



Robotics

Spring 2023

Abhishek Gupta

TAs: Yi Li, Srivatsa GS

Most slides courtesy of Sidd Srinivasa (UW), Pieter Abbeel (UC Berkeley)

Recap: Course Overview

Filtering/Smoothing

Localization

Mapping

SLAM

Search

Motion Planning

TrajOpt

Stability/Certification

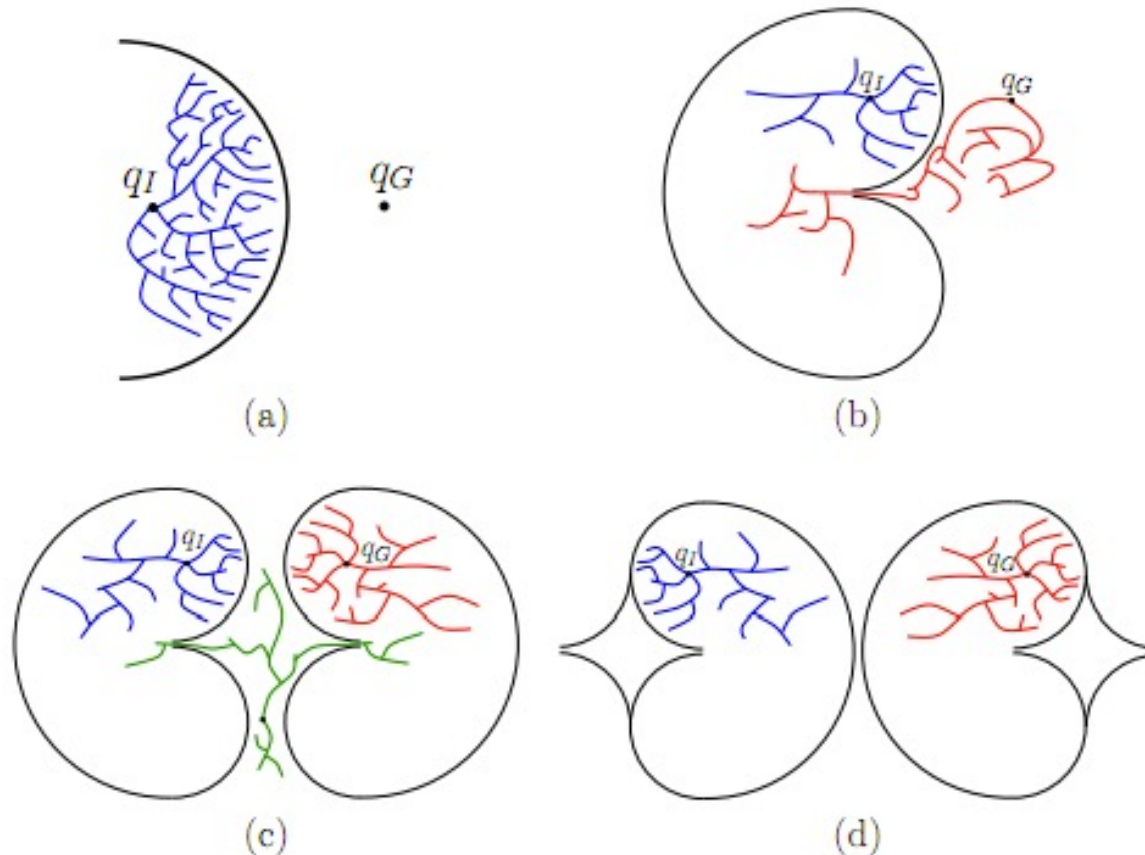
MDPs and RL

Imitation Learning

Solving POMDPs

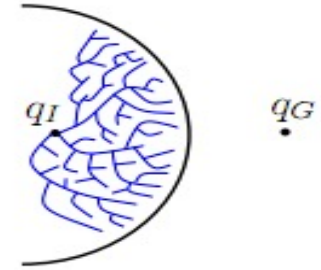
Multi-directional RRT

- Planning around obstacles or through narrow passages can often be easier in one direction than the other



Resolution-Complete RRT (RC-RRT)

