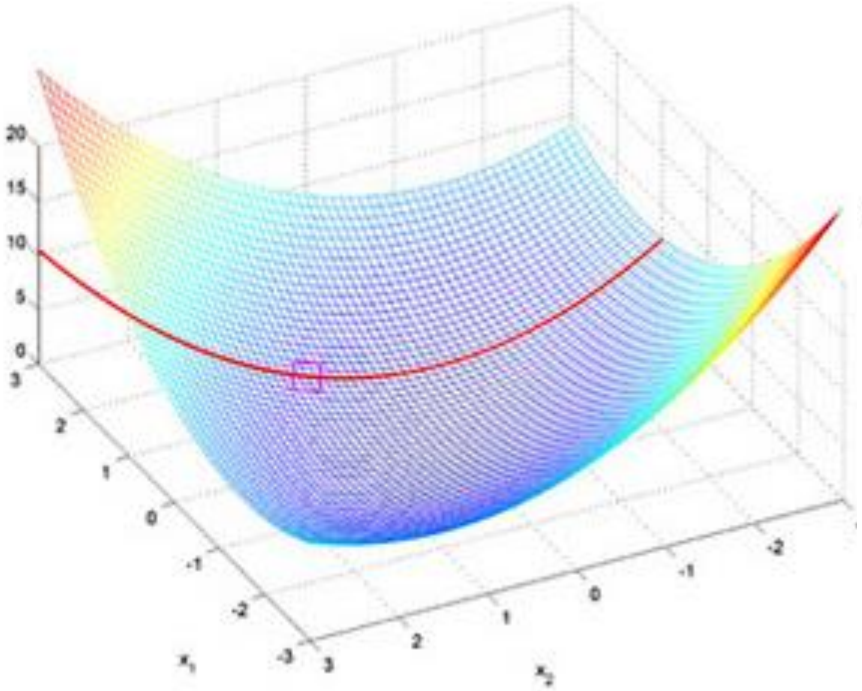
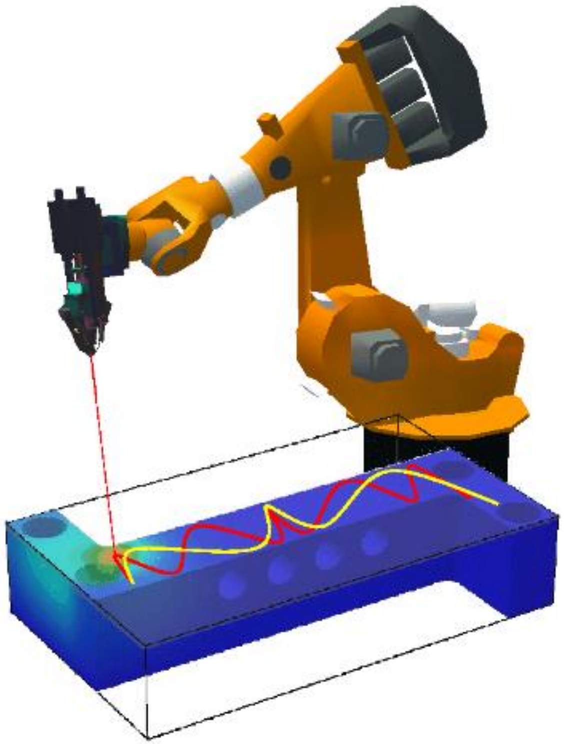
MEGR 7090/8090: Advanced Optimal Control



$$\boxed{V_n(\mathbf{x}_n) = \min_{\{\mathbf{u}_n,\mathbf{u}_{n+1},\cdots,\mathbf{u}_{N-1}\}} \left[\frac{1}{2}\sum_{k=n}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \frac{1}{2}\mathbf{x}_N^T\mathbf{Q}_N\mathbf{x}_N\right]}$$

