Underactuated Robotics

Algorithms for Walking, Running, Swimming, Flying, and Manipulation

Russ Tedrake

© Russ Tedrake, 2024
Last modified 2024-2-22.
How to cite these notes, use annotations, and give feedback.

Note: These are working notes used for a course being taught at MIT. They will be updated throughout the Spring 2024 semester. Lecture videos are available on YouTube.

Previous Chapter | Table of contents | Next Chapter

Launch in Deepnote

Chapter 8
Linear Quadratic Regulators

While solving the dynamic programming problem for continuous systems is very hard in general, there are a few very important special cases where the solutions are very accessible. Most of these involve variants on the case of linear dynamics and convex (e.g. positive quadratic) cost. The simplest case, called the linear quadratic regulator (LQR), is formulated as stabilizing a time-invariant linear system to the origin.

The linear quadratic regulator is likely the most important and influential result in optimal control theory to date. In this chapter we will derive the basic algorithm and a variety of useful extensions.

8.1 Basic Derivation

Consider a linear time-invariant system in state-space form,

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$

with the infinite-horizon cost function given by

$$J = \int_0^\infty \left[\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right] dt, \quad \mathbf{Q} = \mathbf{Q}^T \succeq \mathbf{0}, \mathbf{R} = \mathbf{R}^T \succ 0.$$

Our goal is to find the optimal cost-to-go function $J^*(\mathbf{x})$ which satisfies the HJB:

$$\forall \mathbf{x}, \quad 0 = \min_{\mathbf{u}} \left[\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + \frac{\partial J^*}{\partial \mathbf{x}}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u})\right].$$

There is one important step here -- it is well known that for this problem the optimal cost-to-go function is quadratic. This is easy to verify. Let us choose the form:

$$J^*(\mathbf{x}) = \mathbf{x}^T\mathbf{S}\mathbf{x}, \quad \mathbf{S} = \mathbf{S}^T \succeq 0.$$

The gradient of this function is

$$\frac{\partial J^*}{\partial \mathbf{x}} = 2\mathbf{x}^T\mathbf{S}.$$

Since we have guaranteed, by construction, that the terms inside the min are quadratic and convex (because $\mathbf{R} \succ 0$), we can take the minimum explicitly by finding the solution where the gradient of those terms vanishes:

$$\frac{\partial}{\partial \mathbf{u}} = 2\mathbf{u}^T\mathbf{R} + 2\mathbf{x}^T\mathbf{S}\mathbf{B} = 0.$$

This yields the optimal policy

$$\mathbf{u}^* = \pi^*(\mathbf{x}) = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}\mathbf{x} = -\mathbf{K}\mathbf{x}.$$

Inserting this back into the HJB and simplifying yields

$$0 = \mathbf{x}^T\left[\mathbf{Q} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + 2\mathbf{S}\mathbf{A}\right]\mathbf{x}.$$

All of the terms here are symmetric except for the $2\mathbf{S}\mathbf{A}$, but since $\mathbf{x}^T\mathbf{S}\mathbf{A}\mathbf{x} = \mathbf{x}^T\mathbf{A}^T\mathbf{S}\mathbf{x}$, we can write

$$0 = \mathbf{x}^T\left[\mathbf{Q} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{S}\mathbf{A} + \mathbf{A}^T\mathbf{S}\right]\mathbf{x}.$$

and since this condition must hold for all $\mathbf{x}$, it is sufficient to consider the matrix equation

$$0 = \mathbf{S}\mathbf{A} + \mathbf{A}^T\mathbf{S} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{Q}.$$

This extremely important equation is a version of the *algebraic Riccati equation*. Note that it is quadratic in $\mathbf{S}$, making its solution non-trivial, but it is well known that the equation has a single positive-definite solution if and only if the system is controllable. Moreover, there are good numerical methods for finding that solution, even in high-dimensional problems. Both the optimal policy and optimal cost-to-go function are available from DRAKE by calling `(K,S) = LinearQuadraticRegulator(A,B,Q,R)`.

If the appearance of the quadratic form of the cost-to-go seemed mysterious, consider that the solution to the linear system $\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{BK})\mathbf{x}$ takes the form $\mathbf{x}(t) = e^{(\mathbf{A}-\mathbf{BK})t}\mathbf{x}(0)$, and try inserting this back into the integral cost function. You'll see that the cost takes the form $J = \mathbf{x}^T(0)\mathbf{S}\mathbf{x}(0)$.

It is worth examining the form of the optimal policy more closely. Since the value function represents cost-to-go, the optimal actions will move down this landscape as quickly as possible. Indeed, $-\mathbf{S}\mathbf{x}$ is in the direction of steepest descent of the value function. However, not all directions are possible to achieve in state-space. $-\mathbf{B}^T\mathbf{S}\mathbf{x}$ represents precisely the projection of the steepest descent onto the action space, and is the steepest descent achievable with the control inputs $\mathbf{u}$. Finally, the pre-scaling by the matrix $\mathbf{R}^{-1}$ biases the direction of descent to account for relative cost weightings that we have placed on the different control inputs. Note that although this interpretation is straight-forward, the slope that we are descending (in the value function, $\mathbf{S}$) is a function that takes into account the *long-term* dynamics and cost.

Example 8.1 (LQR for the Double Integrator)

Now we can use LQR to reproduce our HJB example from the previous chapter:

```
import numpy as np
from pydrake.all import LinearQuadraticRegulator

# Define the double integrator's state space matrices.
A = np.array([[0, 1], [0, 0]])
B = np.array([[0], [1]])
Q = np.eye(2)
R = np.eye(1)

(K, S) = LinearQuadraticRegulator(A, B, Q, R)
print("K = " + str(K))
print("S = " + str(S))
```

As in the hand-derived example, our numerical solution returns

$$\mathbf{K} = [1, \sqrt{3}], \qquad \mathbf{S} = \begin{bmatrix} \sqrt{3} & 1 \\ 1 & \sqrt{3} \end{bmatrix}.$$

8.1.1 Local stabilization of nonlinear systems

LQR is extremely relevant to us even though our primary interest is in nonlinear dynamics, because it can provide a local approximation of the optimal control solution for the nonlinear system. Given the nonlinear system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, and a stabilizable operating point, $(\mathbf{x}_0, \mathbf{u}_0)$, with $f(\mathbf{x}_0, \mathbf{u}_0) = 0$. We can define a relative coordinate system

$$\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x}_0, \quad \bar{\mathbf{u}} = \mathbf{u} - \mathbf{u}_0,$$

and observe that

$$\dot{\bar{\mathbf{x}}} = \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}),$$

which we can approximate with a first-order Taylor expansion to

$$\dot{\bar{\mathbf{x}}} \approx f(\mathbf{x}_0, \mathbf{u}_0) + \frac{\partial f(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}_0) + \frac{\partial f(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{u}}(\mathbf{u} - \mathbf{u}_0) = \mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\bar{\mathbf{u}}.$$

Similarly, we can define a quadratic cost function in the error coordinates, or take a (positive-definite) second-order approximation of a nonlinear cost function about the operating point (linear and constant terms in the cost function can be easily incorporated into the derivation by parameterizing a full quadratic form for J^*, as seen in the Linear Quadratic Tracking derivation below).

