Lecture 8

Trajectory Optimization and Model-Based Policy Search

Up to this point, we've spent most of our time discussing value-based methods. Now we turn our attention to policy-based learning methods. In particular, we consider the domain of "policy-search" algorithms, where instead of searching for a parameterization of the "cost-to-go" or other scalar function, we search for a parameterization of the policy.

8.1. Model-Based vs. Model-Free Policy Search

As in value-based methods, there is a large spectrum of model-based vs. model-free policy search algorithms. In general, model-based policy search methods leverage a model, and very often leverage the ability to compute analytic gradients for that model. In some cases, the underlying model-free and model-based methods are identical, but the model-based methods leverage a simulation to generate samples for computing the policy.

8.2. Trajectory Optimization

So far, we have been discussing ways of computing feedback controllers, or closed-loop policies using value-based methods. However, when considering policy-search methods, it is often common to consider the notion of an "open-loop" policy. In this case, there is no associated feedback controller.

As before, we will write our optimization problem as

$$\min_u g_f(x(t_f)) + \int_{t_0}^{t_f} g(x(t), u(t))dt \tag{8.1}$$

$$\text{s.t.} \quad \dot{x} = f(x(t), u(t)), \quad \forall t \in [t_0, t_f], \tag{8.2}$$

$$x(t_0) = x_0 \tag{8.3}$$

$$c_i(x(t), u(t)) \geq 0, \forall i \tag{8.4}$$

where c_i represents a particular constraint, as might be imposed due to obstacle avoidance or some bound on the input.

In general, trajectory optimization methods can be classified as *indirect* or *direct* methods and as *shooting* or *collocation* methods.

Indirect trajectory optimization methods use calculus of variations or the maximum principle to solve the trajectory optimization problem numerically. Direct methods directly transform an optimal control problem into a nonlinear program.

8.3.1 Lagrange Multipliers

Example:

Consider the optimization problem

$$\min_{x+y} x + y \tag{8.5}$$

$$\text{s.t. } x^2 + y^2 = 1. \tag{8.6}$$

Form the Lagrangian:

$$L = x + y + \lambda(x^2 + y^2 - 1). \tag{8.7}$$

To minimize, take the partial derivatives and set equal to zero:

$$\frac{\partial L}{\partial x} = 1 + 2x\lambda = 0 \tag{8.8}$$

$$\frac{\partial L}{\partial y} = 1 + 2y\lambda = 0. \tag{8.9}$$

This gives

$$\lambda = \frac{-1}{2x} = \frac{-1}{2y} \tag{8.10}$$

$$\frac{\partial L}{\partial \lambda} = x^2 + y^2 - 1 = 0. \tag{8.11}$$

Substituting in, we get

$$x = y = \pm\sqrt{\frac{1}{2}}. \tag{8.12}$$

To minimize $x + y$ we need $x = y = -\sqrt{\frac{1}{2}}$.

8.3.2 Derivation of the Adjoint Equations

We desire to minimize the cost

$$\min_{x,u} g_f(x_N, u_N) + \sum_{k=0}^{N-1} g(x_k, u_k) \tag{8.13}$$

such that

$$x_{k+1} = f(x_k, u_k). \tag{8.14}$$

We can write down the Lagrangian as follows:

$$L = g_f(x_N, u_N) + \sum_{k=0}^{N-1} g(x_k, u_k) + \sum_{k=0}^{N-1} \lambda_k^T (f(x_k, u_k) - x_{k+1}). \tag{8.15}$$

We then take the partial derivatives

$$\frac{\partial L}{\partial \lambda_k} = f(x_k, u_k) - x_{k+1} = 0 \tag{8.16}$$

$$\tag{8.17}$$

and

$$\frac{\partial L}{\partial x_k} = \frac{\partial g}{\partial x_k} + \lambda_k^T \frac{\partial f}{\partial x_k} - \lambda_{k-1}^T = 0. \tag{8.18}$$

The sum

$$\sum_{k=0}^{N-1} \lambda_k^T (f(x_k, u_k) - x_{k+1}) \tag{8.19}$$

when inspect at time $k-1$ gives

$$\lambda_{k-1}^T (f(x_{k-1}, u_{k-1}) - x_k). \tag{8.20}$$

Taking the partial derivative with respect to u_k, we have

$$\frac{\partial L}{\partial u_k} = \frac{\partial g}{\partial u_k} + \lambda_k^T \frac{\partial f}{\partial u_k} = 0. \tag{8.21}$$

Re-writing these conditions, we have

$$\lambda_{N-1}^T = \frac{\partial g_f}{\partial x_N} \tag{8.22}$$

$$\lambda_{k-1}^T = \frac{\partial g}{\partial x_k} + \lambda_k^T \frac{\partial f}{\partial x_k}, \tag{8.23}$$

which we can implement in a backward pass to compute the gradient from λ_k as

$$\frac{\partial L}{\partial u_k} = \frac{\partial g}{\partial u_k} + \lambda_k^T \frac{\partial f}{\partial u_k}. \tag{8.24}$$

The forward pass can be carried out as

$$x_{k+1} = f(x_k, u_k) \tag{8.25}$$

and $\frac{\partial L}{\partial u_k}$ can be used as part of a gradient descent approach for trajectory optimization:

$$u_k = u_k - \alpha \frac{\partial L}{\partial u_k} \tag{8.26}$$

where $\alpha > 0$. This is a particular efficient indirect shooting method known as Backpropagation Through Time (BPTT).

It is also the same algorithm used to train neural networks!