$$\begin{aligned} V_n(\mathbf{x}_n) &= \min_{\{\mathbf{u}_n,\mathbf{u}_{n+1},\cdots,\mathbf{u}_{N-1}\}} \left[\frac{1}{2}\sum_{k=n}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \frac{1}{2}\mathbf{x}_N^T\mathbf{Q}_N\mathbf{x}_N\right] \\ &= \min_{\mathbf{u}_n}\left[\frac{1}{2}\left(\mathbf{x}_n^T\mathbf{Q}_n\mathbf{x}_n + \mathbf{u}_n^T\mathbf{R}\mathbf{u}_n\right) + \underbrace{\min_{\{\mathbf{u}_{n+1},\cdots,\mathbf{u}_{N-1}\}}\left[\frac{1}{2}\sum_{k=n+1}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \frac{1}{2}\mathbf{x}_N^T\mathbf{Q}_N\mathbf{x}_N\right]}_{V_{n+1}(\mathbf{x}_{n+1})}\right] \\ &= \min_{\mathbf{u}_n}\left[\frac{1}{2}\left(\mathbf{x}_n^T\mathbf{Q}_n\mathbf{x}_n + \mathbf{u}_n^T\mathbf{R}\mathbf{u}_n\right) + V_{n+1}(\mathbf{x}_{n+1})\right] \end{aligned}$$

$$\boxed{V_n(\mathbf{x}_n) = \min_{\mathbf{u}_n}\left[\frac{1}{2}\left(\mathbf{x}_n^T\mathbf{Q}_n\mathbf{x}_n + \mathbf{u}_n^T\mathbf{R}\mathbf{u}_n\right) + V_{n+1}(\mathbf{x}_{n+1})\right]}$$



Lecture 16

October 17, 2017

Dynamic Programming – Backward Recursion – Reminder

Getting set up:

- Quantize control variables into p discrete values
- Quantize state variables into q discrete values

Solution algorithm:

- Start at step N-1. For each of the q allowable state variables, calculate the stage cost $(g(\mathbf{x}(N-1), u(N-1))$ for each of the p allowable control variables that lead to constraint satisfaction. Control variables that do not satisfy constraints are termed ***inadmissible***. Record the optimal control signals and corresponding stage costs for each originating state.
- Move to step N-2. For each of the q allowable state variables, calculate the stage cost $(g(\mathbf{x}(N-1), u(N-1))$ and associated intermediate state, $\mathbf{x}_i(N-1)$, for each of the p allowable control variables that lead to constraint satisfaction. To determine which control variable is optimal, compute the total cost to go as $J^*_{\mathbf{x}_o(N-2)\to\mathbf{x}_i(N-1)\to\mathbf{x}_f(N)} = J^*_{\mathbf{x}_o(N-2)\to\mathbf{x}_i(N-1)} + J^*_{\mathbf{x}_i(N-1)\to\mathbf{x}_f(N)}$
- Move to step N-3 and repeat the process (total cost to go is now $J^*_{\mathbf{x}_o(N-3)\to\mathbf{x}_i(N-2)\to\mathbf{x}_f(N)} = J^*_{\mathbf{x}_o(N-3)\to\mathbf{x}_i(N-2)} + J^*_{\mathbf{x}_i(N-2)\to\mathbf{x}_f(N)}$). Keep stepping backward in time until step 0.

Dynamic Programming – Forward Recursion – Reminder

Getting set up:

- Quantize control variables into p discrete values
- Quantize state variables into q discrete values
- Usually the initial state, $\mathbf{x}(0)$, is specified (you don't get to choose your initial state)

Solution algorithm:

- Start at step 1. For each of the q allowable values of $\mathbf{x}(1)$, back-compute the value of $\mathbf{x}(0)$ and stage cost that results from each of the p allowable control values. For each admissible control value (whose associated $\mathbf{x}(0)$ satisfies initial condition requirements), determine the optimal stage cost– this cost, denoted by $J^*_{\mathbf{x}_0(0)\rightarrow\mathbf{x}_t(1)}$, is the ***optimal cost to arrive*** at state $\mathbf{x}_t(1)$.
- Move to step 2. For each of the q allowable values of $\mathbf{x}(2)$, back-compute $\mathbf{x}(1)$ and the associated stage cost for each of the p allowable control variables that lead to constraint satisfaction. To determine which control sequence is optimal, compute the total cost to arrive as $J^*_{\mathbf{x}_o(0)\rightarrow\mathbf{x}_i(1)\rightarrow\mathbf{x}_f(2)} = J^*_{\mathbf{x}_o(0)\rightarrow\mathbf{x}_i(1)} + J^*_{\mathbf{x}_i(1)\rightarrow\mathbf{x}_f(2)}$
- Move to step 2 and repeat the process. Keep stepping forward in time until step N.