The resulting controller takes the form $\bar{\mathbf{u}}^* = -\mathbf{K}\bar{\mathbf{x}}$ or

$$\mathbf{u}^* = \mathbf{u}_0 - \mathbf{K}(\mathbf{x} - \mathbf{x}_0).$$

For convenience, DRAKE allows you to call `controller = LinearQuadraticRegulator(system, context, Q, R)` on most dynamical systems (including block diagrams built up of many subsystems); it will perform the linearization for you.

Example 8.2 (LQR for Acrobots, Cart-Poles, and Quadrotors)

LQR provides a very satisfying solution to the canonical "balancing" problem that we've already described for a number of model systems. Here is the notebook with those examples, again:

Launch in Deepnote

Launch in Deepnote

Launch in Deepnote

Launch in Deepnote

I find it very compelling that the same derivation (and effectively identical code) can stabilize such a diversity of systems!

8.2 Finite-horizon formulations

Recall that the cost-to-go for finite-horizon problems is time-dependent, and therefore the HJB sufficiency condition requires an additional term for $\frac{\partial J^*}{\partial t}$.

$$\forall \mathbf{x}, \forall t \in [t_0, t_f], \quad 0 = \min_{\mathbf{u}} \left[\ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) + \frac{\partial J^*}{\partial t} \right].$$

8.2.1 Finite-horizon LQR

Consider systems governed by an LTI state-space equation of the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$

and a finite-horizon cost function, $J = h(\mathbf{x}(t_f)) + \int_0^{t_f} \ell(\mathbf{x}(t), \mathbf{u}(t)) dt$, with

$$h(\mathbf{x}) = \mathbf{x}^T \mathbf{Q}_f \mathbf{x}, \quad \mathbf{Q}_f = \mathbf{Q}_f^T \succeq \mathbf{0}$$
$$\ell(\mathbf{x}, \mathbf{u}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}, \quad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0$$

Writing the HJB, we have

$$0 = \min_{\mathbf{u}} \left[\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} + \frac{\partial J^*}{\partial \mathbf{x}} (\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) + \frac{\partial J^*}{\partial t} \right].$$

Due to the positive definite quadratic form on $\mathbf{u}$, we can find the minimum by setting the gradient to zero:

$$\frac{\partial}{\partial \mathbf{u}} = 2\mathbf{u}^T\mathbf{R} + \frac{\partial J^*}{\partial \mathbf{x}}\mathbf{B} = 0$$
$$\mathbf{u}^* = \pi^*(\mathbf{x}, t) = -\frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^T\frac{\partial J^*}{\partial \mathbf{x}}^T$$

In order to proceed, we need to investigate a particular form for the cost-to-go function, $J^*(\mathbf{x}, t)$. Let's try a solution of the form:

$$J^*(\mathbf{x}, t) = \mathbf{x}^T\mathbf{S}(t)\mathbf{x}, \quad \mathbf{S}(t) = \mathbf{S}^T(t) \succ \mathbf{0}.$$

In this case we have

$$\frac{\partial J^*}{\partial \mathbf{x}} = 2\mathbf{x}^T\mathbf{S}(t), \quad \frac{\partial J*}{\partial t} = \mathbf{x}^T\dot{\mathbf{S}}(t)\mathbf{x},$$

and therefore

$$\mathbf{u}^* = \pi^*(\mathbf{x}, t) = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t)\mathbf{x}$$
$$0 = \mathbf{x}^T\left[\mathbf{Q} - \mathbf{S}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t) + \mathbf{S}(t)\mathbf{A} + \mathbf{A}^T\mathbf{S}(t) + \dot{\mathbf{S}}(t)\right]\mathbf{x}.$$

Therefore, $\mathbf{S}(t)$ must satisfy the condition (known as the continuous-time *differential Riccati equation*):

$$-\dot{\mathbf{S}}(t) = \mathbf{S}(t)\mathbf{A} + \mathbf{A}^T\mathbf{S}(t) - \mathbf{S}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t) + \mathbf{Q},$$

and the terminal condition

$$\mathbf{S}(t_f) = \mathbf{Q}_f.$$

Since we were able to satisfy the HJB with the minimizing policy, we have met the sufficiency condition, and have found the optimal policy and optimal cost-to-go function.

Note that the infinite-horizon LQR solution described in the prequel is exactly the steady-state solution of this equation, defined by $\dot{\mathbf{S}}(t) = 0$. Indeed, for controllable systems this equation is stable (backwards in time), and as expected the finite-horizon solution converges on the infinite-horizon solution as the horizon time limits to infinity.

The solution $\mathbf{S}(t)$ will be the result of numerical integration. On weakly controllable systems (like the perching airplane), I've found that even error-controlled integration can lead to crippling numerical artifacts such as the loss of symmetry or positive definiteness. For a more robust numerical solution, we can instead integrate a factorization of $\mathbf{S}$ that ensures these properties are maintained. Taking $\mathbf{S}(t) = \mathbf{P}(t)\mathbf{P}^T(t)$, and again using the fact that $\mathbf{x}^T\mathbf{P}\dot{\mathbf{P}}^T\mathbf{x} = \mathbf{x}^T\dot{\mathbf{P}}\mathbf{P}^T\mathbf{x}$, we can write the "square-root form" of the Riccati differential equation:

$$-\dot{\mathbf{P}}(t) = \mathbf{A}^T\mathbf{P}(t) - \frac{1}{2}\mathbf{S}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}(t) + \frac{1}{2}\mathbf{Q}\mathbf{P}^{-T}(t), \quad \mathbf{P}(t_f) = \mathbf{Q}_f^{\frac{1}{2}}.$$

This form does require that $\mathbf{P}(t)$ is invertible, which in turn requires that $\mathbf{Q}_f \succ 0$.

8.2.2 Time-varying LQR

The derivation above holds even if the dynamics are given by

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(\mathbf{t})\mathbf{u}.$$

Similarly, the cost functions $\mathbf{Q}$ and $\mathbf{R}$ can also be time-varying. This is quite surprising, as the class of time-varying linear systems is a quite general class of systems. It requires essentially no assumptions on how the time-dependence enters, except perhaps that if $\mathbf{A}$ or $\mathbf{B}$ is discontinuous in time then one would have to use the proper techniques to accurately integrate the differential equation.

