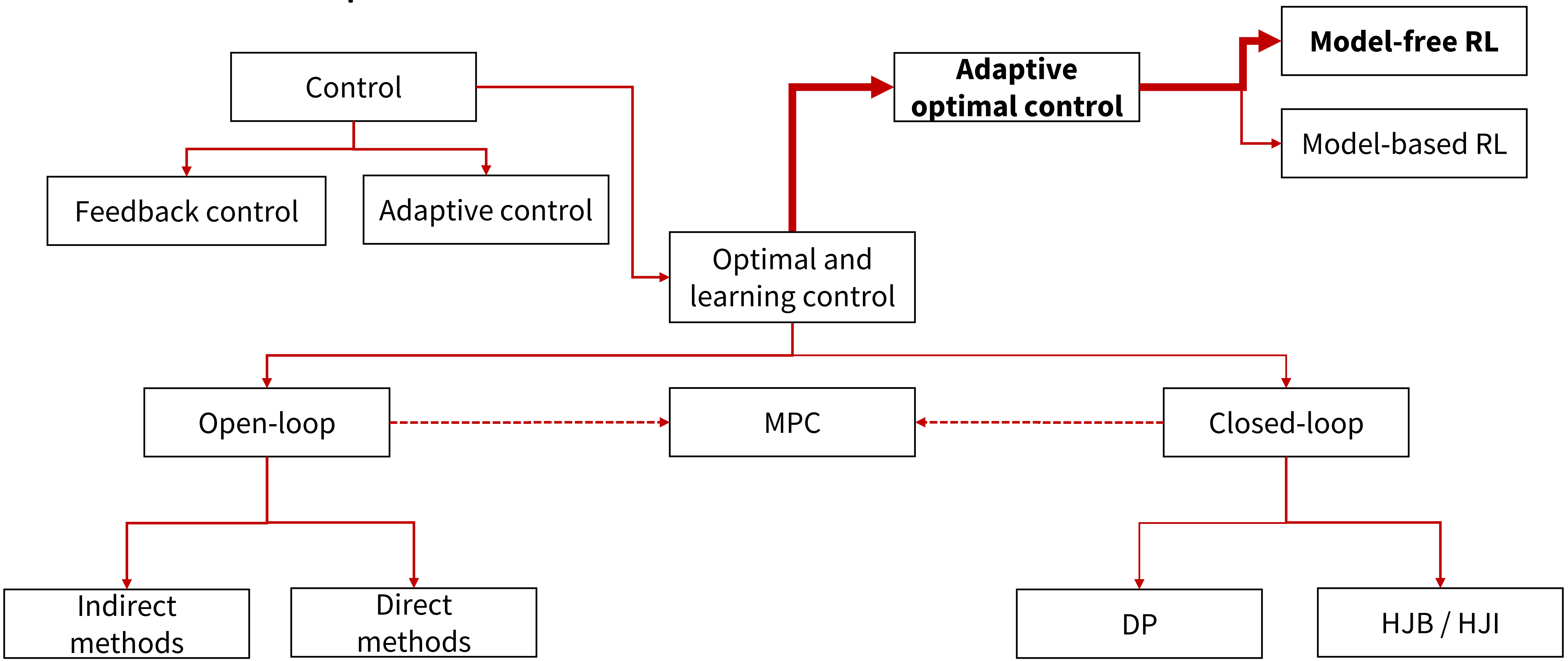
AA203
Optimal and Learning-based Control