Dynamic Programming – Some Comments

Backward vs. forward recursion:

Why backward recursion is usually preferred?

1) Result of DP: $\underline{u}^*$ for every (quantized) $\underline{x}_0$.
   - DP is really slow.
   - Can implement a lookup table based on DP result.
   - Can use result for benchmarking.

2) At step 1 of fwd. recursion, for every quantized state value, we compute $\underline{x}_0$ for control value.

DP vs. SQP:

SQP: Local, continuous (u can take on any value)

DP: Global, gridded

Dynamic Programming – More Realistic Example

Suppose a vehicle's dynamics are given by:

$$m\dot{v} = u - C_{rr}mg - 0.5\rho v^2 C_d A_{ref}$$
$$\dot{x} = v$$

Parameter values: $m = 1000kg,\ g = 9.8\frac{m}{s^2},\ C_{rr} = 0.01,\ \rho = 1.2\frac{kg}{m^3},\ C_d = 0.4,\ A_{ref} = 5m^2$

Suppose that our goals are:

- Given an initial position of $x = 0$, achieve a final position of $x = 1600$ (approximately one "metric" mile of travel) after 60 seconds
- Minimize total energy expended over 60 seconds

Set up a nonlinear optimization problem and solve using dynamic programming

- Identify the states and control signal
- For different quantization levels, assess computational complexity

Dynamic Programming – More Realistic Example

States (2): Position (x) and velocity (v)

- Taking q_x and q_v as the number of quantized positions and velocities, respectively, the number of state variable combinations is given by $q = q_x q_v$

Control variable (1): Applied force (u)… Number of quantized forces denoted p

Horizon length: $N = \frac{60}{\Delta T}$ (where ΔT is the discrete time step)

Computational complexity: Number of cost function evaluations (E) given by:

$$E = Npq = Npq_x q_v$$

Example code available on Canvas.

Dynamic Programming – More Realistic Example

UNC CHARLOTTE

Ex: $m\dot{v} = u - C_{rr}mg - 0.5\rho v^2 C_d A_{ref}$

$\dot{x} = v$

$$v(k+1) = v(k) + \frac{\Delta t}{m}\left(u(k) - C_{rr}mg - 0.5\rho C_d A_{ref} v(k)^2\right)$$

$$x(k+1) = x(k) + \frac{v(k) + v(k+1)}{2}\Delta t \quad \Big\} \text{ trapezoidal approximation}$$

$$J(\underline{u}; \underline{x}(0)) = \int_0^{60} u(t)\,v(t)\,dt \approx \sum_{i=0}^{N-1} u(i)\,v(i)\,\Delta t$$

Constraints: $x(N) \geq 1600$

$v(N) \geq 25$

States: x; v $\leftarrow$ 21 values $= q_v$; $x \leftarrow$ 41 values $\triangleq q_x$

