2
LQR: The Analytic MDP

2.1 The Linear Quadratic Regulator

In the previous chapter we defined MDPs and investigated how to compute the value function at any state with Value Iteration. While the examples thus far have involved discrete state and action spaces, important applications of the basic algorithms and theory of MDPs include problems where both states and actions are continuous. Perhaps the simplest such problem is the *Linear Quadratic Regulator* (LQR) problem.

LQR solutions are one of the most effective and widely used methods in robotics and control systems design. The basic problem is to identify a mapping from states to controls that minimizes the quadratic cost of a linear (possibly time invariant) system. A quadratic cost has the form,

$$c(x, u) = x^\top Q\, x + u^\top R\, u, \tag{2.1.1}$$

where $x \in \mathbb{R}^n$ is the state of the system, and $u \in \mathbb{R}^k$ is the control.[1] In the cost function, Q should be symmetric positive semi-definite ($Q = Q^\top, Q \succeq 0$).[2] It does not have to be strictly positive definite in general.[3] For example, in the cartpole problem, we only need the pendulum to stay upright and we do not care much about where the cart is. However, to avoid infinite control effort, R should be strictly positive definite ($R = R^\top, R \succ 0$).

[1] which is precisely the *action a* in the previous section. Here we choose to use u to denote actions in order to be consistent with the broad literature on control.

[2] Why? Note what would fail if Q did not have these properties? Is symmetry a requirement

[3] For instance because sometimes we do not need *every* component of the state to reach 0 and don't care about these components

Exercise

There is confusion in the literature as to what *positive definite* applied to a matrix Q means: does it imply symmetry, or just that $x^\top Q\, x > 0$ for all non-zero x? Let's see the root of this confusion: Why can we consider $Q = Q^\top$ without any loss of generality in the LQR problem? Specifically, make the opposite assumption, and then consider a symmetric Q that would lead to precisely the cost function. In a sense

then, we can simply assume positive definiteness implies symmetry as this is simpler to countenance and will lead to equivalent results.

Continuous Control of a Discrete-Time System

An example of a continuous time-invariant system with quadratic cost is the problem of balancing a simple inverted pendulum. The pendulum is illustrated in Figure 2.1.1. The simple pendulum consists of a bob, modeled as a point mass, and attached to a mass-less rigid rod. Let the mass of the bob be m, the length of the rod be l, and gravity be g. The angle between the pendulum and the y-axis θ is controlled by the torque τ exerted at the origin. The dynamics of this system is given by

$$\begin{aligned} ml^2\ddot{\theta} &= mgl\sin\theta + \tau \\ \implies \ddot{\theta} &= \frac{g}{l}\sin\theta + \frac{1}{ml^2}\tau \\ &\approx \frac{g}{l}\theta + \frac{1}{ml^2}\tau \end{aligned} \tag{2.1.2}$$

To find the control policy of the system, we first linearize it about

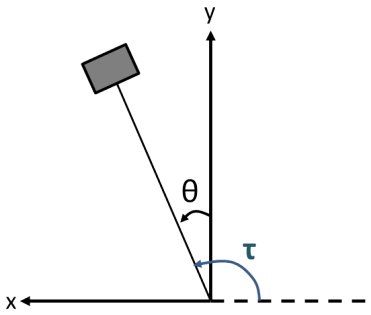

Figure 2.1.1: An inverted pendulum.

the up-right configuration. Let $\alpha = g/l$, and assume $ml^2 = 1$. The state space equations become

$$\begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix}_{t+1} = \begin{bmatrix}1 + \frac{1}{2}\Delta t^2 \cdot \alpha & \Delta t \\ c \cdot \Delta t & 1\end{bmatrix}\begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix}_t + \begin{bmatrix}\frac{1}{2}\Delta t^2 \\ \Delta t\end{bmatrix}\tau \tag{2.1.3}$$

The optimal control policy can be found by formulating an MDP. For the linearized simple pendulum,

- state: $x_t = \begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix}_t$,
- action: $u_t = \tau$,
- cost: $c(x, u) = x^\top Q x + u^\top R u$,

- dynamics: $x_{t+1} = Ax_t + Bu_t$,
  where $A = \begin{bmatrix} 1 + \frac{1}{2}\Delta t^2 \cdot c & \Delta t \\ c \cdot \Delta t & 1 \end{bmatrix}$ and $B = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix}$.

We already know how to solve this problem: Value Iteration! Let's look at this more closely.

2.2 *Value Iteration for Linear Quadratic MDPs*

Let the value function of the MDP for a finite-horizon problem with horizon T be be $J^\pi(x_t, t)$, i.e.

$$J^\pi(x_t, t) = \sum_{t'=t}^{T-1} c(x_{t'}, \pi(x_{t'}, t')). \tag{2.2.1}$$

Recall the **Bellman Equation** for the finite horizon problem:

$$\begin{aligned} J^*(x, t) &= \min_{u_t} \left[c(x_t, u_t) + J^*(x_{t+1}, t+1)\right] \\ &= \min_{u_t} \left[({x_t}^\top Q x_t + {u_t}^\top R u_t) + J^*(x_{t+1}, t+1)\right] \end{aligned} \tag{2.2.2}$$

and

$$J^*(x, T-1) = \min_{u_{T-1}} \left[c(x_{T-1}, u_{T-1})\right] \tag{2.2.3}$$

Let's consider the recursive formulation for solving this problem.