8.2.3 Local trajectory stabilization for nonlinear systems

One of the most powerful applications of time-varying LQR involves linearizing around a nominal trajectory of a nonlinear system and using LQR to provide a trajectory controller. This will tie in very nicely with the algorithms we develop in the chapter on trajectory optimization.

Let us assume that we have a nominal trajectory, $\mathbf{x}_0(t), \mathbf{u}_0(t)$ defined for $t \in [t_1, t_2]$. Similar to the time-invariant analysis, we begin by defining a local coordinate system relative to the trajectory:

$$\bar{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}_0(t), \quad \bar{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}_0(t).$$

Now we have

$$\dot{\bar{\mathbf{x}}} = \dot{\mathbf{x}} - \dot{\mathbf{x}}_0 = f(\mathbf{x}, \mathbf{u}) - f(\mathbf{x}_0, \mathbf{u}_0),$$

which we can again approximate with a first-order Taylor expansion to

$$\dot{\bar{\mathbf{x}}} \approx f(\mathbf{x}_0, \mathbf{u}_0) + \frac{\partial f(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}_0) + \frac{\partial f(\mathbf{x}_0, \mathbf{u}_0)}{\partial \mathbf{u}}(\mathbf{u} - \mathbf{u}_0) - f(\mathbf{x}_0, \mathbf{u}_0) = \mathbf{A}(t)\bar{\mathbf{x}} + \mathbf{B}(t)\bar{\mathbf{u}}.$$

This is very similar to using LQR to stabilize a fixed-point, but with some important differences. First, the linearization is time-varying. Second, our linearization is valid for any state along a feasible trajectory (not just fixed-points), because the coordinate system is moving along with the trajectory.

Similarly, we can define a quadratic cost function in the error coordinates, or take a (positive-definite) second-order approximation of a nonlinear cost function along the trajectory (linear and constant terms in the cost function can be easily incorporated into the derivation by parameterizing a full quadratic form for J^*, as seen in the Linear Quadratic Tracking derivation below).

The resulting controller takes the form $\bar{\mathbf{u}}^* = -\mathbf{K}(t)\bar{\mathbf{x}}$ or

$$\mathbf{u}^* = \mathbf{u}_0(t) - \mathbf{K}(t)(\mathbf{x} - \mathbf{x}_0(t)).$$

DRAKE provides a `FiniteHorizonLinearQuadraticRegulator` method; if you pass it a nonlinear systems it will perform the linearization in the proper coordinates for you using automatic differentiation.

Remember that stability is a statement about what happens as time goes to infinity. In order to talk about stabilizing a trajectory, the trajectory must be defined for all $t \in [t_1, \infty)$. This can be accomplished by considering a finite-time trajectory which terminates at a stabilizable fixed-point at a time $t_2 \geq t_1$, and remains at the fixed point for $t \geq t_2$. In this case, the finite-horizon Riccati equation is initialized with the infinite-horizon LQR solution: $S(t_2) = S_\infty$, and solved backwards in time from t_2 to t_1 for the remainder of the trajectory. And *now* we can say that we have *stabilized* the trajectory!

8.2.4 Linear Quadratic Optimal Tracking

For completeness, we consider a slightly more general form of the linear quadratic regulator. The standard LQR derivation attempts to drive the system to zero. Consider now the problem:

$$\begin{gathered}\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ h(\mathbf{x}) = (\mathbf{x} - \mathbf{x}_d(t_f))^T \mathbf{Q}_f (\mathbf{x} - \mathbf{x}_d(t_f)), \quad \mathbf{Q}_f = \mathbf{Q}_f^T \succeq 0 \\ \ell(\mathbf{x}, \mathbf{u}, t) = (\mathbf{x} - \mathbf{x}_d(t))^T \mathbf{Q} (\mathbf{x} - \mathbf{x}_d(t)) + (\mathbf{u} - \mathbf{u}_d(t))^T \mathbf{R} (\mathbf{u} - \mathbf{u}_d(t)), \\ \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0\end{gathered}$$

Now, guess a solution

$$J^*(\mathbf{x},t) = \mathbf{x}^T\mathbf{S}_{xx}(t)\mathbf{x} + 2\mathbf{x}^T\mathbf{s}_x(t) + s_0(t), \quad \mathbf{S}_{xx}(t) = \mathbf{S}_{xx}^T(t) \succ \mathbf{0}.$$

In this case, we have

$$\frac{\partial J^*}{\partial \mathbf{x}} = 2\mathbf{x}^T\mathbf{S}_{xx}(t) + 2\mathbf{s}_x^T(t), \quad \frac{\partial J^*}{\partial t} = \mathbf{x}^T\dot{\mathbf{S}}_{xx}(t)\mathbf{x} + 2\mathbf{x}^T\dot{\mathbf{s}}_x(t) + \dot{s}_0(t).$$

Using the HJB,

$$0 = \min_{\mathbf{u}} \left[(\mathbf{x}-\mathbf{x}_d(t))^T\mathbf{Q}(\mathbf{x}-\mathbf{x}_d(t)) + (\mathbf{u}-\mathbf{u}_d(t))^T\mathbf{R}(\mathbf{u}-\mathbf{u}_d(t)) + \frac{\partial J^*}{\partial \mathbf{x}}(\mathbf{A}\mathbf{x}+\mathbf{B}\mathbf{u}) + \frac{\partial J^*}{\partial t} \right],$$

we have

$$\begin{aligned} \frac{\partial}{\partial \mathbf{u}} &= 2(\mathbf{u}-\mathbf{u}_d(t))^T\mathbf{R} + (2\mathbf{x}^T\mathbf{S}_{xx}(t) + 2\mathbf{s}_x^T(t))\mathbf{B} = 0, \\ \mathbf{u}^*(t) &= \mathbf{u}_d(t) - \mathbf{R}^{-1}\mathbf{B}^T\left[\mathbf{S}_{xx}(t)\mathbf{x} + \mathbf{s}_x(t)\right] \end{aligned}$$

The HJB can be satisfied by integrating backwards

$$\begin{aligned} -\dot{\mathbf{S}}_{xx}(t) =& \mathbf{Q} - \mathbf{S}_{xx}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}_{xx}(t) + \mathbf{S}_{xx}(t)\mathbf{A} + \mathbf{A}^T\mathbf{S}_{xx}(t) \\ -\dot{\mathbf{s}}_x(t) =& -\mathbf{Q}\mathbf{x}_d(t) + \left[\mathbf{A}^T - \mathbf{S}_{xx}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\right]\mathbf{s}_x(t) + \mathbf{S}_{xx}(t)\mathbf{B}\mathbf{u}_d(t) \\ -\dot{s}_0(t) =& \mathbf{x}_d(t)^T\mathbf{Q}\mathbf{x}_d(t) - \mathbf{s}_x^T(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{s}_x(t) + 2\mathbf{s}_x(t)^T\mathbf{B}\mathbf{u}_d(t), \end{aligned}$$

from the final conditions

$$\begin{aligned} \mathbf{S}_{xx}(t_f) =& \mathbf{Q}_f \\ \mathbf{s}_x(t_f) =& -\mathbf{Q}_f\mathbf{x}_d(t_f) \\ s_0(t_f) =& \mathbf{x}_d^T(t_f)\mathbf{Q}_f\mathbf{x}_d(t_f). \end{aligned}$$

Notice that the solution for $\mathbf{S}_{xx}$ is the same as the simpler LQR derivation, and is symmetric (as we assumed). Note also that $s_0(t)$ has no effect on control (even indirectly), and so can often be ignored.

