Lecture 5

Iterative LQR and Differential Dynamic Programming

We have seen that we can apply dynamic programming numerically through value iteration to achieve control of a large class of nonlinear systems subject to dimensionality constraints. We have also been able to apply dynamic programming analytically to solve the special case of linear time-invariant systems.

However, we can in fact, apply dynamic programming to nonlinear systems, if we are willing to find a solution that is only locally valid.

5.1. Time-Varying LQR for Trajectory Tracking

Assume we have a nominal trajectory given by $x_0(t)$ and $u_0(t)$.

Define a local coordinate system relative to that trajectory where $\bar{x}(t) = x(t) - x_0(t)$ and $\bar{u}(t) = u(t) - u_0(t)$. We can then write

$$\dot{\bar{x}} = \dot{x} - \dot{x}_0 = f(x, u) - f(x_0, u_0) \tag{5.1}$$

$$\dot{\bar{x}} \approx f(x_0, u_0) + \frac{\partial f(x_0, u_0)}{\partial x}(x - x_0) + \frac{\partial f(x_0, u_0)}{\partial u}(u - u_0) - f(x_0, u_0) \tag{5.2}$$

$$\approx A(t)\bar{x} + B(t)\bar{u}. \tag{5.3}$$

Then use LQR to compute $K(t)$.

Therefore, if we have a feasible trajectory, we can easily compute a time-varying feedback policy to achieve tracking of that trajectory.

The linearization required may seem limiting, but in practice it is really quite powerful and can provide a means of controlling fairly challenging underactuated systems.

5.2. Iterative LQR

Still, it would be nice if we could apply the machinery of dynamic programming to arbitrary optimal control problems. It turns out that there is a way to use the principles of LQR (linearizing about a trajectory) to solve for both the feedback policy and the trajectory.

Consider the value function, V_k and associated Q-function, Q_k. Here

$$V_k = \min_u \left[g(x_k, u_k) + V_{k+1}(f(x_k, u_k)) \right] \tag{5.4}$$

$$V_k = \min_u Q_k(x_k, u_k), \tag{5.5}$$

where $Q_k = g(x_k, u_k) + V_{k+1}(f(x_k, u_k))$.

Approximate Q_k to second order about a nominal trajectory. Let $\delta x = x - x^i$ and $\delta u = u - u^i$.

To start, we take the second order expansion of $g(x_k, u_k)$ about x_k^i and u_k^i.

$$g(x_k, u_k) = g(x_k^i, u_k^i) + \frac{\partial g^T}{\partial x_k}\delta x_k + \frac{\partial g^T}{\partial u_k}\delta u_k + \frac{1}{2}\delta x_k^T \frac{\partial^2 g}{\partial x_k^2}\delta x_k + \frac{1}{2}\delta u_k^T \frac{\partial^2 g}{\partial u_k^2}\delta u_k + \delta u_k^T \frac{\partial^2 g}{\partial u_k \partial x_k}\delta x_k \tag{5.6}$$

$$= g(x_k^i, u_k^i) + g_{x,k}^T \delta x_k + g_{u,k}^T \delta u_k + \frac{1}{2}\delta x_k^T g_{xx,k}\delta x_k + \frac{1}{2}\delta u_k^T g_{uu,k}\delta u_k + \delta u_k^T g_{ux,k}\delta x_k \tag{5.7}$$

where we assume $g_{xu,k} = g_{ux,k}^T$.

We then take the second order expansion of V_{k+1}, which is given as

$$V_{k+1} \approx V_{k+1}(x_k^i, u_k^i) + \frac{\partial V_{k+1}}{\partial x_{k+1}}^T \delta x_{k+1} + \frac{1}{2}\delta x_{k+1}^T \frac{\partial^2 V_{k+1}}{\partial x_{k+1}^2}\delta x_{k+1} \tag{5.8}$$

$$= V_{k+1}(x_k^i, u_k^i) + V_{x,k+1}^T \delta x_{k+1} + \frac{1}{2}\delta x_{k+1}^T V_{xx,k+1}\delta x_{k+1}. \tag{5.9}$$

where

$$\delta x_{k+1} = x_{k+1} - x_{k+1}^i \tag{5.10}$$

$$= A_k \delta x_k + B_k \delta u_k. \tag{5.11}$$

We can then write the second order Taylor series expansion of Q_k as

$$Q_k \approx Q_k(x_k^i, u_k^i) + \begin{bmatrix} Q_{x,k} \\ Q_{u,k} \end{bmatrix}^T \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} + \frac{1}{2}\begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix}^T \begin{bmatrix} Q_{xx,k} & Q_{ux_k}^T \\ Q_{ux,k} & Q_{uu,k} \end{bmatrix} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix}. \tag{5.12}$$

For the Q-function, we know

$$\delta u = \arg\min_{\delta u} Q_k. \tag{5.13}$$

Taking the derivative and setting to zero gives

$$0 = Q_u^T + Q_{ux}\delta x + Q_{uu}\delta u. \tag{5.14}$$

The control δu can be given as

$$\delta u^* = -Q_{uu}^{-1}[Q_u^T + Q_{ux}\delta x] \tag{5.15}$$

$$= k + K\delta x. \tag{5.16}$$

Substitute this back into the Bellman equation gives

$$V_k(x_k^i, u_k^i) + V_{x,k}^T \delta x_k + \frac{1}{2}\delta x_k^T V_{xx,k}\delta x_k = Q_k(x_k^i, u_k^i) + \begin{bmatrix} Q_{x,k} \\ Q_{u,k} \end{bmatrix}^T \begin{bmatrix} \delta x_k \\ \delta u_k^* \end{bmatrix} + \frac{1}{2}\begin{bmatrix} \delta x_k \\ \delta u_k^* \end{bmatrix}^T \begin{bmatrix} Q_{xx,k} & Q_{ux_k}^T \\ Q_{ux,k} & Q_{uu,k} \end{bmatrix} \begin{bmatrix} \delta x_k \\ \delta u_k^* \end{bmatrix}. \tag{5.17}$$

This provides us with the backwards pass and allows us to compute $V_{x,k}$,$V_{xx,k}$.

The general algorithm for iterative-LQR is as follows:

Initialize with $k^i = 0$, $K^i = 0$, and a given u_k^i.

Iterate until convergence:

1. Forward Pass
   - Generate x_k^{i+1} and u_k^{i+1} using $x_{k+1}^{i+1} = f_k(x_k, u_k)$ where $u_k^{i+1} = u_k^i + \alpha k_k^i + K_k^i \delta x^i$
2. Backward Pass
   - Linearize the dynamics about x_k^{i+1} and u_k^{i+1} to get A_k^{i+1} and B_k^{i+1}.
   - Use the backwards pass to compute $V_{x,k}$,$V_{xx,k}$ from $V_{x,N} = g_{x,N}$ and $V_{xx,N} = g_{xx,N}$.
   - Use $V_{x,k}$,$V_{xx,k}$ with K^{i+1} and k^{i+1}.

α can be chosen using the Armigo rule.

5.3. Differential Dynamic Programming

Differential dynamic programming (DDP) is identical to DDP with the exception that a second order expansion of the dynamics is used, while the Q function is still truncated to second order.

Bibliography

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

[2] Dimitri Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

[3] Drew Bagnell, Byron Boots, and Sanjiban Choudhury. *Modern Adaptive Control and Reinforcement Learning*. 2022.