Time T − 1:

At the last time step $t = T - 1$, the solution to Equation 2.2.2 is $u_{T-1} = 0$. This is due to the fact that we are not concerned with the next step since we already reach the time limit. Hence, *any* action will increase the cost: minimizing (2.2.3) is essentially minimizing $u_{T-1}^\top R u_{T-1}$. By definition, R is a positive definite matrix, and therefore setting $u_{T-1} = 0$ can result in minimum cost at $t = T - 1$.

Now let calculate the optimal value function $J^*(x_{T-1}, T-1)$. Since ${u_{T-1}}^\top R u_{T-1} = 0$, by (2.2.3),

$$J^*(x_{T-1}, T-1) = {x_{T-1}}^\top Q x_{T-1} \doteq {x_{T-1}}^\top V_{T-1} x_{T-1}, \tag{2.2.4}$$

where V_{T-1} is the *value matrix*.[4]

[4] In literature, people sometimes use P_t instead of V_t to denote the value matrix.

In summary, at the last time step, we have a zero control and a value that is quadratic in the state.

Time T − 2:

The optimal value function at $t = T - 2$ is,

$$J^*(x_{T-2}, T-2) = \min_{u_{T-2}} c(x_{T-2}, u_{T-2}) + J^*(x_{T-1}, T-1) \tag{2.2.5}$$

$$= \min_{u_{T-2}} \left({x_{T-2}}^\top Q x_{T-2} + {u_{T-2}}^\top R u_{T-2} + {x_{T-1}}^\top V_{T-1} x_{T-1}\right). \tag{2.2.6}$$

For the sake of notational simplicity, let $x = x_{T-2}$ and $u = u_{T-2}$. From the dynamics of the system, $x_{T-1} = Ax + Bu$.

$$J^*(x, T-2) = \min_u \left\{ x^\top Q x + u^\top R u + (Ax + Bu)^\top V_{T-1}(Ax + Bu) \right\} \tag{2.2.7}$$

Taking the partial derivative of the function to be minimized with respect to u and setting it to 0 yields

$$\begin{aligned} 2Ru + 2B^\top V_{T-1} Ax + 2B^\top V_{T-1} Bu &= 0 \\ (R + B^\top V_{T-1} B)u &= -B^\top V_{T-1} Ax \\ u &= -(R + B^\top V_{T-1} B)^{-1} B^\top V_{T-1} Ax \end{aligned} \tag{2.2.8}$$

The solution to u always exists because the inverse of $R + B^\top V_{T-1} B$ exists since R is positive definite and $B^\top V_{T-1} B$ is at least positive semi-definite. Let $K_{T-2} = -(R + B^\top V_{T-1} B)^{-1} B^\top V_{T-1} A$,

$$u_{T-2} = K_{T-2} x_{T-2}. \tag{2.2.9}$$

The control u_{T-2} is a linear function of state x_{T-2} with control matrix K_{T-2}. The optimal value function at $t = T-2$ can be found as

$$\begin{aligned} J^*(x_{T-2}, T-2) &= {x_{T-2}}^\top Q x_{T-2} + {x_{T-2}}^\top K_{T-2}^\top R K_{T-2} x_{T-2} \\ &\quad + {x_{T-2}}^\top (A + BK_{T-2})^\top V_{T-1}(A + BK_{T-2}) x_{T-2} \\ &= {x_{T-2}}^\top (Q + K_{T-2}^\top R K_{T-2} + (A + BK_{T-2})^\top V_{T-1}(A + BK_{T-2})) x_{T-2} \\ &\doteq {x_{T-2}}^\top V_{T-2} x_{T-2}. \end{aligned} \tag{2.2.10}$$

Observe that in this time step, the value is *also* quadratic in state. Therefore, we can derive similar results of linear control and quadratic value for every time step prior to $t = T-2$:

$$\begin{aligned} K_t &= -(R + B^\top V_{t+1} B)^{-1} B^\top V_{t+1} A \\ V_t &= \underbrace{Q}_{\text{current cost}} + \underbrace{K_t^\top R K_t}_{\text{cost of action at t}} + \underbrace{(A + BK_t)^\top V_{t+1}(A + BK_t)}_{\text{cost to go}} \end{aligned} \tag{2.2.11}$$

and the optimal value function is,

$$J^*(x_t, t) = {x_t}^\top V_t x_t. \tag{2.2.12}$$

Algorithm 7 summarizes value iteration for LQRs:

Algorithm `OptimalValue`(A, B, Q, R, t, T)
  if $t = T-1$ **then**
    return Q
  end
  else
    $V_{t+1} =$ `OptimalValue`$(A, B, Q, R, t+1, T)$
    $K_t = -(R + B^\top V_{t+1} B)^{-1} B^\top V_{t+1} A$
    return $V_t = Q + K_t^\top R K_t + (A + BK_t)^\top V_{t+1}(A + BK_t)$
  end

Algorithm 7: LQR value Iteration

The complexity of the above algorithm is a function of the horizon T, the dimensionality of the state space n, and the dimensionality of the action space k: $O(T(n^3 + k^3))$.

