ACRL (16-899, S11) Lecture #4 (Thursday- January 20, 2010)

Differential Dynamic Programming and Neighboring Optimal Control

Lecturer: Drew Bagnell *Scribe: Siyuan Feng*

1 LQR recap

In the last lecture, we derived the Kalman gain and Value matrix for LQR

$$\begin{aligned} u_t &= Kx = (R + B^T V_{t+1} B)^{-1}(B^T V_{t+1} A)x \\ V_t &= Q + K^T R K + (A + BK)^T V_{t+1}(A + BK) \end{aligned} \tag{1}$$

2 Affine case and tracking

Imagine we have an excellent pilot flying a RC helicopter, and we record his control inputs and the helicopter states along the way. Then we want to replay the controls on the helicopter and make it do crazy tricks too. Ideally, we could simply play back these controls to get the same states and the problem is solved. However, this absolutely never happens due to random perturbations.

We want the LQR to handle problems that has a nominal x^* and u^* trajectory, thus making the process model affine and cost function having linear terms. We can borrow the Homogeneous coordinates from vision and graphics people to do the transformation so that we don't need to rederive LQR again.

$$\hat{x_{t+1}} = \begin{bmatrix} x_{t+1} \\ 1 \end{bmatrix} = \hat{A} \begin{bmatrix} x_t \\ 1 \end{bmatrix} + \hat{B}\hat{u}, \tag{2}$$

where x_{off} is the offset from nominal, and u^* is the nominal control. Let

$$\begin{aligned} \hat{A} &= \begin{bmatrix} A & x_{off} + Bu_t^* \\ 0 & 1 \end{bmatrix} \\ \hat{B} &= \begin{bmatrix} B \\ 0 \end{bmatrix} \\ \hat{u_t} &= \Delta u_t = u_t - u_t^* \end{aligned} \tag{3}$$

If we expand the equation out, we get

$$x_{t+1} = Ax_t + x_{off} + Bu_t^* + Bu_t - Bu_t^*, \tag{4}$$

which is the linear process with an x offset.

We define the cost function as

$$C = (x_t - x_t^*)^T Q(x_t - x_t^*) + (u_t - u_t^*)^T R(u_t - u_t^*). \tag{5}$$

Let

$$q = x_t^{*T} Q$$
$$\hat{Q} = \begin{bmatrix} Q & q \\ q & \alpha \end{bmatrix} \tag{6}$$

where α doesn't really matter. In practice, α is chosen to be some big number that makes $\hat{Q}$ positive semi definite. Then we can rewrite the cost function as

$$C = \hat{x}^T \hat{Q} \hat{x} + \hat{u}^T R \hat{u} \tag{7}$$

3 iLQR

Given a target trajectory, and a non linear system, we want to solve the tracking problem. The idea is to compute $\hat{A}, \hat{B}, \hat{Q}, \hat{R}$, and call LQR solver to get a optimal policy, execute it, use the resulting states as new linearization points and repeat the process until converge. In most cases, $\hat{Q}$ and $\hat{R}$ are constant, so we only need to compute $\hat{A}$ and $\hat{B}$.

The outline for iLQR looks like:

- 1. Propose trajectory
- 2. Linearize f about trajectory
- 3."Quadratisize" cost function C (compute the second order Taylor expansion of C)
- 4. $\Pi = LQR(\hat{A}, \hat{B}, \hat{Q}, \hat{R})$
- 5. Execute Π, and returns a new trajectory
- 6. Repeat from 1 until converge

During every iteration of iLQR, the new trajectory is used as the linearization points for the next iteration. x^* and u^* are used as the initial linearization points.

This looks a lot like Newton's Method, but it isn't the same, and is in fact better. This algorithm is logically optimal, but it's possible that there could be issues with linearization. If Q or R is indefinite, this becomes bad. It's also possible that they could be negative definite, which would be extremely bad. One fix is to compute a $\tilde{Q} = \lambda I + Q$. Another is to do an eigenvalue decomposition and simply set the negative eigenvalues to zero. Another problem with the algorithm is that it could orbit the optimal point; as a fix, we could somehow limit the step size. Usually, smaller steps are needed at the beginning.