- Issue: nearest points chosen for expansion are (too) often the ones stuck behind an obstacle

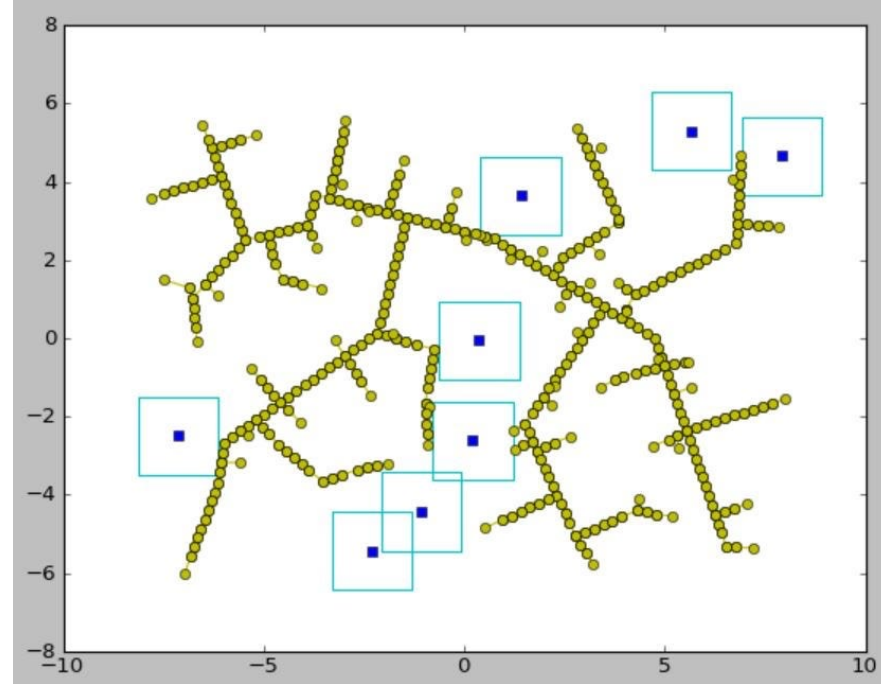
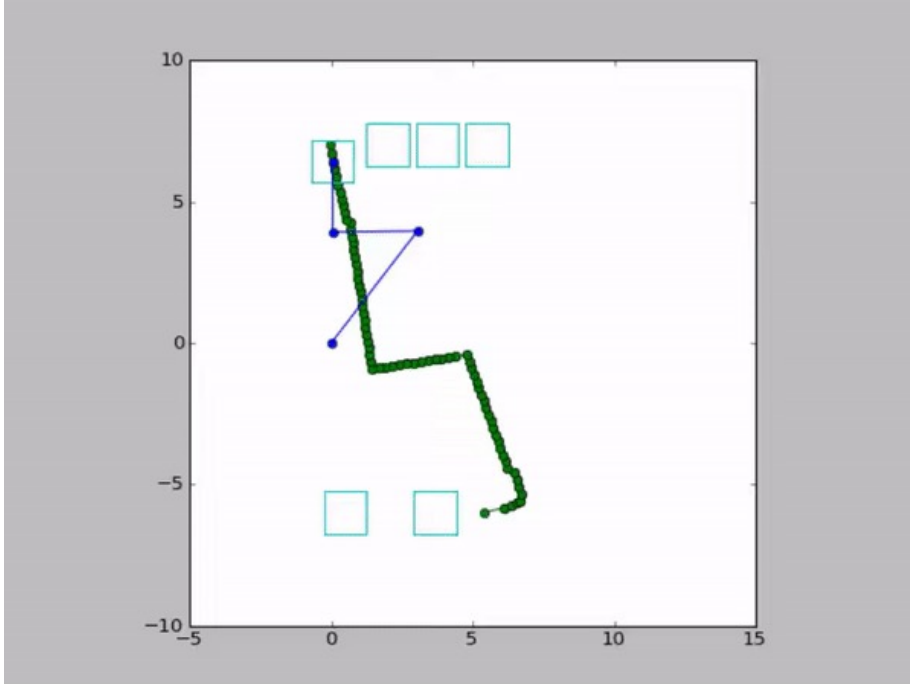


RC-RRT solution:

- Choose a maximum number of times, m , you are willing to try to expand each node
- For each node in the tree, keep track of its Constraint Violation Frequency (CVF)
- Initialize CVF to zero when node is added to tree
- Whenever an expansion from the node is unsuccessful (e.g., per hitting an obstacle):
 - Increase CVF of that node by 1
 - Increase CVF of its parent node by $1/m$, its grandparent $1/m^2$, ...
- When a node is selected for expansion, skip over it with probability CVF/m

Why is RRT not enough?