Convergence of Value Iteration

Recall that in the finite horizon LQR problem, K_t and V_t are computed backward in time as,

$$\begin{aligned} K_t &= -(R + B^\top V_{t+1} B)^{-1} B^\top V_{t+1} A \\ V_t &= Q + K_t^\top R K_t + (A + BK_t)^\top V_{t+1}(A + BK_t). \end{aligned} \tag{2.2.13}$$

One natural idea is to keep applying (2.2.13) until K_t and V_t converge to a fixed point. The associated question is thus, do K_t and V_t always converge? And further, if they do not always converge, when do they actually converge? The answer is, K_t and V_t converge if the system is so called *stabilizable*,[5] and they converge to the solution to the Discrete Algebraic Ricatti Equation (DARE): [6]

$$\begin{aligned} V &= Q + K^\top R K + (A + BK)^\top V (A + BK) \\ K &= -(R + B^\top V B)^{-1} B^\top V A \end{aligned} \tag{2.2.14}$$

Moreover, the K and V that solve the DARE indeed yield the optimal policy for the infinite horizon LQR problem. We can view V as a combination of the cost of current state and control, along with the future cost. If the system is not stabilizable, for example, a system of two motors controlling two inverted pendulums with one of the motors broken, then K_t and V_t no longer converge. However, the value iteration will still return the policy that can get the system to work as well as possible by using the good motor to attempt to stabilize the system. On the infinite horizon, it may, of course lead to a diverging value estimate– in essence, the issues that happen in finite state spaces with non-converging value functions can happen in solving the Ricatti equation. [7]

[5] Brian D. O. Anderson and John B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall, Inc., 1990

[6] The conditions for LQR to converge are effectively identical to that of any other value iteration problem. It's enough here that we can asymptotically drive all the state variables to 0.

[7] There are linear-algebraic methods to solve the Ricatti equations as well as simply the natural Value-Iteration backup procedure; these can be more computationally efficient, but are rarely required

2.3 *Extensions of LQR*

In the following sections, we continue to expand the domain of applicability of the general strategy for solving LQR problems developed above. The basic techniques that we will augment LQR with include

1. Allowing the system to be time varying
2. Allowing general affine systems (via homogenous coordinates or direct derivation)
3. Moving from controls to "deviations" in control

4. Iteratively re-linearizing

We visit each of these incrementally, as it's useful to see each addition, and end up with a general algorithm for a wide class of control problems.

Tracking Trajectories with LQR

The method described in Algorithm 7 will not work for a pendulum "swing up" problem, since the system dynamics at $\theta = 0^\circ$ (unstable) and $\theta = 180^\circ$ (stable) are *qualitatively* different. Linearization will fail as the linearized model (2.1.3) is a good approximation of the non-linear dynamics only at a small region around $\theta = 0^\circ$.

Given a trajectory, possibly recorded from an expert demonstration, (x_t, u_t) from $\theta = 180^\circ$ to $\theta = 0^\circ$ (see Fig 2.3.1), one might imagine that it could simply be replayed to balance the inverted pendulum. However, this doesn't work in practice due to modeling error– moreover, the same sequence of controls is unlikely to produce exactly the same behavior when played twice on a real system due to minor variations in the system. However, a reference trajectory can still be useful. One way to use an expert trajectory in presence of uncertainty, is to use LQR *tracking*, which we describe below. Be-

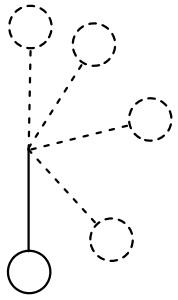

Figure 2.3.1: Solving inverted pendulum swing up using LQR tracking.

fore describing how tracking works, we first introduce several minor variations on the LQR approach, including LQR for *Linear Time Varying* dynamical systems, *Affine Quadratic Regulation*, and LQR with stochastic dynamics.