A quick observation about the quadratic form, which might be helpful in debugging. We know that $J(\mathbf{x},t)$ must be uniformly positive. This is true iff $\mathbf{S}_{xx} \succ 0$ and $s_0 > \mathbf{s}_x^T\mathbf{S}_{xx}^{-1}\mathbf{s}_x$, which comes from evaluating the function at $\mathbf{x}_{min}(t)$ defined by $\left[\frac{\partial J^*(\mathbf{x},t)}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}_{min}(t)} = 0$.

8.2.5 Linear Final Boundary Value Problems

The finite-horizon LQR formulation can be used to impose a strict final boundary value condition by setting an infinite $\mathbf{Q}_f$. However, integrating the Riccati equation backwards from an infinite initial condition isn't very practical. To get around this, let us consider solving for $\mathbf{P}(t) = \mathbf{S}(t)^{-1}$. Using the matrix relation $\frac{d\mathbf{S}^{-1}}{dt} = -\mathbf{S}^{-1}\frac{d\mathbf{S}}{dt}\mathbf{S}^{-1}$, we have:

$$-\dot{\mathbf{P}}(t) = -\mathbf{P}(t)\mathbf{Q}\mathbf{P}(t) + \mathbf{B}\mathbf{R}^{-1}\mathbf{B} - \mathbf{A}\mathbf{P}(t) - \mathbf{P}(t)\mathbf{A}^T,$$

with the final conditions

$$\mathbf{P}(t_f) = 0.$$

This Riccati equation can be integrated backwards in time for a solution.

It is very interesting, and powerful, to note that, if one chooses $\mathbf{Q} = 0$, therefore imposing no position cost on the trajectory before time T, then this inverse Riccati equation becomes a linear ODE which can be solved explicitly. These relationships are used in the derivation of the controllability Grammian, but here we use them to design a feedback controller.

8.3 Variations and Extensions

8.3.1 Discrete-time Riccati Equations

Essentially all of the results above have a natural correlate for discrete-time systems. What's more, the discrete time versions tend to be simpler to think about in the model-predictive control (MPC) setting that we'll be discussing below and in the next chapters.

Consider the discrete time dynamics:

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$

and we wish to minimize

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0.$$

The cost-to-go is given by

$$J(\mathbf{x}, n-1) = \min_{\mathbf{u}} \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + J(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, n).$$

If we once again take

$$J(\mathbf{x}, n) = \mathbf{x}^T\mathbf{S}[\mathbf{n}]\mathbf{x}, \quad \mathbf{S}[n] = \mathbf{S}^T[n] \succ 0,$$

then we have

$$\mathbf{u}^*[n] = -\mathbf{K}[\mathbf{n}]\mathbf{x}[n] = -(\mathbf{R} + \mathbf{B}^T\mathbf{S}[n]\mathbf{B})^{-1}\mathbf{B}^T\mathbf{S}[n]\mathbf{A}\mathbf{x}[n],$$

yielding

$$\mathbf{S}[n-1] = \mathbf{Q} + \mathbf{A}^T\mathbf{S}[n]\mathbf{A} - (\mathbf{A}^T\mathbf{S}[n]\mathbf{B})(\mathbf{R} + \mathbf{B}^T\mathbf{S}[n]\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{S}[n]\mathbf{A}), \quad \mathbf{S}[N] = 0,$$

which is the famous *Riccati difference equation*. The infinite-horizon LQR solution is given by the (positive-definite) fixed-point of this equation:

$$\mathbf{S} = \mathbf{Q} + \mathbf{A}^T\mathbf{S}\mathbf{A} - (\mathbf{A}^T\mathbf{S}\mathbf{B})(\mathbf{R} + \mathbf{B}^T\mathbf{S}\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{S}\mathbf{A}).$$

Like in the continuous time case, this equation is so important that we have special numerical recipes for solving this discrete-time algebraic Riccati equation (DARE). DRAKE delegates to these numerical methods automatically when you evaluate the `LinearQuadraticRegulator` method on a system that has only discrete state and a single periodic time step.

Example 8.3 (Discrete-time vs Continuous-time LQR)

You can explore the relationship between the discrete-time and continuous-time formulations in this notebook:

Launch in Deepnote

In reinforcement learning, it is popular to consider the infinite-horizon "discounted" cost: $\min \sum_{n=0}^{\infty} \gamma^n(\mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n])$. The optimal controller is

$$\mathbf{u}^* = -\gamma(\mathbf{R} + \gamma\mathbf{B}^T\mathbf{S}\mathbf{B})^{-1}\mathbf{B}^T\mathbf{S}\mathbf{A}\mathbf{x}.$$

and the corresponding Riccati equation is

$$\mathbf{S} = \mathbf{Q} + \gamma\mathbf{A}^T\mathbf{S}\mathbf{A} - \gamma^2(\mathbf{A}^T\mathbf{S}\mathbf{B})(\mathbf{R} + \gamma\mathbf{B}^T\mathbf{S}\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{S}\mathbf{A}),$$

whose solution can be found using `DiscreteAlgebraicRiccatiEquation`$(\sqrt{\gamma}\mathbf{A}, \mathbf{B}, \mathbf{Q}, \frac{1}{\gamma}\mathbf{R})$.

Example 8.4 (LQR via Fitted Value Iteration)

In the dynamic programming chapter, we developed general purpose tools for approximating value functions with function approximators. I think it is particularly illustrative to see how these work out for LQR. Let's consider the discrete-time, infinite-horizon, discounted case.

For LQR, we know that the optimal value function will take a quadratic form, $\mathbf{x}^T\mathbf{S}\mathbf{x}$. Although it is quadratic in $\mathbf{x}$, this form is linear in the parameters, $\mathbf{S}$. We can leverage some of the special tools for fitted value iteration with a *linear function approximator*.