Control: $u \leftarrow$ 31 values $\triangleq p$

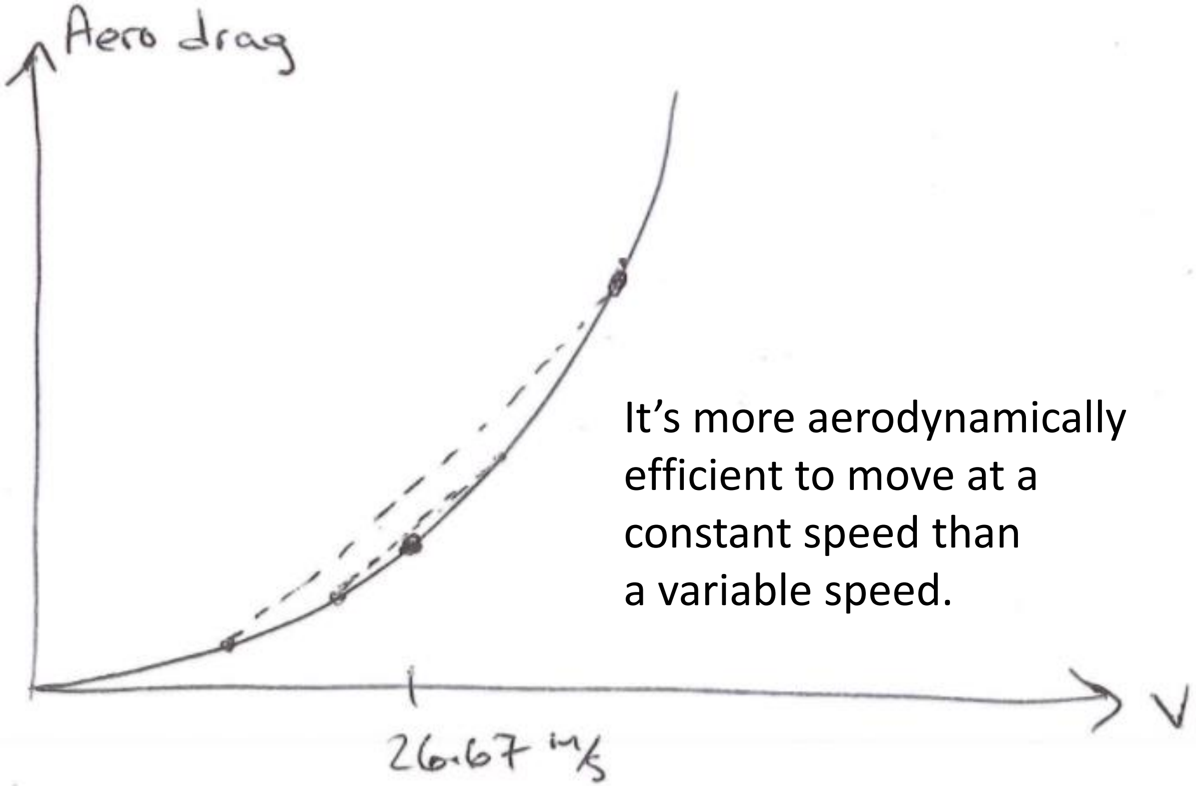
$N = 30 \Rightarrow \Delta t = 2$ seconds.

Dynamic Programming – More Realistic Example

cost to go

Req'd # of computations: $N \, q_x \, q_v \, p = 800{,}730$

curse of dimensionality



It's more aerodynamically efficient to move at a constant speed than a variable speed.

$$KE = \frac{1}{2} m v^2$$

Work must be done to increase kinetic energy. Therefore, the vehicle should finish at the minimum allowable speed. Note that its ***average*** speed must be greater than this to achieve $x(N) = 1600$ meters.

Dynamic Programming and the Principle of Optimality – A Famous Result

Consider the following system dynamics:

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + Bu(k) \qquad\qquad \mathbf{x} \in \mathbb{R}^n, u \in \mathbb{R}$$

…and the following objective function:

$$J(\mathbf{x}(0), \mathbf{u}) = \mathbf{x}^T(N)S\mathbf{x}(N) + \sum_{i=0}^{N-1} (\mathbf{x}^T(i)Q\mathbf{x}(i) + Ru^2(i))$$

where: $S \succ 0, Q \succ 0, R > 0$

- This is known as the **finite-horizon discrete-time linear quadratic regulator** problem
- Important notation (seems strange now, will make sense later): $P(0) \triangleq S$

Note: A detailed derivation for this problem is given in Kirk section 3.10

Discrete-Time LQR – Backward Recursion Step N-1

Note: $J^*_{\mathbf{x}(N)\longrightarrow\mathbf{x}(N)} = \mathbf{x}^T(N)S\mathbf{x}(N) = \mathbf{x}^T(N)P(0)\mathbf{x}(N)$