LQR for Linear Time-Varying Dynamical Systems

Thus far, we have assumed that we were modeling a linear, time-invariant system. As we will see, we might be interested in systems that are linear, but time varying

$$x_{t+1} = A_t x_t + B_t u_t \tag{2.3.1}$$

$$c(x_t, u_t) = {x_t}^\top Q_t x_t + {u_t}^\top R_t u_t \tag{2.3.2}$$

In this case, the LQR equations are simply updated to

$$K_t = -(B_t^\top V_{t+1} B_t + R_t)^{-1} B_t^\top V_{t+1} A_t \tag{2.3.3}$$
$$V_t = Q_t + K_t^\top R_t K_t + (A_t + B_t K_t)^\top V_{t+1}(A_t + B_t K_t) \tag{2.3.4}$$

Affine Quadratic Regulation

Let's now consider a generic *affine* system with time varying dynamics A_t and B_t and a state *offset* x_t^{off}:

$$x_{t+1} = A_t x_t + B_t u_t + x_t^{\text{off}}. \tag{2.3.5}$$

Affine problems can be converted to linear problems by using *homogeneous coordinates*[8]:

[8] https://en.wikipedia.org/wiki/Homogeneous_coordinates

$$\tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{2.3.6}$$
$$\tilde{x}_{t+1} = \begin{bmatrix} A_t & x_t^{\text{off}} \\ 0 & 1 \end{bmatrix} \tilde{x}_t + \begin{bmatrix} B_t \\ 0 \end{bmatrix} u_t \doteq \tilde{A}_t \tilde{x}_t + \tilde{B}_t u_t \tag{2.3.7}$$

This is just a new LQR problem with modified state and dynamics and a new cost defined as $c(\tilde{x}_t, u_t) = \tilde{x}_t^\top \tilde{Q}_t \tilde{x}_t + u_t^\top R_t u_t$, where the choice of $\tilde{Q}$ is problem dependent. We will later see how we can design $\tilde{Q}$ for the tracking problem. The Affine Quadratic Regulation problem can then be solved in exactly the same way as the LQR problem.[9]

[9] Essentially the same trick can be applied to enable us to have linear cost functions terms in the controls as well, but we defer this to the general formulation derived at the end.

Tracking

There are two natural formulations for a tracking cost function:

$$c_t(x_t, u_t) = (x_t - x_t^*)^\top Q (x_t - x_t^*) + (u_t - u_t^*)^\top R (u_t - u_t^*) \tag{2.3.8}$$
$$c_t(x_t, u_t) = (x_t - x_t^*)^\top Q (x_t - x_t^*) + u_t^\top R u_t \tag{2.3.9}$$

where x_t^* and u_t^* are the nominal trajectory and nominal control input obtained from the expert (not necessarily optimal ones!). Q penalizes the deviation from the nominal trajectory and R penalizes either the deviation from the nominal controls or is just a penalty on the control (*e.g.* lots of actuation is bad).

Expanding the term corresponding to state error in the cost function:

$$\begin{aligned}(x_t - x_t^*)^\top Q (x_t - x_t^*) &= x_t^\top Q x_t + \underbrace{x_t^{*\top} Q x_t^*}_{\text{constant at time } t} - \underbrace{2 x_t^{*\top} Q}_{\text{constant at time } t} x_t \\ &= x_t^\top Q x_t + d_t - 2 q_t^\top x_t,\end{aligned}$$

where $d_t \doteq {x_t^*}^\top Q x_t^*$ and $q_t \doteq Q x_t^*$. Next, we choose a $\tilde{Q}_t$ defined as:

$$\tilde{Q}_t = \begin{bmatrix} Q & -q_t \\ -q_t^\top & d_t \end{bmatrix},$$

such that the state error term of the cost function can be formulated as $\tilde{x}_t^\top \tilde{Q}_t \tilde{x}_t$, where $\tilde{x}_t$ is the homogeneous coordinates in (2.3.6). Note that d_t is a constant, which only shifts the cost function in an uninteresting way.

For cost functions with the control error term of the form $(u_t - u_t^*)^\top R(u_t - u_t^*)$, let $\tilde{u}_t = (u_t - u_t^*)$. Then the corresponding term of the cost function can be modified as $\tilde{u}_t^\top R \tilde{u}_t$. In order to use $\tilde{u}_t$ instead of u_t in the cost function defined as in Eq. 2.3.8, the dynamics needs to be modified as follows:

$$\tilde{x}_{t+1} = \begin{bmatrix} A_t & x_t^{\text{off}} + B_t u_t^* \\ 0 & 1 \end{bmatrix} \tilde{x}_t + \begin{bmatrix} B \\ 0 \end{bmatrix} \tilde{u}_t. \tag{2.3.10}$$

The modified cost function is:

$$c_t(\tilde{x}_t, \tilde{u}_t) = \tilde{x}_t^\top \tilde{Q}_t \tilde{x}_t + \tilde{u}_t^\top R \tilde{u}_t. \tag{2.3.11}$$

Solving the LQR for the system using the above cost function

$$\tilde{u}_t = -\tilde{K}_t \tilde{x}_t.$$

Subsequently u_t is obtained as $u_t = \tilde{u}_t + u_t^*$.

2.4 *Iterative LQR (iLQR)*

So far, we have seen how to use LQR to solve problems with linear (or affine) dynamics and quadratic costs. However, real world systems will only rarely be close to linear. [10]

Differential Dynamic Programming (DDP) [11] is a general approach to using quadratic approximations of the value function to solve a broader class of control problems than merely linear-Gaussian. *Iterative LQR* (iLQR) is a simplified variant of DDP, an approach that repeatedly solves LQR (actually affine!) problems to solve for a locally optimal change to a trajectory and a controller around that. The idea of iLQR is very closely related to Newton's method (where we first approximate the objective function to a quadratic function, minimize it, and iterate until convergence). In iLQR, we first approximate the dynamics with an affine model and approximate the cost function with a quadratic function. Crudely speaking, we then solve the LQR problem for the resulting approximate problem, and iterate the process until convergence.