Launch in Deepnote

I've included two versions of the value iteration update in the notebook -- one that samples over both $\mathbf{x}$ and $\mathbf{u}$, and one that samples only over $\mathbf{x}$ and uses the LQR policy (given the current estimated cost-to-go) to determine $\mathbf{u}$. The difference is not subtle when $\gamma \to 1$. Take a look!

We must remember that there are multiple solutions to the Riccati equation -- the solution to the LQR problem is the (unique) positive definite solution. But there are non-positive definite solutions, too, which lead to unstable controllers -- these solutions do achieve zero Bellman residual. Every solution to the Riccati equation is a fixed point of the (fitted) value iteration update, but only the positive-definite solution is a stable fixed point of the algorithm.

To see this, write $\hat{J}(\mathbf{x}) = \mathbf{x}^T(\mathbf{S}^* + \mathbf{\Delta})\mathbf{x}$, where $\mathbf{S}^*$ is any solution to the Riccati equation, and $\mathbf{\Delta}$ is a small deviation matrix. If we write the error dynamics through the fitted value iteration update, we can see that

$$\mathbf{\Delta}_{i+1} = (\mathbf{A} - \mathbf{B}\mathbf{K}_i)^T\mathbf{\Delta}_i(\mathbf{A} - \mathbf{B}\mathbf{K}_i),$$

where $\mathbf{K}_i$ is the optimal controller given $\mathbf{S}_i = \mathbf{S}^* + \mathbf{\Delta}_i \approx \mathbf{K}^*$. This error converges to zero if and only if $(\mathbf{A} - \mathbf{B}\mathbf{K}^*)$ is stable, which is only if $\mathbf{S}^* \succ 0$.

Go ahead and try many different initial $\mathbf{S}$ in the code to verify it for yourself.

8.3.2 LQR with input and state constraints

A natural extension for linear optimal control is the consideration of strict constraints on the inputs or state trajectory. Most common are linear inequality constraints, such as $\forall n, |\mathbf{u}[n]| \leq 1$ or $\forall n, \mathbf{x}[n] \geq -2$ (any linear constraints of the form $\mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \leq \mathbf{e}$ can be solved with the same tools). Much is known about the solution to this problem in the discrete-time case, but it's computation is signficantly harder than the unconstrained case. Almost always, we give up on solving for the best control policy in closed form, and instead solve for the optimal control

trajectory $\mathbf{u}[\cdot]$ from a particular initial condition $\mathbf{x}[0]$ over some finite horizon. Fortunately, this problem is a convex optimization and we can often solve it quickly and reliably enough to solve it at every time step, effectively turning a motion planning algorithm into a feedback controller; this idea is famously known as model-predictive control (MPC). We will provide the details in the trajectory optimization chapter.

8.3.3 LQR on a manifold

One important case that does have closed-form solutions is LQR with linear *equality* constraints. This is particularly relevant in the case of stabilizing robots with kinematic constraints such as a closed kinematic chain, which appears in four-bar linkages or even for the linearization of a biped robot with two feet on the ground.

Our formulation is as follows: in addition to the linear dynamics (in continuous time or discrete time), we have a linear equality constraint of the form $\mathbf{F}\mathbf{x} = 0$, with $\mathbf{F} \in \mathbb{R}^{(n-d)\times n}$. Form a matrix $\mathbf{P} \in \mathbb{R}^{d\times n}$ that forms an orthonormal basis for the nullspace of $\mathbf{F}$, such that

$$\mathbf{P}\mathbf{P}^T = \mathbf{I}^{d\times d}, \quad \mathbf{P}\mathbf{F}^T = \mathbf{0}^{d\times(n-d)}.$$

Then for any $\mathbf{x}$ we can write $\mathbf{x} = \mathbf{P}^T\mathbf{y} + \mathbf{F}^T\mathbf{z}$, form some $\mathbf{y} \in \mathbb{R}^d$ and $\mathbf{z} \in \mathbb{R}^{n-d}$. However, as the result of the constraints we know that $\mathbf{F}\mathbf{x} = \mathbf{z} = 0$. Instead of stabilizing $\mathbf{x}$ via LQR, we will stabilize $\mathbf{y}$, using the projected cost and dynamics given by

$$\mathbf{A}_\mathbf{y} = \mathbf{P}\mathbf{A}_\mathbf{x}\mathbf{P}^T, \qquad \mathbf{B}_\mathbf{y} = \mathbf{P}\mathbf{B}_\mathbf{x}, \qquad \mathbf{Q}_\mathbf{y} = \mathbf{P}\mathbf{Q}_\mathbf{x}\mathbf{P}^T$$

where $\mathbf{A}_\mathbf{x}, \mathbf{B}_\mathbf{x}$, and $\mathbf{Q}_\mathbf{x}$ are the matrices from the full coordinates model. I've chosen to write it this way because it is the same in both discrete and continuous time. The resulting cost-to-go is given by $\mathbf{S}_\mathbf{x} = \mathbf{P}^T\mathbf{S}_y\mathbf{P}$ and the controller is given by $\mathbf{K}_\mathbf{x} = \mathbf{K}_\mathbf{y}\mathbf{P}$, and the where $\mathbf{K}_y, \mathbf{S}_y$ are the solutions to the reduced LQR problem in $\mathbf{y}$.

More generally, for a nonlinear system with nonlinear equality constraints, we would linearize both the system dynamics and the constraint, and then apply the derivation above. [1, IIIb] provides the time-varying LQR version, which follows naturally.

Example 8.5 (Balancing a Segway)

A nice example of this is the task of balancing a Segway in the $x-z$ plane. In 3D, one could either consider the nonholonomic wheels of the Segway, or turn it into a Ballbot. For the purposes of simulation, one can simply construct the ground, the ball, and the bot with appropriate collision geometry, and drop it into `MultibodyPlant`. The frictional contact between the ball and the ground generates the expected dynamics.

Designing the LQR controller is another story. We will use a `PlanarJoint` for the floating base, giving us 4 degrees of freedom: $[x, z, \theta_{ball}, \theta_{bot-ball}]$. If we linearize this model and call LQR, the LQR call will fail because *the system is not controllable in the full coordinates*. The first reason is easy to see: there is nothing that the actuators can do to change z, the vertical position of the ball. Perhaps the more subtle reason is that there is a (holonomic) constraint from the ball rolling without slip on the ground: $x = -r\theta_{ball}$, where r is the radius of the ball.

I will use one small implementation "trick". In my code, rather than deal with finding the resting penetration of the ball, I decided to remove the vertical degree of freedom completely, and used a prismatic joint for x and a revolute joint for θ_{ball} instead of the `PlanarJoint` for the floating base. I configured it so that the ball was sufficiently in penetration that the normal forces from the ground were substantial enough that the friction cone could support the horizontal frictional forces I needed to roll. This effectively implements the z constraint by construction (completely removing that degree of freedom).