Intro to reinforcement learning; dual control; LQG

Stanford University

Autonomous Systems Lab

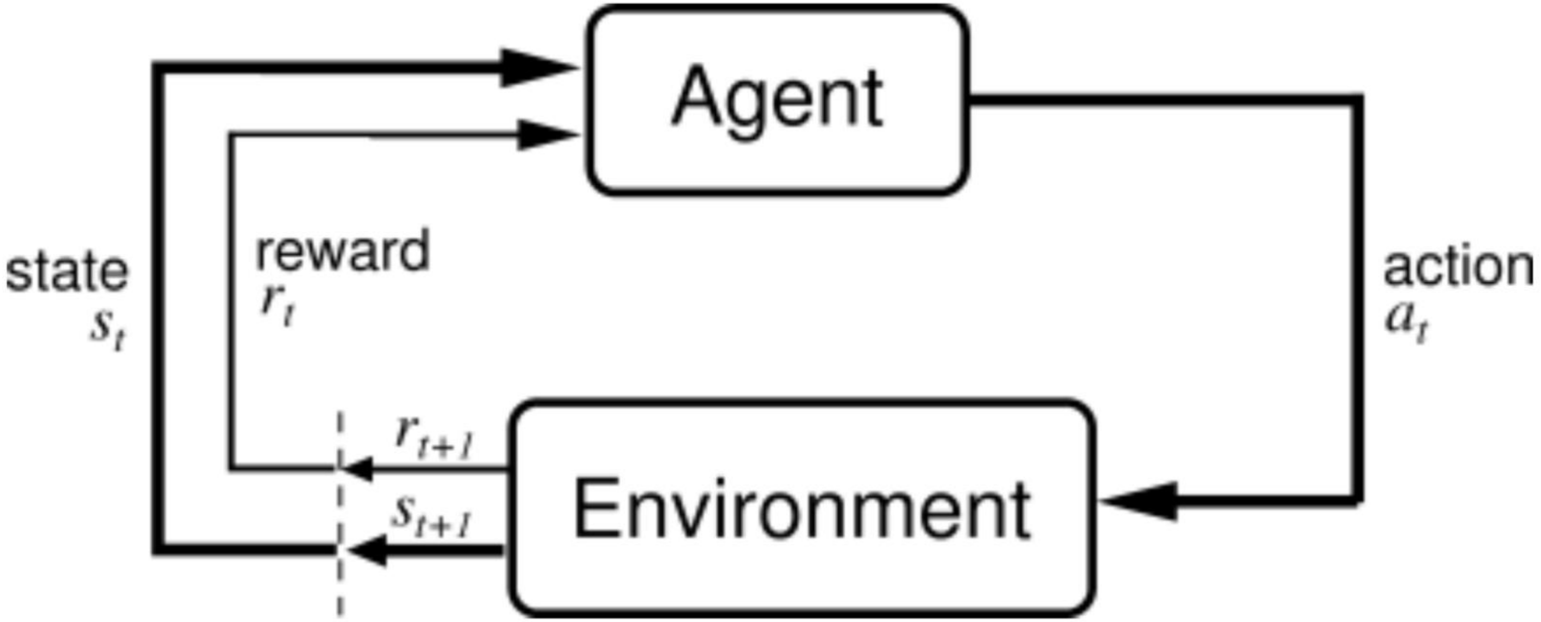
Roadmap



Control
Feedback control
Adaptive control
Optimal and learning control
Adaptive optimal control
Model-free RL
Model-based RL
Open-loop
MPC
Closed-loop
Indirect methods
Direct methods
DP
HJB / HJI

What is Reinforcement Learning?

Learning how to make good decisions by interaction.



Why Reinforcement Learning?

- Only need to specify a **reward function**.
  Agent learns everything else!
- Successes in
  - Helicopter acrobatics
  - Superhuman Gameplay: Backgammon, Go, Atari
  - Investment portfolio management
  - Making a humanoid robot walk

Why Reinforcement Learning?

- Only need to specify a **reward function**. Agent learns everything else!
- Successes in
  - Helicopter acrobatics
    - positive for following desired traj, negative for crashing
  - Superhuman Gameplay: Backgammon, Go, Atari
    - positive/negative for winning/losing the game
  - Investment portfolio management
    - positive reward for $$$
  - Making a humanoid robot walk
    - positive for forward motion, negative for falling

Infinite Horizon MDPs

State:	$x \in \mathcal{X}$	(often $s \in \mathcal{S}$)
Action:	$u \in \mathcal{U}$	(often $a \in \mathcal{A}$)
Transition Function:	$T(x_t \mid x_{t-1}, u_{t-1}) = p(x_t \mid x_{t-1}, u_{t-1})$	
Reward Function:	$r_t = R(x_t, u_t)$	
Discount Factor:	γ	

MDP: $\mathcal{M} = (\mathcal{X}, \mathcal{U}, T, R, \gamma)$

Infinite Horizon MDPs

MDP: $\mathcal{M} = (\mathcal{X}, \mathcal{U}, T, R, \gamma)$

Stationary policy: $u_t = \pi(x_t)$

Goal: Choose policy that **maximizes cumulative (discounted) reward**

$$V^* = \max_{\pi} E\left[\sum_{t \geq 0} \gamma^t R\big(x_t, \pi(x_t)\big)\right];$$