8.3.3 Pontryagin's Minimum Principle

There is also a continuous-time version of the above derivation. Given the initial conditions, x_0, a continuous dynamics, $\dot{x} = f(x, u)$, and the instantaneous cost $g(x, u)$, for a trajectory $x^*(t)$ and $u^*(t)$ defined over $t \in [t_0, t_f]$ to be optimal, it must satisfy the conditions that

$$\forall t \in [t_0, t_f], \quad \dot{x}^* = f(x^*, u^*), \quad x^*(0) = x_0 \tag{8.27}$$

$$\forall t \in [t_0, t_f], \quad -\dot{\lambda}^* = \frac{\partial g}{\partial x}^T + \frac{\partial f}{\partial x}^T \lambda^*, \quad \lambda^*(T) = \frac{\partial g}{\partial x}^T \tag{8.28}$$

$$\forall t \in [t_0, t_f], \quad u^* = \arg\min_{u \in U} [g(x^*, u) + (\lambda^*)^T f(x^*, u)] \tag{8.29}$$

8.4. Direct Trajectory Optimization

Direct trajectory optimization methods *directly* transform an optimal control problem into a nonlinear program.

Consider our original optimization problem

$$\min_u g_f(x(t_f)) + \int_{t_0}^{t_f} g(x(t), u(t))dt \tag{8.30}$$

$$\text{s.t.} \quad \dot{x} = f(x(t), u(t)), \forall t \in [t_0, t_f], \tag{8.31}$$

$$x(t_0) = x_0. \tag{8.32}$$

8.4.1 Direct Shooting

Let us choose a numerical integration approach, such as Euler integration, where $x_{k+1} = x_k + f(x_k, u_k)dt$.

We can rewrite our optimization problem in discrete time as

$$\min_u g_f(x_N) + \sum_{k=0}^{N-1} g(x_k, u_k)dt \tag{8.33}$$

$$\tag{8.34}$$

where

$$x_{k+1} = x_k + f(x_k, u_k)dt, \quad \forall k \in [0, N], \tag{8.35}$$

$$x_0 = x_i. \tag{8.36}$$

In a direct shooting approach, we satisfy the dynamics constraints via forward integration and only u_k is an optimization parameter. For a gradient descent approach, we must calculate $\frac{\partial J}{\partial u_k}$, where $J = g_f(x_N) + \sum_{k=0}^{N-1} g(x_k, u_k)dt$. We can calculate these gradients either numerically via finite differences or analytically. The analytical gradients are given as

$$\frac{\partial J}{\partial u_k} = \frac{\partial g_f}{\partial u_k} + \sum_{k=0}^{N-1} \left(\frac{\partial g}{\partial x_k} \frac{\partial x_k}{\partial u_k} + \frac{\partial g}{\partial u_k} \right) \tag{8.37}$$

where

$$\frac{\partial x_{k+1}}{\partial u_k} = \frac{\partial f}{\partial x_k} \frac{\partial x_k}{\partial u_k} + \frac{\partial f}{\partial u_k}. \tag{8.38}$$

8.4.2 Direct Transcription

Shooting methods can be sensitive to initial conditions; it is also challenging to incorporate path constraints (such as obstacle avoidance). Direct transcription approaches incorporate dynamic feasibility as an explicit constraint in the optimization problem formulation.

$$\min_{u,x} g_f(x_N) + \sum_{k=0}^{N-1} g(x_k, u_k)dt \tag{8.39}$$

$$\text{s.t. } x_{k+1} = x_k + f(x_k, u_k)dt, \quad \forall k \in [0, N], \tag{8.40}$$

$$x_0 = x_i \tag{8.41}$$

$$c_i(x(t), u(t)) \geq 0, \forall i \tag{8.42}$$

8.4.3 Direct Collocation

Collocation methods represent the time-varying trajectories as splines. The interesting insight here is that for a state trajectory represented by a cubic polynomial and the input trajectory represented as a first-order polynomial, we can apply constraints at "collocation" points to enforce dynamic feasibility.

If $x = c_0 + c_1 s + c_2 s^2 + c_3 s^3$, where $s \in [0, 1]$ Here $x(0) = x_1$, $x(1) = x_2$, $\frac{\partial x}{\partial s(0)} = \dot{x}_1$, and $\frac{\partial x}{\partial s(1)} = \dot{x}_2$.

We can then write

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ \dot{x}_1 \\ x_2 \\ \dot{x}_2 \end{bmatrix}. \tag{8.43}$$

Next, we can then write

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \dot{x}_1 \\ x_2 \\ \dot{x}_2 \end{bmatrix}. \tag{8.44}$$

Now, let us choose our collocation point to be the midpoint for the spline

$$x_c = (x_1 + x_2)/2 + h(f_1 - f_2)/8 \tag{8.45}$$
$$\dot{x}_c = -3(x_1 - x_2)/(2h) - (f_1 + f_2)/4 \tag{8.46}$$
$$\Delta = f_c - \dot{x}_c. \tag{8.47}$$

Interestingly, if we employ direct transcription with Hermite-Simpson Integration, we get

$$x_0 - x_1 + (h/6.0)(\dot{x}_k + 4\dot{x}_c + \dot{x}_{k+1}) = 0 \tag{8.48}$$
$$x_c = (x_0 + x_1)/2 + h(\dot{x}_0 + \dot{x}_1)/8. \tag{8.49}$$

In this case, you get the same accuracy but you are linear in h. These direct transcription and collocation approaches are often solved using sequential quadratic programming solvers such as SNOPT, or interior point methods such as IPOPT.

Bibliography

[1] Russ Tedrake. *Underactuated Robotics*. 2023.