To use LQR, we will write the constraints $x = -r\theta_{ball}$ and the implied $\dot{x} = -r\dot{\theta}_{ball}$ as

$$\mathbf{F}\mathbf{x} = \begin{bmatrix} 1 & r & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & r & 0 \end{bmatrix} \mathbf{x} = 0$$

The nullspace matrix $\mathbf{P}$ can be obtained numerically, but here it was simple enough to get from inspection,

$$\mathbf{P} = \begin{bmatrix} \frac{r}{\sqrt{1+r^2}} & -\frac{1}{\sqrt{1+r^2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{r}{\sqrt{1+r^2}} & -\frac{1}{\sqrt{1+r^2}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

We can obtain the matrices $\mathbf{A}_x \in \mathbb{R}^{6\times 6}, \mathbf{B}_x \in \mathbb{R}^{6\times 1}$ by linearizing the `MultibodyPlant` dynamics. Then we will call LQR with the projected matrices. Note that the we will use the discrete-time dynamics mode of the `MultibodyPlant` (by passing in a non-zero `time_step` to the constructor), because discrete-time dynamics engine contains our most advanced and robust algorithms for simulating contact.

Importantly, although we design the LQR controller using the reduced model, we can run this controller directly on the original model.

Launch in Deepnote

8.3.4 LQR for linear systems in implicit form

Consider a linear system given by the state-space equations,

$$\mathbf{E}\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$

where $\mathbf{E}$ may or may not be invertible. We consider this model to be in "implicit form", and will discuss much more about the use of implicit forms, and their strong connections to mechanical systems, in the next chapter. Under stabilizability and detectability conditions[2], the solution for the standard infinite-horizon LQR cost function is given by

$$J^*(\mathbf{x}) = \mathbf{x}^T\mathbf{E}^T\mathbf{S}\mathbf{E}\mathbf{x}, \qquad \mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}\mathbf{E}\mathbf{x},$$

where $\mathbf{S}$ is the positive-definite solution to a generalized Riccati equation:

$$\mathbf{E}^T\mathbf{S}\mathbf{A} + \mathbf{A}\mathbf{S}\mathbf{E}^T - \mathbf{E}^T\mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}\mathbf{E} + \mathbf{Q} = 0.$$

The discrete-time form follows similarly.

8.3.5 LQR as a convex optimization

One can also design the LQR gains using linear matrix inequalities (LMIs). I will defer the derivation til we cover the policy gradient view of LQR, because the LMI formulation is based on a change of variables from the basic policy evaluation criterion. If you want to look ahead, you can find that formulation here.

Solving the algebraic Riccati equation is still the preferred way of computing the LQR solution. But it is helpful to know that one could also compute it with convex optimization. In addition to deepening our understanding, this can be useful for generalizing the basic LQR solution (e.g. for robust stabilization) or to solve for the LQR gains jointly as part of a bigger optimization.

8.3.6 Finite-horizon LQR via least squares

We can also obtain the solution to the discrete-time finite-horizon (including the time-varying or tracking variants) LQR problem using optimization -- in this case it actually reduces to a simple least-squares problem. The presentation in this section can be viewed as a simple implementation of the Youla parameterization (sometimes called "Q-parameterization") from controls. Small variations on the formulation here play an important role in the minimax variants of LQR (which

optimize a worst-case performance), which we will discuss in the robust control chapter (e.g. [3, 4]).

First, let us appreciate that the default parameterization is not convex. Given

$$\min \sum_{n=0}^{N-1} \mathbf{x}^T[n]\mathbf{Q}\mathbf{x}[n] + \mathbf{u}^T[n]\mathbf{R}\mathbf{u}[n], \qquad \mathbf{Q} = \mathbf{Q}^T \succeq 0, \mathbf{R} = \mathbf{R}^T \succ 0$$
$$\text{subject to} \quad \mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n],$$
$$\mathbf{x}[0] = \mathbf{x}_0$$

if we wish to search over controllers of the form

$$\mathbf{u}[n] = \mathbf{K}_n \mathbf{x}[n],$$

then we have

$$\begin{aligned}
\mathbf{x}[1] &= \mathbf{A}\mathbf{x}_0 + \mathbf{B}\mathbf{K}_0\mathbf{x}_0, \\
\mathbf{x}[2] &= \mathbf{A}(\mathbf{A} + \mathbf{B}\mathbf{K}_0)\mathbf{x}_0 + \mathbf{B}\mathbf{K}_1(\mathbf{A} + \mathbf{B}\mathbf{K}_0)\mathbf{x}_0 \\
\mathbf{x}[n] &= \left(\prod_{i=0}^{n-1}(\mathbf{A} + \mathbf{B}\mathbf{K}_i)\right)\mathbf{x}_0
\end{aligned}$$

As you can see, the $\mathbf{x}[n]$ terms in the cost function include our decision variables multiplied together -- resulting in a non-convex objective. The trick is to re-parameterize the decision variables, and write the feedback in the form:

$$\mathbf{u}[n] = \tilde{\mathbf{K}}_n \mathbf{x}_0,$$

leading to

$$\begin{aligned}
\mathbf{x}[1] &= \mathbf{A}\mathbf{x}_0 + \mathbf{B}\tilde{\mathbf{K}}_0\mathbf{x}_0, \\
\mathbf{x}[2] &= \mathbf{A}(\mathbf{A} + \mathbf{B}\tilde{\mathbf{K}}_0)\mathbf{x}_0 + \mathbf{B}\tilde{\mathbf{K}}_1\mathbf{x}_0 \\
\mathbf{x}[n] &= \left(\mathbf{A}^n + \sum_{i=0}^{n-1}\mathbf{A}^{n-i-1}\mathbf{B}\tilde{\mathbf{K}}_i\right)\mathbf{x}_0
\end{aligned}$$

Now all of the decision variables, $\tilde{\mathbf{K}}_i$, appear linearly in the solution to $\mathbf{x}[n]$ and therefore (convex) quadratically in the objective.