Beginning backward recursion (step N-1):

$$J_{\mathbf{x}(N-1)\longrightarrow\mathbf{x}(N)} = \mathbf{x}^T(N-1)Q\mathbf{x}(N-1) + Ru^2(N-1) + \mathbf{x}^T(N)P(0)\mathbf{x}(N)$$

$$= \mathbf{x}^T(N-1)Q\mathbf{x}(N-1) + Ru^2(N-1) + (A\mathbf{x}(N-1) + Bu(N-1))^T P(0)(A\mathbf{x}(N-1) + Bu(N-1))$$

Collecting like terms, differentiating with respect to $u(N-1)$, and setting the derivative to 0:

$$\frac{\partial J_{N-1\longrightarrow N}}{\partial u(N-1)} = 2(Ru(N-1) + B^T P(0)Bu(N-1) + B^T P(0)A\mathbf{x}(N-1))$$

$$u^*(N-1) = -\underbrace{(R + B^T P(0)B)^{-1}B^T P(0)A}_{\triangleq K(N-1)}\mathbf{x}(N-1)$$

Discrete-Time LQR – Backward Recursion Step N-2

Note: $J^*_{\mathbf{x}(N-1)\to\mathbf{x}(N)} = \mathbf{x}^T(N-1)P(1)\mathbf{x}(N-1)$ where:

$$P(1) = \big(A - BK(N-1)\big)^T P(0)\big(A - BK(N-1)\big) + K^T(N-1)RK(N-1) + Q$$

Continuing backward recursion (step N-1):

$$J_{\mathbf{x}(N-2)\to\mathbf{x}(N)} = \mathbf{x}^T(N-2)Q\mathbf{x}(N-2) + Ru^2(N-2) + \mathbf{x}^T(N-1)P(1)\mathbf{x}(N-1)$$

$$= \mathbf{x}^T(N-2)Q\mathbf{x}(N-2) + Ru^2(N-2) + (A\mathbf{x}(N-2) + Bu(N-2))^T P(1)(A\mathbf{x}(N-2) + Bu(N-2))$$

Key observation: The cost to go above is **identical** in structure to $J_{\mathbf{x}(N-1)\to\mathbf{x}(N)}$, except that:

- $u(N-1)$ and $\mathbf{x}(N-1)$ have been replaced with $u(N-2)$ and $\mathbf{x}(N-2)$
- $P(0)$ has been replaced with $P(1)$

Result: The optimal control signal ($u^*(N-2)$) will be the same as before, except that $\mathbf{x}(N-1)$ and $P(0)$ will be replaced with $\mathbf{x}(N-2)$ and $P(1)$, respectively

Discrete-Time LQR – Derivation in Class

UNC CHARLOTTE

Given: $\underline{x}(k+1) = A\underline{x}(k) + Bu(k)$

Minimize $J(\underline{u}; \underline{x}(0)) = \underline{x}^T(N)S\underline{x}(N) + \sum_{i=0}^{N-1}\left(\underline{x}^T(i)Q\underline{x}(i) + Ru^2(i)\right)$

Equivalent to minimizing

$$J(\underline{u}; \underline{x}(0)) = \sum_{i=0}^{N-1}\left[\underline{x}^T(i+1)Q\underline{x}(i+1) + Ru^2(i)\right] \quad \text{when } S = Q$$

$$= \underline{x}^T(N)Q\underline{x}(N) + \sum_{i=0}^{N-1}\left[\underline{x}^T(i)Q\underline{x}(i) + Ru^2(i)\right] - \underbrace{\underline{x}^T(0)Q\underline{x}(0)}_{\substack{\text{Constant} \\ \Rightarrow \text{no impact} \\ \text{on minimizer}}}$$

$$\boxed{P(0) \triangleq S}$$

$$J^*_{\underline{x}(N)\to\underline{x}(N)} = \underline{x}^T(N)P(0)\underline{x}(N)$$