- RRT guarantees probabilistic completeness but not optimality (shortest path)
 - In practice leads to paths that are very roundabout and non-direct -> not shortest paths



Asymptotically optimal RRT \rightarrow RRT*

- Asymptotically optimal version of RRT*
- Main idea:
 - Swap new point in as parent for nearby vertices who can be reached along shorter path through new point than through their original (current) parent
 - Consider path lengths and not just connectivity

RRT*

Algorithm 6: RRT*

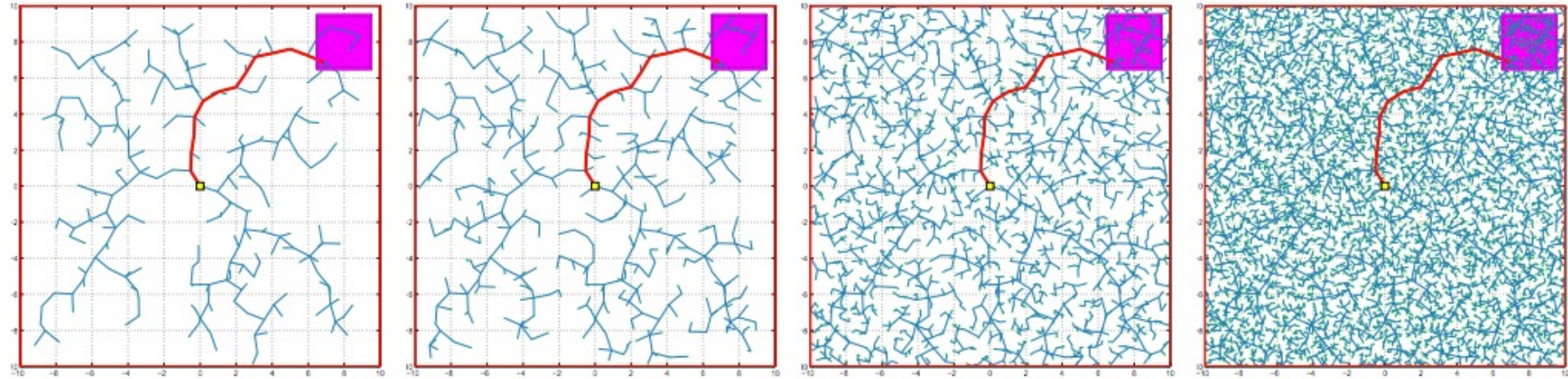
```
1  $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset;$ 
2 for  $i = 1, \dots, n$  do
3    $x_{\text{rand}} \leftarrow \text{SampleFree}_i;$ 
4    $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}});$ 
5    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}});$ 
6   if  $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$  then
7      $X_{\text{near}} \leftarrow \text{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\});$ 
8      $V \leftarrow V \cup \{x_{\text{new}}\};$ 
9      $x_{\text{min}} \leftarrow x_{\text{nearest}}; c_{\text{min}} \leftarrow \text{Cost}(x_{\text{nearest}}) + c(\text{Line}(x_{\text{nearest}}, x_{\text{new}}));$ 
10    foreach  $x_{\text{near}} \in X_{\text{near}}$  do // Connect along a minimum-cost path
11      if  $\text{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$  then
12         $x_{\text{min}} \leftarrow x_{\text{near}}; c_{\text{min}} \leftarrow \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}}))$ 
13     $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$ 
14    foreach  $x_{\text{near}} \in X_{\text{near}}$  do // Rewire the tree
15      if  $\text{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \text{Cost}(x_{\text{new}}) + c(\text{Line}(x_{\text{new}}, x_{\text{near}})) < \text{Cost}(x_{\text{near}})$ 
16        then  $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}});$ 
17         $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$ 
18 return  $G = (V, E);$ 
```