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t \geq 0} \gamma^t R\big(x_t, \pi(x_t)\big)\right]$$

Infinite Horizon MDPs

- The optimal value function $V^*(x)$ satisfies Bellman's equation

$$V^*(x) = \max_u \Bigg(\underbrace{R(x,u) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x,u)\, V^*(x')}_{Q^*(x,u)} \Bigg)$$

- For any stationary policy π, the value $V_\pi(x) \coloneqq E\left[\sum_{t\geq 0} \gamma^t R\big(x_t, \pi(x_t)\big)\right]$ is the unique solution to the equation

$$V_\pi(x) = \underbrace{R(x,\pi(x)) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x,\pi(x))\, V_\pi(x')}_{Q_\pi(x,\pi(x))}$$

Infinite Horizon MDPs

- The optimal state-action value function (Q function) $Q^*(x, u)$ satisfies Bellman's equation

$$Q^*(x,u) = R(x,u) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x,u) \max_{u'} Q^*(x',u')$$

- For any stationary policy π, the corresponding Q function satisfies

$$Q_\pi(x,u) = R(x,u) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x,u)\, Q_\pi(x',\pi(x'))$$

Solving infinite-horizon MDPs

If you know the model (i.e., the transition function T and reward function R), use ideas from dynamic programming

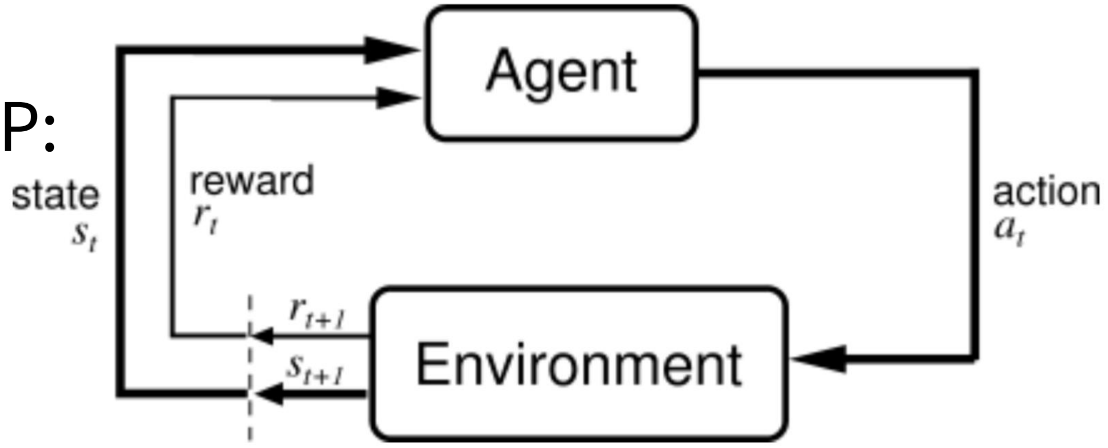
- Value Iteration / Policy Iteration

Reinforcement Learning: learning from interaction

- Model-based (related to system ID -- will see more later)
- Model-free
  - Value based (today) – SARSA, Q-learning, etc.
  - Policy based – policy gradient methods

Learning from Experience

- Without access to the model, agent needs to optimize a policy from interaction with an MDP
- Only have access to trajectories in MDP:
- $\tau = (x_0, u_0, r_0, x_1, \ldots, u_{H-1}, r_{H-1}, x_H)$



Learning from Experience

How to use trajectory data?

- Model-based approach: estimate $T(x'|x, u)$, then use model to plan

- Model-free:
  - Value based approach: estimate optimal value (or Q) function from data
  - Policy based approach: use data to determine how to improve policy
  - Actor Critic approach: learn both a policy and a value/Q function

Temporal difference (TD) learning

- Main idea: use *bootstrapped* Bellman equation to update value estimates
- *Bootstrapping*: use learned value for next state to estimate value at current state
  - Combines Monte Carlo and dynamic programming; aim to enforce consistency with respect to Bellman's equation:

$$\mathrm{E}[\underbrace{Q_\pi(x_k, u_k) - (r_k + \gamma Q_\pi(x_{k+1}, u_{k+1})}_{\text{Temporal Difference (TD) error}}] = 0$$

TD policy evaluation

Suppose we have a policy π; we want to compute an estimate of Q_π.