We still have an objective function that depends on $\mathbf{x}_0$, but we would like to find the optimal $\tilde{\mathbf{K}}_i$ *for all* $\mathbf{x}_0$. To achieve this let us evaluate the optimality conditions of this least squares problem, starting by taking the gradient of the objective with respect to $\tilde{\mathbf{K}}_i$, which is:

$$\mathbf{x}_0\mathbf{x}_0^T\left(\tilde{\mathbf{K}}_i^T\left(\mathbf{R} + \sum_{m=i+1}^{N-1}\mathbf{B}^T(\mathbf{A}^{m-i-1})^T\mathbf{Q}\mathbf{A}^{m-i-1}\mathbf{B}\right) + \sum_{m=i+1}^{N-1}(\mathbf{A}^m)^T\mathbf{Q}\mathbf{A}^{m-i-1}\mathbf{B}\right).$$

We can satisfy this optimality condition for all $\mathbf{x}_0$ by solving the *linear* matrix equation:

$$\tilde{\mathbf{K}}_i^T\left(\mathbf{R} + \sum_{m=i+1}^{N-1}\mathbf{B}^T(\mathbf{A}^{m-i-1})^T\mathbf{Q}\mathbf{A}^{m-i-1}\mathbf{B}\right) + \sum_{m=i+1}^{N-1}(\mathbf{A}^m)^T\mathbf{Q}\mathbf{A}^{m-i-1}\mathbf{B} = 0.$$

We can always solve for $\tilde{\mathbf{K}}_i$ since it's multiplied by a (symmetric) positive definite matrix (it is the sum of a positive definite matrix and many positive semi-definite matrices), which is always invertible.

If you need to recover the original $\mathbf{K}_i$ parameters, you can extract them recursively with

$$\begin{aligned}\tilde{\mathbf{K}}_0 &= \mathbf{K}_0,\\ \tilde{\mathbf{K}}_n &= \mathbf{K}_n\prod_{i=0}^{n-1}(\mathbf{A} + \mathbf{B}\mathbf{K}_i), \qquad 0 < n \leq N-1.\end{aligned}$$

But often this is not actually necessary. In some applications it's enough to know the performance cost under LQR control, or to handle the response to disturbances explicitly with the disturbance-based feedback (which I've already promised for the robust control chapter). After all, the problem formulation that we've written here, which makes no mention of disturbances, assumes the model is perfect and the controls $\tilde{\mathbf{K}}_n\mathbf{x}_0$ are just as suitable for deployment as $\mathbf{K}_n\mathbf{x}[n]$.

"System-Level Synthesis" (SLS) is the name for an important and slightly different approach, where one optimizes the *closed-loop response* directly[5]. Although SLS is a very general tool, for the particular formulation we are considering here it reduces to creating additional decision variables $\mathbf{\Phi}_i$, such at that

$$\mathbf{x}[n] = \mathbf{\Phi}_n\mathbf{x}[0],$$

and writing the optimization above as

$$\min_{\tilde{\mathbf{K}}_*, \mathbf{\Phi}_*} \sum_{n=0}^{N-1} \mathbf{x}^T[0] \left(\mathbf{\Phi}_n^T \mathbf{Q} \mathbf{\Phi}_n + \tilde{\mathbf{K}}_n^T \mathbf{R} \tilde{\mathbf{K}}_n \right) \mathbf{x}[0],$$
$$\text{subject to} \quad \forall n, \quad \mathbf{\Phi}_{n+1} = \mathbf{A}\mathbf{\Phi}_n + \mathbf{B}\tilde{\mathbf{K}}_n.$$

Once again, the algorithms presented here are not as efficient as solving the Riccati equation if we only want the solution to the simple case, but they become very powerful if we want to combine the LQR synthesis with other objectives/constraints. For instance, if we want to add some sparsity constraints (e.g. enforcing that some elements of $\tilde{\mathbf{K}}_i$ are zero), then we could solve the quadratic optimization subject to linear equality constraints [6].

8.3.7 Minimum-time LQR

Coming soon. The basic result can be found in [7], and some robotics applications in, for instance, [8, 9]. For discrete-time systems, [10] provides a new computational approach.

8.3.8 Parameterized Riccati Equations

When we are designing LQR controllers based on a linearization (around some operating state or trajectory) of a nonlinear system, then it is interesting to ask: how does the LQR solution change as you change the nominal point (or trajectory)? If the nonlinear dynamics are smooth, does the solution also change smoothly?

The answer is yes -- the Riccati solution does change smoothly, and its local variation can be obtained with a little bit of work by taking Taylor approximation of the $\mathbf{A}$ and $\mathbf{B}$ matrices [11].

8.4 Exercises

Exercise 8.1 (LQR on a Drake Diagram)

To help you get a little more familiar with Drake, the exercise in this notebook will step you through the process of building a Drake Diagram, and designing an LQR controller for it. Systems and Diagrams are the way we write modular code describing dynamical systems, and implementing an algorithm like LQR against the System abstraction becomes very powerful.

8.5 Notes

8.5.1 Finite-horizon LQR derivation (general form)

For completeness, I've included here the derivation for continuous-time finite-horizon LQR with all of the bells and whistles.

Consider an time-varying affine (approximation of a) continuous-time dynamical system in state-space form:

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{u} + \mathbf{c}(t),$$

and a running cost function in the general quadratic form:

$$\ell(t, \mathbf{x}, \mathbf{u}) = \begin{bmatrix}\mathbf{x}\\1\end{bmatrix}^T \mathbf{Q}(t) \begin{bmatrix}\mathbf{x}\\1\end{bmatrix} + \begin{bmatrix}\mathbf{u}\\1\end{bmatrix}^T \mathbf{R}(t) \begin{bmatrix}\mathbf{u}\\1\end{bmatrix} + 2\mathbf{x}^T\mathbf{N}(t)\mathbf{u},$$
$$\forall t \in [t_0, t_f], \quad \mathbf{Q}(t) = \begin{bmatrix}\mathbf{Q}_{xx}(t) & \mathbf{q}_x(t)\\ \mathbf{q}_x^T(t) & q_0(t)\end{bmatrix}, \mathbf{Q}_{xx}(t) \succeq 0, \quad \mathbf{R}(t) = \begin{bmatrix}\mathbf{R}_{uu}(t) & \mathbf{r}_u(t)\\ \mathbf{r}_u^T(t) & r_0(t)\end{bmatrix}, \mathbf{R}_{uu}(t) \succ 0.$$

Observe that our LQR "optimal tracking" derivation fits in this form, as we can always write

$$(\mathbf{x} - \mathbf{x}_d(t))^T\mathbf{Q}_t(\mathbf{x} - \mathbf{x}_d(t)) + (\mathbf{u} - \mathbf{u}_d(t))^T\mathbf{R}_t(\mathbf{u} - \mathbf{u}_d(t)) + 2(\mathbf{x} - \mathbf{x}_d(t))^T\mathbf{N}_t(\mathbf{u} - \mathbf{u}_d(t)),$$

by taking

$$\begin{aligned}\mathbf{Q}_{xx} &= \mathbf{Q}_t, \quad \mathbf{q}_x = -\mathbf{Q}_t\mathbf{x}_d - \mathbf{N}_t\mathbf{u}_d, \quad q_0 = \mathbf{x}_d^T\mathbf{Q}_t\mathbf{x}_d + 2\mathbf{x}_d^T\mathbf{N}_t\mathbf{u}_d,\\ \mathbf{R}_{uu} &= \mathbf{R}_t, \quad \mathbf{r}_u = -\mathbf{R}_t\mathbf{u}_d - \mathbf{N}_t^T\mathbf{x}_d, \quad r_0 = \mathbf{u}_d^T\mathbf{R}_t\mathbf{u}_d, \quad \mathbf{N} = \mathbf{N}_t.\end{aligned}$$