Connect new node to a better parent

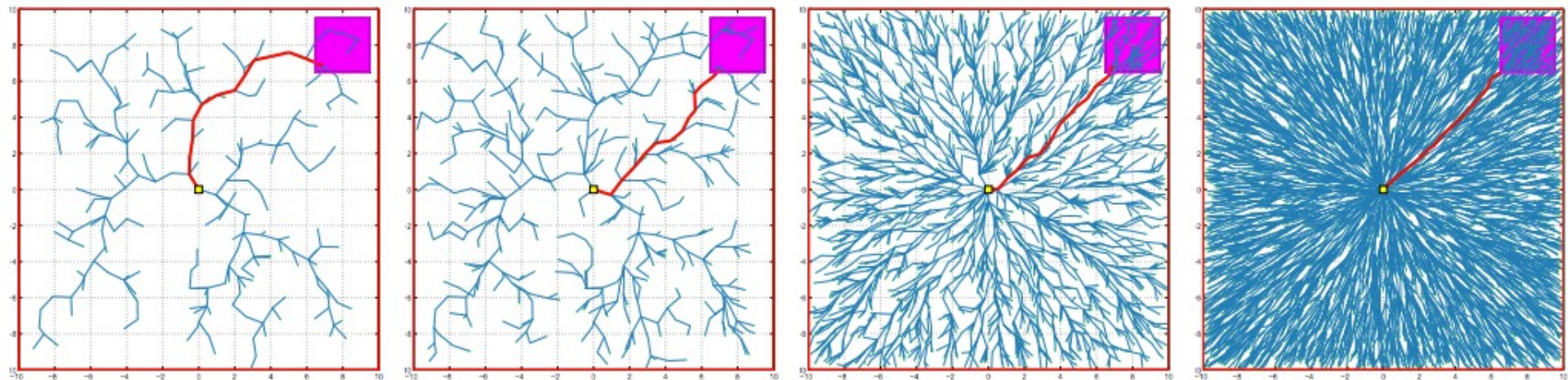
Rewire nearby nodes through new node