With step size $\alpha \in (0,1)$, loop:

1. Sample (x_k, u_k, r_k, x_{k+1}) from MDP
2. $\hat{Q}(x_k, u_k) \leftarrow \hat{Q}(x_k, u_k) + \alpha\left(r_k + \gamma \hat{Q}(x_{k+1}, u_{k+1}) - \hat{Q}(x_k, u_k)\right)$

Notes:

- Can consider a decreasing sequence of step sizes to ensure convergence
- TD-Gammon: the AlphaGo of the early 90s!

Generalized policy iteration

Loop:

1. Perform *policy evaluation* step to estimate Q_π
2. Perform *policy improvement* step using Q_π to yield π'
3. Set $\pi \leftarrow \pi'$

SARSA (state-action-reward-next state-next action)

Online (*on-policy*) learning algorithm; while sampling from MDP using a policy π, combine

1. TD policy evaluation step

with

2. Policy improvement step:

$$\pi'(x) = \text{argmax}_u \, Q_\pi(x, u)$$

Greedy (with respect to Q function) policy improvement at each time step.

Q-learning

Instead of estimating Q_π, try to estimate Q^* via

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha\left(r_k + \gamma \max_{\mathrm{u}} Q(x_{k+1}, u) - Q(x_k, u_k)\right)$$

(using the TD error for the optimal policy π^*, instead of π).

Thus, we aim to estimate Q^* from a (possibly sub-optimal) demonstration policy π. This property is known as *off-policy* learning.

Exploration vs. Exploitation

In contrast to standard machine learning on fixed data sets, in RL we **actively gather the data we use to learn.**

- We can only learn about states we visit and actions we take
- Need to **explore** to ensure we get the data we need
- Efficient exploration is a fundamental challenge in RL!

Simple strategy: add noise to the policy.

ϵ-greedy exploration:

- With some small probability ϵ, take a random action; otherwise take the most promising action

On-policy Q-learning algorithm

Initialize $Q(x, u)$ for all states and actions.

Let $\pi(x)$ be an ϵ-greedy policy according to Q, i.e.,

$$\pi(x) = \begin{cases} \text{UniformRandom}(\mathcal{U}) & \text{with probability } \epsilon \\ \text{argmax}_u Q(x, u) & \text{with probability } (1 - \epsilon) \end{cases}$$

Loop:

1. Take action: $u_k \sim \pi(x_k)$.
2. Observe reward and next state: (r_k, x_{k+1}).
3. Update Q to minimize TD error:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha\left(r_k + \max_u Q(x_{k+1}, u) - Q(x_k, u_k)\right)$$

Fitted Q Learning

How to deal with large/continuous state/action spaces?

Use parametric model for Q function: $Q_\theta(x, u)$ (e.g., $Q_\theta(x, u) = \theta^T \phi(x, u)$)

Stochastic gradient descent on squared TD error to update θ:

$$\theta \leftarrow \theta + \underset{\textbf{learning rate}}{\alpha} \left(\underbrace{r_k + \gamma \max_u Q_\theta(x_{k+1}, u) - Q_\theta(x_k, u_k)}_{\frac{d(\textit{Squared TD Error})}{dQ}} \right) \underbrace{\nabla_\theta Q_\theta(x_k, u_k)}_{\frac{dQ}{d\theta}}$$

Q Learning Recap

Pros:

- Can learn Q function from any interaction data, not just trajectories gathered using the current policy (“**off-policy” algorithm**)
- Relatively data-efficient compared to SARSA (can reuse old interaction data)

Cons:

- Need to optimize over actions: hard to apply to continuous action spaces
- Optimal Q function can be complicated, hard to learn
- Optimal policy might be much simpler!

Problems with imperfect state information

Motivating question: can we devise a notion of optimal exploration?