[10] There is a well-known saying among control theorists,

> Classifying systems as linear and nonlinear is like classifying the Universe as bananas and non-bananas.

[11] D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Programming*. Elsevier, 1970

The algorithm

The general iLQR strategy is as follows:

1. Propose some initial (feasible) trajectory $\{x_t, u_t\}_{t=0}^{T-1}$
2. Linearize the dynamics, f about trajectory:

$$\left.\frac{\partial f}{\partial x}\right|_{x_t} = A_t, \quad \left.\frac{\partial f}{\partial u}\right|_{u_t} = B_t$$

   Linearization can be obtained by three methods:

   (a) Analytical: either manually or via *auto-diff*, compute the correct derivatives.

   (b) Numerical: use finite differencing.

   (c) Statistical: Collect data under fixed control sequence, fit linear model.

3. Compute second order Taylor series expansion the cost function $c(x, u)$ around x_t and u_t and get a quadratic approximation $c_t(\tilde{x}_t, \tilde{u}_t) = \tilde{x}_t^\top \tilde{Q}_t \tilde{x}_t + \tilde{u}_t^\top R \tilde{u}_t$ where the $\tilde{x}_t, \tilde{u}_t$ variables represent *changes* in the proposed trajectory in homogenous coordinates. [12]
4. Given $\{A_t, B_t, \tilde{Q}_t, \tilde{R}_t\}_{t=0}^{T-1}$, solve an affine quadratic control problem and obtain the proposed feedback matrices (on the homogeneous represenation of x).
5. Forward simulate the full nonlinear model $f(x, u)$ using the computed controls $\{u_t\}_{t=0}^{T-1}$ that arise from feedback matrices applied to the sequence of states $\{x_t\}_{t=0}^{T-1}$ that arise from that forward simulation.
6. Using the newly obtained $\{x_t, u_t\}_{t=0}^{T-1}$ repeat steps from 2.

[12] We haven't derived using homogeneous coordinates in control; it's essentially equivalent to simply completing the square and finding a "nominal" control, and the appendix to these notes presents the general derivation.

Issues with iLQR

- Q and R can be indefinite when the actual cost function is not convex. Hacks that are typical in the literature include:
  - Projection: $Q = U \underbrace{\Sigma}_{\text{set negative Eigenvalues to 0}} U^\top$. Formally, this can be shown as finding the closest (in L_2 sense cost matrix that actually is PSD.
  - Regularize: Increase the diagonal values until Q becomes positive definite: $Q = Q + \lambda I$
- Trust regions: Sometimes the approximation of the cost function is poor and in such cases its a good idea to restrict the step size (deviation from the trajectory of the previous iteration) while

executing the control. This can be accomplished in the following ways:

- interpolate between the control at current iteration and the previous iterations
- Modify cost to penalize derivation from the trajectory of the previous iteration:

  $\tilde{c} = c + \alpha \cdot (\text{penalty for deviation from the previous trajectory in controls or states})$

  These last known as *control* and *state* damping are extremely common in implementions.

- Some notes:
  LQR recieved significant practical criticism in the 1970s as it was difficult to prevent the resulting synthesized controllers from exciting dynamics that were under-modeled. Without care, LQR (particularly using filtered estimates of the true state, rather than "oracle" access to the true state) will often generate, "stiff", high frequency controls that are not robust. To damp high frequency control from being generated we can do some simple modifications:

  - Penalize *changes* in control from previous control. This is to ensure that the control is smooth. Higher order of smoothness can be obtained by passing the control signal through a filter, modeled in the system dynamics, and then using the output of that as effective control input for the system.
  - More generally, we can implement a *filter* on the execute control dynamics by storing previous controls in the state vector and penalizing any linear operation ont hese.

  It's often useful to model latency by a simple "loading" controls into states by including that delay in the dynamics:

$$\begin{bmatrix} x_t \\ u_{t-1} \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ u_{t-2} \end{bmatrix} + \begin{bmatrix} 0 \\ u_{t-1} \end{bmatrix}. \tag{2.4.1}$$

  This method for modeling delay is crude but effective. More sophisticated approaches include providing an *immutable* region of controls are used used in *receeding* horizon control.

2.5 *Differential Dynamic Programming (DDP)*

The original, fancier version of linearizing value iteration for linear quadratic systems is called differential dynamic programming (DDP).

iLQR and DDP are very similar, the difference being that iLQR assumes a simpler linear model for the system dynamics, while DDP uses a full quadratic model and then truncates any terms that are higher than second order in the value function expansion. The result is that DDP provides a correct-to-second-order expansion of the value function. iLQR is slightly simpler to implement than DDP and often provides similar or better results empirically for less computation.