Of course, we can also add a quadratic final cost with $\mathbf{Q}_f$. Let's search for a positive quadratic, time-varying cost-to-go function of the form:

$$J(t, \mathbf{x}) = \begin{bmatrix}\mathbf{x}\\1\end{bmatrix}^T \mathbf{S}(t) \begin{bmatrix}\mathbf{x}\\1\end{bmatrix}, \quad \mathbf{S}(t) = \begin{bmatrix}\mathbf{S}_{xx}(t) & \mathbf{s}_x(t)\\ \mathbf{s}_x^T(t) & s_0(t)\end{bmatrix}, \mathbf{S}_{xx}(t) \succ 0,$$
$$\frac{\partial J}{\partial \mathbf{x}} = 2\mathbf{x}^T\mathbf{S}_{xx} + 2\mathbf{s}_x^T, \quad \frac{\partial J}{\partial t} = \begin{bmatrix}\mathbf{x}\\1\end{bmatrix}^T \dot{\mathbf{S}} \begin{bmatrix}\mathbf{x}\\1\end{bmatrix}.$$

Writing out the HJB:

$$\min_{\mathbf{u}} \left[\ell(\mathbf{x}, \mathbf{u}) + \frac{\partial J}{\partial \mathbf{x}}[\mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{u} + \mathbf{c}(t)] + \frac{\partial J}{\partial t}\right] = 0,$$

we can find the minimizing $\mathbf{u}$ with

$$\frac{\partial}{\partial \mathbf{u}} = 2\mathbf{u}^T\mathbf{R}_{uu} + 2\mathbf{r}_u^T + 2\mathbf{x}^T\mathbf{N} + (2\mathbf{x}^T\mathbf{S}_{xx} + 2\mathbf{s}_x^T)\mathbf{B} = 0$$

$$\mathbf{u}^* = -\mathbf{R}_{uu}^{-1}\begin{bmatrix}\mathbf{N} + \mathbf{S}_{xx}\mathbf{B} \\ \mathbf{r}_u^T + \mathbf{s}_x^T\mathbf{B}\end{bmatrix}^T \begin{bmatrix}\mathbf{x} \\ 1\end{bmatrix} = -\mathbf{K}(t)\begin{bmatrix}\mathbf{x} \\ 1\end{bmatrix} = -\mathbf{K}_x(t)\mathbf{x} - \mathbf{k}_0(t).$$

Inserting this back into the HJB gives us the updated Riccati differential equation. Since this must hold for all $\mathbf{x}$, we can collect the quadratic, linear, and offset terms and set them each individually equal to zero, yielding:

$$\begin{aligned}
-\dot{\mathbf{S}}_{xx} =& \mathbf{Q}_{xx} - (\mathbf{N} + \mathbf{S}_{xx}\mathbf{B})\mathbf{R}_{uu}^{-1}(\mathbf{N} + \mathbf{S}_{xx}\mathbf{B})^T + \mathbf{S}_{xx}\mathbf{A} + \mathbf{A}^T\mathbf{S}_{xx}, \\
-\dot{\mathbf{s}}_x =& \mathbf{q}_x - (\mathbf{N} + \mathbf{S}_{xx}\mathbf{B})\mathbf{R}_{uu}^{-1}(\mathbf{r}_u + \mathbf{B}^T\mathbf{s}_x) + \mathbf{A}^T\mathbf{s}_x + \mathbf{S}_{xx}\mathbf{c}, \\
-\dot{s}_0 =& q_0 + r_0 - (\mathbf{r}_u + \mathbf{B}^T\mathbf{s}_x)^T\mathbf{R}_{uu}^{-1}(\mathbf{r}_u + \mathbf{B}^T\mathbf{s}_x) + 2\mathbf{s}_x^T\mathbf{c},
\end{aligned}$$

with the final conditions $\mathbf{S}(t_f) = \mathbf{Q}_f$.

In the discrete-time version, we have...

Phew! Numerical solutions to these equations can be obtained by calling the `FiniteHorizonLinearQuadraticRegulator` methods in DRAKE. May you never have to type them in and unit test them yourself.

REFERENCES

1. Michael Posa and Scott Kuindersma and Russ Tedrake, "Optimization and stabilization of trajectories for constrained dynamical systems", *Proceedings of the International Conference on Robotics and Automation (ICRA)* , pp. 1366-1373, May, 2016. [link]

2. Volker Mehrmann, "The autonomous linear quadratic control problem: theory and numerical solution", Springer , no. 163, 1991.

3. J. Lofberg, "Approximations of closed-loop minimax {MPC}", *42nd {IEEE} {International} {Conference} on {Decision} and {Control} ({IEEE} {Cat}. {No}.03CH37475)* , vol. 2, pp. 1438--1442 Vol.2, Dec, 2003.

4. Sadra Sadraddini and Russ Tedrake, "Robust Output Feedback Control with Guaranteed Constraint Satisfaction", *In the Proceedings of 23rd ACM International Conference on Hybrid Systems: Computation and Control* , pp. 12, April, 2020. [link]

5. James Anderson and John C. Doyle and Steven Low and Nikolai Matni, "System {Level} {Synthesis}", *arXiv:1904.01634 [cs, math]*, apr, 2019.

6. Yuh-Shyang Wang and Nikolai Matni and John C. Doyle, "Localized {LQR} {Optimal} {Control}", *arXiv:1409.6404 [cs, math]*, September, 2014.

7. E.I. Verriest and F.L. Lewis, "On the Linear Quadratic Minimum-Time Problem", *IEEE Transactions on Automatic Control*, vol. 36, no. 7, pp. 859-863, July, 1991.

8. Elena Glassman and Russ Tedrake, "A Quadratic Regulator-Based Heuristic for Rapidly Exploring State Space", *Under review*, 2010. [link]

9. Sikang Liu and Nikolay Atanasov and Kartik Mohta and Vijay Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control", *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* , pp. 2872-2879, Sep., 2017.

10. Tobia Marcucci and Jack Umenberger and Pablo A. Parrilo and Russ Tedrake, "Shortest Paths in Graphs of Convex Sets", *arXiv preprint*, 2023. [link]

11. Anirudha Majumdar and Mark Tobenkin and Russ Tedrake, "Algebraic Verification for Parameterized Motion Planning Libraries", *Proceedings of the 2012 American Control Conference (ACC)* , pp. 8, June, 2012. [link]

© Russ Tedrake, 2024