Consider a more general problem setup:

- Now the controller, instead of having perfect knowledge of the state, has access to observations $\boldsymbol{z}_k$ of the form
$$\boldsymbol{z}_0 = h_0(\boldsymbol{x}_0, \boldsymbol{v}_0), \qquad \boldsymbol{z}_k = h(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k)$$
- The random observation disturbance is characterized by a given probability distribution
$$P_{\boldsymbol{v}_k}(\cdot \mid \boldsymbol{x}_k, \dots, \boldsymbol{x}_0, \boldsymbol{u}_{k-1}, \dots, \boldsymbol{u}_0, \boldsymbol{w}_{k-1}, \dots, \boldsymbol{w}_0, \boldsymbol{v}_{k-1}, \dots, \boldsymbol{v}_0)$$
- The initial state $\boldsymbol{x}_0$ is also random and characterized by given $P_{\boldsymbol{x}_0}$

Partially Observed MDP (POMDP)

- MDP with *observation model* $H(z|x, u)$
- Observations do not have Markov property: current observation does not provide same amount of info as history of all observations
  ➔ DP methods aren't strictly applicable (Bellman's equation holds only for Markovian systems)
- Includes systems with unknown parameters
  - Unknown parameters fixed in time: *Bayes-adaptive MDP*

Reduction to fully observed case

- Define the *information vector* as

$$\boldsymbol{I}_k = (\boldsymbol{z}_0, \dots, \boldsymbol{z}_k, \boldsymbol{u}_0, \dots, \boldsymbol{u}_{k-1}), \qquad \boldsymbol{I}_0 = \boldsymbol{z}_0$$

- Focus is now on policies $\pi_k(\boldsymbol{I}_k) \in U_k$, i.e., we want to find a policy that minimizes

$$J_\pi = E_{\substack{\boldsymbol{x}_0, \boldsymbol{w}_k, \boldsymbol{v}_k \\ k=0,\dots,N-1}} \left[g_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} g_k(\boldsymbol{x}_k, \pi_k(\boldsymbol{I}_k), \boldsymbol{w}_k) \right]$$

Solution strategies

1. Reformulation as a perfect state information problem (main idea: make the information vector the state of the system)
   - Main drawback: state has *expanding* dimension!
2. Reason in terms of sufficient statistics, i.e., quantities that ideally are smaller than $\boldsymbol{I}_k$ and yet summarize all its essential content
   - Main example: filtering to maintain a conditional probability distribution $P_{\boldsymbol{x}_k|\boldsymbol{I}_k}$; the belief distribution over the state (assuming $\boldsymbol{v}_k \sim P_{\boldsymbol{v}_k}(\cdot \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}))$
   - Condition probability distribution leads to a decomposition of the optimal controller in two parts:

$$\pi_k\left(P_{\boldsymbol{x}_k|\boldsymbol{I}_k}\right)$$

Actuator

Estimator

Dual control

- By performing DP in this "hyperstate" $\boldsymbol{I}_k$, one can find a controller that optimally probes/explores the system
- Practically, designing dual controllers is difficult, so sub-optimal exploration heuristics are used
- Active area of research: see Wittenmark, B. "Adaptive dual control," (2008) for an introduction

Special case: Linear Quadratic Gaussian (LQG) control

Discrete LQG: find control policy that minimizes

$$E\left[\boldsymbol{x}_N^T Q \boldsymbol{x}_N + \sum_{k=0}^{N-1} \left(\boldsymbol{x}_k^T Q \boldsymbol{x}_k + \boldsymbol{u}_k^T R \boldsymbol{u}_k\right)\right]$$

subject to

- the dynamics $\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k + \boldsymbol{w}_k$
- the measurement equation $\boldsymbol{z}_k = C\boldsymbol{x}_k + \boldsymbol{v}_k$

and with $\boldsymbol{x}_0, \{\boldsymbol{w}_k\}, \{\boldsymbol{v}_k\}$, independent and Gaussian vectors (and in addition $\{\boldsymbol{w}_k\}, \{\boldsymbol{v}_k\}$ zero mean)

LQG

- LQG separation principle (see notes Section 3.4.1):
  Estimation error $x_k - E[x_k|I_k]$ is independent of control actions $u_{0:k-1}$
  - Briefly, linearity makes it so that there is no such thing as active exploration; information gain is the same from anywhere in the state space
  - Upshot is that *state estimator* and *controller* can be designed independently
- Specifically, the solution results in:
  - $\hat{x}_k = E[x_k|I_k]$ computed via Kalman filter (optimal linear quadratic estimator)
  - Optimal feedback $u_k = F_k \hat{x}_k$; F_k same as in LQR case
- We can design *state estimator* and *controller* independently
- Certainty-equivalent LQR control on estimated state is optimal dual controller – certainly not true in general!

Next time

- Nonlinearity: trajectory optimization, iterative LQR and DDP