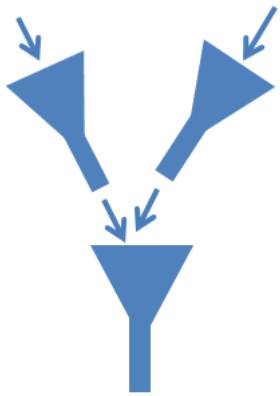

Figure 2.5.1: Funnels can be a metaphor for controllers, and you can think of composing funnels that cover different parts of the space of states.

DDP (or iLQR) builds a second order approximation of the value function, giving a quadratic bowl at every timestep. This ends up acting like a series of funnels [2]. When you are in the area covered by a funnel, you are pulled toward the optimum. You can think of composing funnels such that one funnel dumps you out into another funnel. If you cover the entire space with funnels, then you can imagine that each one is a controller that is good in a certain section of the space (See Figure 2.5.1). With iLQR, we built a quadratic value function about a particular trajectory, but you can imagine starting somewhere else. If you can get from that starting point into the region covered by your value function, then you already know what to do from there. Chris Atkeson wrote a classic paper on this subject, in which he looks at covering the state space with DDP policies [3]. Imagine an inverted pendulum: there will be some controller that is good for the near-vertical case. One can then have other controllers covering other parts of the space, and each controller gets closer to the set of states it knows how to handle, funneling states towards the goal.

LQR with Stochastic Dynamics

The treatment of the Linear Quadratic Control problem up until now has assumed that the dynamics of the system are deterministic:

the next state of the system can be determined precisely from the previous state and the control input.

$$x_{t+1} = Ax_t + Bu_t \tag{2.5.1}$$

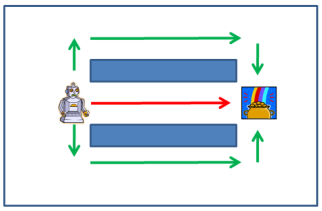

Figure 2.5.2: Robot in grid world showing optimal policy for deterministic (red) vs. stochastic (green) motion.

It is not at all clear, however, that the policy built for the deterministic case is the policy that you would follow if you knew there was noise. For example, imagine a robot in a grid world (See Figure 2.5.2). The robot is positioned on the opposite side of two obstacles from the goal (pot of gold). Hitting an obstacle is catastrophic for the robot, but there is just enough space for the robot to drive between the two obstacles to reach the goal. If the robot's motion is deterministic, then the best policy is to drive between the obstacles. But if the robot's motion is stochastic, and with a 10% probability, the robot moves in a random direction instead of the commanded direction, then the best policy is to avoid walking the tightrope between the dangerous obstacles, and to instead go around the obstacles.

We can extend LQR to handle a simple case of stochastic dynamics and derive the optimal policy for this case. We will assume that at each time step, a zero mean Gaussian perturbation affects the state [13].

$$x_{t+1} = Ax_t + Bu_t + \varepsilon_t \tag{2.5.2}$$

where $\varepsilon_t \sim \mathcal{N}(0, \Sigma)$. x_{t+1} can also be written as

$$x_{t+1} \sim \mathcal{N}(Ax_t + Bu_t, \Sigma) \tag{2.5.3}$$

[13] Note that the noise that we are adding is motion model noise. We are not considering a non-trivial observation model here.

Recall that for the deterministic case, the optimal policy at time t, π_t^*, is given by finding the action that minimizes the sum of action cost and cost-to-go from the resulting state

$$\pi_t^* = \underset{u_t}{\operatorname{argmin}}\, c(x_t, u_t) + J^*(x_{t+1}, t+1) \tag{2.5.4}$$

The problem is that in the stochastic case, the next state x_{t+1} can not be predicted exactly. As with value iteration, the solution is to replace the optimal cost-to-go J^* by the expected value of J^* given the previous state and selected action. The expression for π_t^* thus

becomes

$$\pi_t^* = \underset{u_t}{\operatorname{argmin}}\, c(x_t, u_t) + \mathbb{E}\left[J^*(x_{t+1}, t+1)\right] \tag{2.5.5}$$

The expectation term in this expression is the integral

$$\mathbb{E}\left[J^*(x_{t+1}, t+1)\right] = \int_{\mathbb{X}} x_{t+1}^\top V_{t+1} x_{t+1} \mathcal{N}(x_{t+1}; Ax_t + Bu_t, \Sigma)\, \mathrm{d}x_{t+1} \tag{2.5.6}$$

This integral belongs to a class of integrals called Gaussian Integrals and has a simple closed form solution.