Just as the purpose (and Iraq war metaphor) of the strange agrarian community in M. Night Shyamalan's *The Village* is initially hazy but becomes remarkably clear later on, the purpose of the mysterious assignment of $P(0)$ will become clear…with stunning implications for LQR control

Discrete-Time LQR – Derivation in Class

Stage N-1:

$$J_{\underline{x}(N-1)\to \underline{x}(N)} = \underline{x}^T(N-1)Q\,\underline{x}(N-1) + Ru^2(N-1) + \underline{x}^T(N)\,P(0)\,\underline{x}(N)$$

$$= \underline{x}^T(N-1)Q\,\underline{x}(N-1) + Ru^2(N-1) + [A\underline{x}(N-1) + Bu(N-1)]^T P(0) \ldots$$

$$[A\underline{x}(N-1) + Bu(N-1)]$$

$$\frac{\partial J_{\underline{x}(N-1)\to x(N)}}{\partial u(N-1)} = 2Ru(N-1) + 2B^T P(0)Bu(N-1)$$

$$+ 2B^T P(0) A\underline{x}(N-1) = 0 \text{ when}$$

$$u(N-1) = u^*(N-1)$$

$$\Rightarrow u^*(N-1) = -\underbrace{(R + B^T P(0) B)^{-1}(B^T P(0) A)}_{K(N-1)}\,\underline{x}(N-1)$$

Discrete-Time LQR – Derivation in Class

$$J^*_{\underline{x}(N-1)\to\underline{x}(N)} = \underline{x}^T(N-1)Q\underline{x}(N-1) + \underline{x}^T(N-1)K^T(N-1)\overset{R}{}K(N-1)\underline{x}(N-1)$$

$$+\left[(A-BK(N-1))\underline{x}(N-1)\right]^T P(0)\left[(A-BK(N-1))\underline{x}(N-1)\right]$$

$$= \underline{x}^T(N-1)\underbrace{\left[Q+K^T(N-1)RK(N-1)+(A-BK(N-1))^T P(0)(A-BK(N-1))\right]}_{P(1)}\underbrace{}_{\underline{x}(N-1)}$$

$$\Rightarrow J^*_{\underline{x}(N-1)\to\underline{x}(N)} = \boxed{\underline{x}^T(N-1)P(1)\underline{x}(N-1)}$$

Same as at stage N, with $P(0)$ replaced by $P(1)$ (ahh…so that's the reason for $P(i)$) and $\mathbf{x}(N)$ replaced with $\mathbf{x}(N-1)$

…At stage N-2, we have an identical optimization problem to the one at stage N-1, with $\mathbf{x}(N-1)$ replaced with $\mathbf{x}(N-2)$, $u(N-1)$ replaced with $u(N-2)$, and $P(0)$ replaced with $P(1)$

…And if the problems are identical (with index substitutions), then the solutions will be identical (with index substitutions)!

Discrete-Time LQR – General Control Law

Remember: $u^*(N-1) = -K(N-1)\mathbf{x}(N-1)$ and $u^*(N-2) = -K(N-2)\mathbf{x}(N-2)$

where: $K(N-1) = -(R + B^T P(0) B)^{-1} B^T P(0) A$ and $K(N-2) = -(R + B^T P(1) B)^{-1} B^T P(1) A$

where: $P(0) = S$ and $P(1) = \big(A - BK(N-1)\big)^T P(0) \big(A - BK(N-1)\big) + K^T(N-1) R K(N-1) + Q$

Continuing the backward recursion (generic step N-i):

$$u^*(N-i) = -K(N-i)\mathbf{x}(N-i)$$

where: $K(N-i) = -(R + B^T P(i-1) B)^{-1} B^T P(i-1) A$

where: $P(i) = \big(A - BK(N-i)\big)^T P(i-1) \big(A - BK(N-i)\big) + K^T(N-i) R K(N-i) + Q$

Key takeaway: The resulting controller is a linear, time-varying controller whose gains at each step can be found recursively!

