent in f and whose magnitude specifies that rate of ascent.

Example

Let $f(x, y) = 2x + y + xy$. We computed the partial derivatives of this function above, so we can simply say that

$$\nabla_{x,y} f = \begin{bmatrix} 2 + y \\ 1 + x \end{bmatrix}$$

5.4.5 Ordinary Derivatives of Vector-Valued Functions

Definition

A vector-valued function is a function whose range is a set of vectors. For example, let

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix}$$

be a single-variable vector-valued function. This function maps a single variable into a n-dimensional vector. Because f is a function of a single variable, we can take its ordinary derivative by simply differen-

tiating each of the functions $f_1, f_2, \ldots, f_n$ that compose f's matrix:

$$f'(x) = \begin{bmatrix} \frac{df_1}{dx} \\ \frac{df_2}{dx} \\ \vdots \\ \frac{df_n}{dx} \end{bmatrix}$$

This ordinary derivative is a vector of functions of a single variable, and when it is evaluated at a given input point, it produces a vector.

Example

Let

$$f(x) = \begin{bmatrix} x^2 + 2 \\ 2x \end{bmatrix}$$

Then, we can compute

$$f'(x) = \begin{bmatrix} 2x \\ 2 \end{bmatrix}$$

This is a vector of functions of x, and we can evaluate it at a particular point to get a vector.

5.4.6 Jacobians of Multivariate Vector-Valued Functions

Definition

The Jacobian matrix is a generalization of the gradient to multivariate vector-valued functions. Let $f(x_1, x_2, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}^m$ be a multivariate vector-valued function:

$$f(x_1, x_2, \ldots, x_n) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_m(x_1, x_2, \ldots, x_n) \end{bmatrix}$$

Then, the Jacobian of f is defined as

$$\frac{df}{d(x_1, x_2, \ldots, x_n)} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

This matrix is often represented by $\mathbf{J}$.

Usage in Kinematics

Let $\phi_{EE}(q)$ be a forward kinematics function that maps a configuration q from d-dimensional C-space to a location of a robot's end effector in $\mathbb{R}^3$:

$$\phi_{EE}(q) = \begin{bmatrix} x(q) \\ y(q) \\ z(q) \end{bmatrix}$$

The Jacobian of ϕ gives us a 3 by d matrix whose rows are the gradients of x, y, and z with respect to the configuration q. This is very useful, as it gives us a way to map a rate of change in configuration space to a rate of change in the real world.

The Jacobian is also useful in inverse kinematics with the psuedo-inverse Jacobian and Jacobian transpose methods.

Properties

If we have a vector

$$v = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

then if, for some matrix $\mathbf{A}$, we can write $f(x_1, x_2, \ldots, x_n) = \mathbf{A}v$ then $\mathbf{J} = \mathbf{A}$.

Example

Let

$$f(x, y) = \begin{bmatrix} 2x + y \\ y \end{bmatrix}$$

Then, we can calculate

$$\frac{df}{d(x, y)} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}$$

We could compute this directly or by noting that

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

and using the first property of the Jacobian listed above.