$$\int (x-b)^\top P(x-b)\mathcal{N}(x; \mu, \Sigma)\, \mathrm{d}x = (\mu - b)^\top P(\mu - b) + \operatorname{Tr}\left[P\Sigma\right] \tag{2.5.7}$$

substituting we get

$$\mathbb{E}\left[J^*(x_{t+1}, t+1)\right] = (Ax_t + Bu_t)^\top V_{t+1}(Ax_t + Bu_t) + \operatorname{Tr}\left[V_{t+1}\Sigma\right] \tag{2.5.8}$$

or, since $J^*(x_t, t) = x_t^\top V_t x_t$,

$$\mathbb{E}\left[J^*(x_{t+1}, t+1)\right] = J^*(Ax_t + Bu_t, t+1) + \operatorname{Tr}\left[V_{t+1}\Sigma\right] \tag{2.5.9}$$

Thus using the expectation of the optimal cost-to-go in the stochastic case gives almost the same expression as using the value of cost-to-go in the deterministic case. The only difference is the trace term which is a constant when Σ is fixed or depends only on t. Since all the values under the argmin are shifted by the same constant value, the policy will remain unchanged by the presence of noise, even though the value function has changed. The new trace term added to the cost-to-go can be considered the cost incurred due to uncertainty.

It should be emphasized that this analysis only holds when Σ is independent of the control u. In many real settings, this does not hold. For example, on a robot, the larger the motion the larger the induced uncertainty in position is.

2.6 *Related Reading*

[1] Y Tassa, T Erez, E Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization." IEEE/RSJ International Conference on Intelligent Robotcs and Systems, 2012

[2] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors." International Journal of Robotics Research, 18(6):534–555, June 1999.

[3] C. G. Atkeson, "Using Local Trajectory Optimizers to Speed Up Global Optimization", Proceedings of Neural Information Processing Systems, December 1993.

2.7 *Appendix: Derivation of the General ILQR Backup steps*

The following provides a detailed derivation of the iLQR approach. At each iteration of the algorithm, we execute a proposed current policy to get a trajectory. That we compute the dynamic program below to provide an update to that policy. This is iterated until convergence.

Given the true dynamics F, we can find the Taylor expansion around a proposed trajectory $(x_t\star, u_t\star)$:

$$\begin{aligned} x_{t+1} &= A(x_t - x_t\star) + B(u_t - u_t\star) + F(x_t\star, u_t\star) \\ \Rightarrow x_{t+1} - x_{t+1}\star &= A(x_t - x_t\star) + B(u_t - u_t\star) \\ z_{t+1} &= Az_t + Bv_t \end{aligned}$$

where we define $z_t = x_t - x_t\star$ as the change to the state trajectory and $v_t = u_t - u_t\star$ as the change to the control trajectory.

Similarly, given the true cost function C, the second order taylor expansion is:

$$\begin{aligned} c_t(x_t, u_t) &= \frac{1}{2}[(x_t - x_t\star)^T, (u_t - u_t\star)^T] H \begin{bmatrix}(x_t - x_t\star)\\(u_t - u_t\star)\end{bmatrix} + g^T \begin{bmatrix}(x_t - x_t\star)\\(u_t - u_t\star)\end{bmatrix} + C(x_t\star, u_t\star) \\ &= \frac{1}{2}[(x_t - x_t\star)^T, (u_t - u_t\star)^T] \begin{bmatrix}Q & P\\ P^T & R\end{bmatrix} \begin{bmatrix}(x_t - x_t\star)\\(u_t - u_t\star)\end{bmatrix} + [g_x^T, g_u^T] \begin{bmatrix}(x_t - x_t\star)\\(u_t - u_t\star)\end{bmatrix} + C(x_t\star, u_t\star) \\ &= \frac{1}{2}(x_t - x_t\star)^T Q(x_t - x_t\star) + (x_t - x_t\star)^T P(u_t - u_t\star) + \frac{1}{2}(u_t - u_t\star)^T R(u_t - u_t\star) + g_x^T(x_t - x_t\star) + g_u^T(u_t - u_t\star) + c \end{aligned}$$

and thus that we can right down a cost function in the *changes* to state/action as:

$$\Rightarrow c(z_t, v_t) = \frac{1}{2} z_t^T Q z_t + z_t^T P v_t + \frac{1}{2} v_t^T R v_t + g_x^T z_t + g_u^T v_t + c$$

Dynamic Programming (Value-Iteration) Backup

Assume we have now a control policy of the form of a "feedforward" update term k_t and feedback term K_T that is a linear controller response to "errors" in z_T:

$$v_T = K_T z_T + k_T, \tag{2.7.1}$$

Inductively, we assume the next-state value function (i.e. of the future timestep) can be written in the form,

$$J_{T+1} = \frac{1}{2} z_{T+1} V_{T+1} z_{T+1} + G_{T+1} z_{T+1} + W_{T+1}. \tag{2.7.2}$$