Discrete-Time LQR – Block Diagram and Observations

Key takeaway (reminder): The resulting controller is a linear, time-varying controller whose gains at each step can be found recursively!

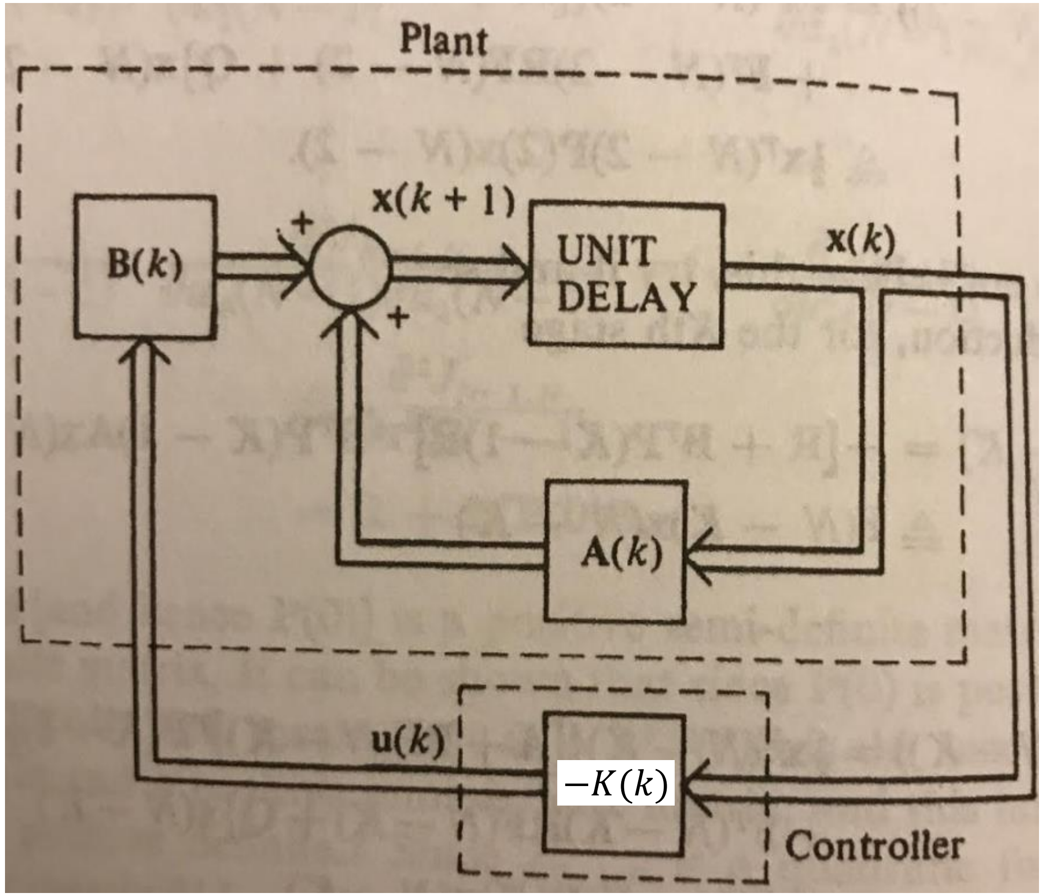


Image source: Kirk, page 82

Key differences from standard DP:

- No quantization of control and states (since cost can be differentiated)
- Unlike DP, where ***a sequence of control signals is computed all at once***, LQR lends itself to a ***feedback control law*** (which happens to be linear and time-varying)

Infinite Horizon LQR - Setup

Consider the following system dynamics:

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + Bu(k) \qquad\qquad \mathbf{x} \in \mathbb{R}^n, u \in \mathbb{R}$$

…and the following objective function:

$$J(\mathbf{x}(0), \mathbf{u}) = \sum_{i=0}^{\infty} (\mathbf{x}^T(i) Q \mathbf{x}(i) + Ru^2(i))$$

where: $Q \succ 0, R > 0$

…Same problem as before, except that:

- $N = \infty$
- There is no terminal penalty

Infinite Horizon LQR - Solution

Key facts:

- In the finite-horizon LQR problem, $J^*_{\mathbf{x}(i)\to\mathbf{x}(N)} = \mathbf{x}^T(i)P(i)\mathbf{x}(i)$, regardless of the presence of a terminal penalty
- For a time-invariant system, the infinite horizon cost to go from a given state ($\mathbf{x}$) is the same regardless of the time instant at which the controller is "turned on":

$$J^*_{\mathbf{x}(i)\to\infty} = J^*_{\mathbf{x}(i+1)\to\infty}$$

- Putting these facts together, it must be true that $P(i) = P(i+1) = P$…P is a constant!

Recall the update laws for $\boldsymbol{P}$ and $\boldsymbol{K}$: $K(N-i) = -(R + B^T P(i-1)B)^{-1}B^T P(i-1)A$

$$P(i) = \left(A - BK(N-i)\right)^T P(i-1)\left(A - BK(N-i)\right) + K^T(N-i)RK(N-i) + Q$$

The "update" laws now become: $K = -(R + B^T PB)^{-1}B^T PA$

$$\boxed{P = (A-BK)^T P(A-BK) + K^T RK + Q}$$

Discrete-time algebraic Riccati equation

Infinite Horizon LQR - Solution

Remember: $J^*_{\underline{x}(i) \to x(\infty)} = J^*_{\underline{x}(i+1) \to \underline{x}(\infty)}$

same state values

$$\Rightarrow J^*_{\underline{x}(i) \to \underline{x}(\infty)} = \underline{x}^T(i)\, P(\infty - i)\, \underline{x}(i)$$

$$J^*_{\underline{x}(i+1) \to \underline{x}(\infty)} = \underline{x}^T(i+1)\, P(\infty - i - 1)\, \underline{x}(i+1)$$

If $\underline{x}(i) = \underline{x}(i+1)$, $P(\infty - i - 1) = P(\infty - i)$

Infinite Horizon LQR - Solution

UNC CHARLOTTE

Given $\underline{x}^{(k+1)} = A\underline{x}^{(k)} + Bu^{(k)}$, we can design a state feedback-based regulator in 2 ways:

1) Pole placement:

$$u = -K\underline{x}(k)$$

$$\Rightarrow \underline{x}_{(k+1)} = \underbrace{(A - BK)}\underline{x}(k)$$

Set eigenvalues equal to desired pole locations.

2) LQR design:

Specify Q, R $(Q \in \mathbb{R}^{N \times N}, R \in \mathbb{R})$

where $J(\underline{x}_0, K) = \sum_{i=0}^{\infty} [\underline{x}^T(i) Q \underline{x}(i) + R u(i)^2]$

$\Rightarrow$ Determine K based on discrete-time algebraic Riccati equation.

Infinite Horizon LQR Solution in MATLAB

Syntax:

```
K = dlqr(A,B,Q,R);
```

Resulting feedback gain vector

Example to work in class:

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + Bu(k)$$

$$J(\mathbf{x}(0), \mathbf{u}) = \sum_{i=0}^{\infty} \left(\mathbf{x}^T(i)Q\mathbf{x}(i) + Ru^2(i)\right)$$

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad R = 1$$

Observations About the LQR Solution

Good features:
- Closed-form solution available (via differentiation of cost to go), so no need to quantize states and control variables
- Optimal controller is linear state feedback…can be implemented as a feedback controller, rather than just a sequence of control values that begins at time 0
- With an infinite horizon and time-invariant system, the feedback gain is time-invariant

Not so good features:
- Quadratic cost function (in states and control variables) required
- No constraints considered

Question to be addressed in future lectures: Can we obtain the good features above while allowing for more general cost function structures and constraints?

Preview of Upcoming Lectures

Model predictive control – answers the question of "how do we take the trajectory optimization algorithms that we've learned about and use them for *real-time feedback control*?":

- Basic MPC setup and notation (October 19)
- MPC implementation in MATLAB and Simulink (October 24)
- Stability and robustness of MPC (October 26)