Since

$$\begin{aligned} z_{T+1} &= Az_T + Bv_T && (2.7.3)\\ &= Az_T + B(K_T z_T + k_T) && (2.7.4)\\ &= (A + BK_T)z_T + Bk_T, && (2.7.5) \end{aligned}$$

we can write, J_{T+1} as:

$$J_{T+1} = \frac{1}{2}((A+BK_T)z_T + Bk_T)^T V_{T+1}((A+BK_T)z_T + Bk_T) + G_{T+1}((A+BK_T)z_T + Bk_T) + W_{T+1} \tag{2.7.6}$$

$$= \frac{1}{2}z_T^T(A+BK_T)^T V_{T+1}(A+BK_T)z_T + \frac{1}{2}k_T^T B^T V_{T+1}Bk_T + k_T^T B^T V_{T+1}(A+BK_T)z_T \tag{2.7.7}$$

$$+ G_{T+1}(A+BK_T)z_T + G_{T+1}Bk_T + W_{T+1} \tag{2.7.8}$$

$$= \frac{1}{2}z_T^T(A+BK_T)^T V_{T+1}(A+BK_T)z_T + \left(k_T^T B^T V_{T+1}(A+BK_T) + G_{T+1}(A+BK_T)\right) z_T \tag{2.7.9}$$

$$+ G_{T+1}Bk_T + \frac{1}{2}k_T^T B^T V_{T+1}Bk_T + W_{T+1} \tag{2.7.10}$$

Additionally, we can write the cost $c_T(z_T, v_T)$ as:

$$c_T = \frac{1}{2}z_T^T Q z_T + z_T^T P v_T + \frac{1}{2}v_T^T R v_T + g_x^T z_T + g_u^T v_T + c + J_{T+1} \tag{2.7.11}$$

$$= \frac{1}{2}z_T^T Q z_T + z_T^T P(K_T z_T + k_T) + \frac{1}{2}(K_T z_T + k_T)^T R(K_T z_T + k_T) + g_x^T z_T + g_u^T(K_T z_T + k_T) + c \tag{2.7.12}$$

$$= \frac{1}{2}z_T^T Q z_T + z_T^T P K_T z_T + k_T^T P^T z_T + \frac{1}{2}z_T^T K_T^T R K_T z_T + \frac{1}{2}k_T^T R k_T + k_T^T R K_T z_T + g_x^T z_T \tag{2.7.13}$$

$$+ g_u^T K_T z_T + g_u^T k_T + c$$

$$= \frac{1}{2}z_T^T\left(Q + 2PK_T + K_T^T R K_T\right) z_T + \left(k_T^T P^T + k_T^T R K_T + g_x^T + g_u^T K_T\right) z_T + \frac{1}{2}k_T^T R k_T + g_u^T k_T + c \tag{2.7.14}$$

Then, we can write $J_T = c_T(z_T, v_T) + J_{T+1} = \frac{1}{2}z_T^T V_T z_T + G_T z_T + W_T$ by combining like terms from above, where

$$V_T = Q + 2PK_T + K_T^T R K_T + (A+BK_T)^T V_{T+1}(A+BK_T) \tag{2.7.15}$$

$$G_T = k_T^T P^T + k_T^T R K_T + g_x^T + g_u^T K_T + k_T^T B^T V_{T+1}(A+BK_T) + G_{T+1}(A+BK_T) \tag{2.7.16}$$

$$W_T = \frac{1}{2}k_T^T R k_T + g_u^T k_T + c + G_{T+1}Bk_T + \frac{1}{2}k_T^T B^T V_{T+1}Bk_T + W_{T+1} \tag{2.7.17}$$

We find the control policy by minimizing J_T with respect to v_T.

$$v_T = \min_{v_T} c_T + J_{T+1} \tag{2.7.18}$$

$$= z_T^T P v_T + \frac{1}{2}v_T^T R v_T + g_u^T v_T + \frac{1}{2}(Az_T + Bv_T)^T V_{T+1}(Az_T + Bv_T) + G_{T+1}(Az_T + Bv_T) \tag{2.7.19}$$

$$= \left(z_T^T P + z_T^T A^T V_{T+1}B\right) v_T + (G_{T+1}B + g_u^T)v_T + \frac{1}{2}v_T^T\left(R + B^T V_{T+1}B\right) v_T \tag{2.7.20}$$

$$\tag{2.7.21}$$

Taking the derivative with respect to v_T and setting equal to 0, we get,

$$0 = \left(P^T + B^T V_{T+1}A\right) z_T + (B^T G_{T+1}^T + g_u) + \left(R + B^T V_{T+1}B\right) v_T \tag{2.7.22}$$

$$v_T = -\left(R + B^T V_{T+1}B\right)^{-1}\left(P^T + B^T V_{T+1}A\right) z_T - \left(R + B^T V_{T+1}B\right)^{-1}(B^T G_{T+1}^T + g_u) \tag{2.7.23}$$

$$= K_T z_T + k_T \tag{2.7.24}$$

where $K_T = -\left(R + B^T V_{T+1}B\right)^{-1}\left(P^T + B^T V_{T+1}A\right)$ and $k_T = -\left(R + B^T V_{T+1}B\right)^{-1}(B^T G_{T+1}^T + g_u)$.

Plugging this resulting policy back in to the expression for V_T, G_T and W_T completes the dynamic programming by providing us a quadratic form for the value function. (Note that W_T and c are actually irrelevant as they are constants in the optimization)