RRT*

RRT

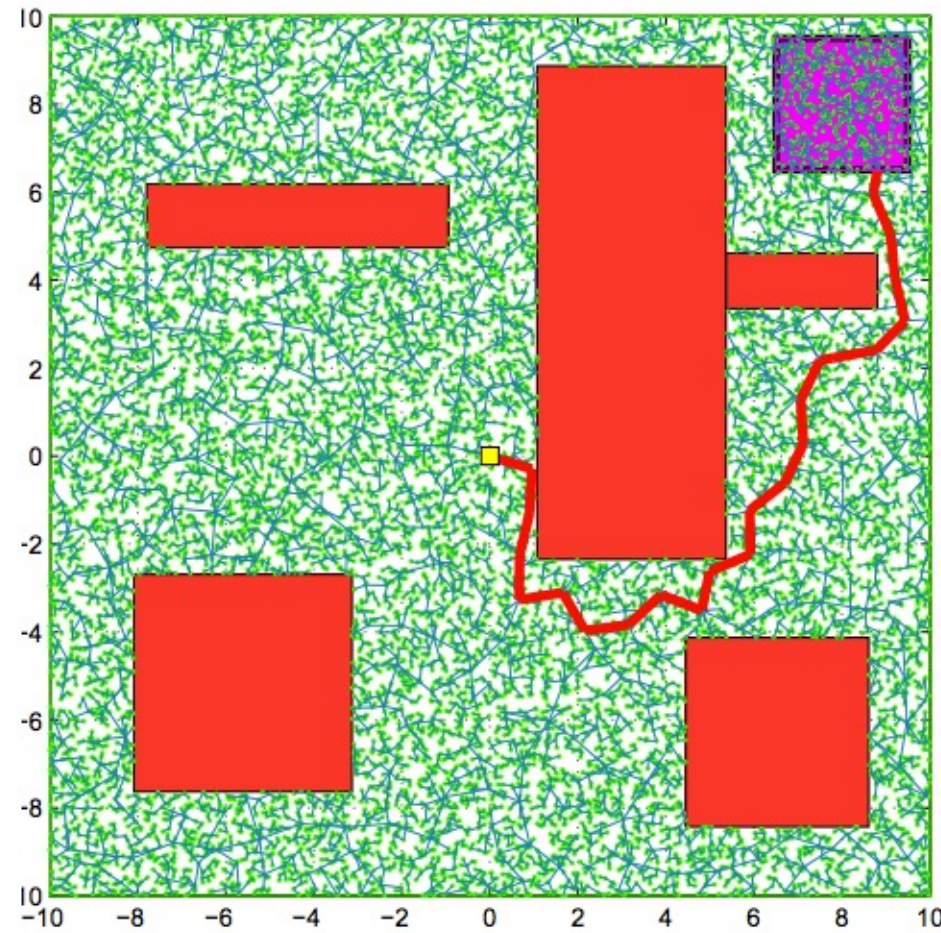


RRT*

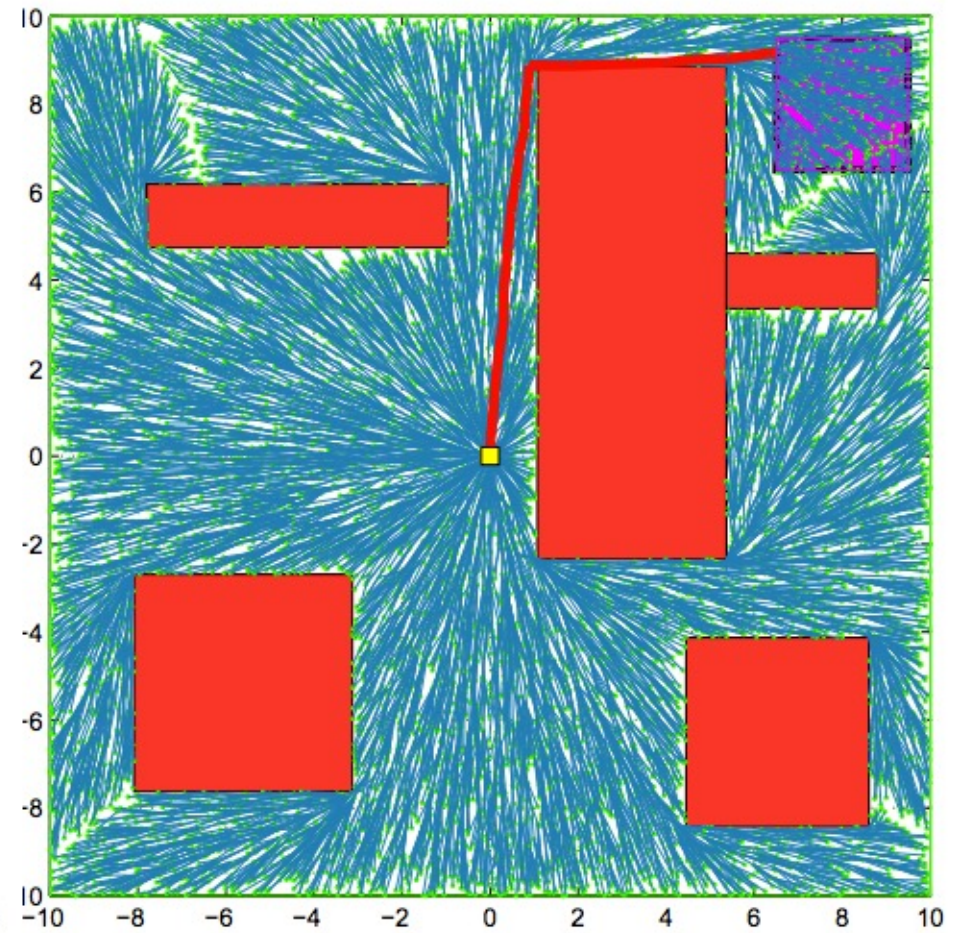


RRT*

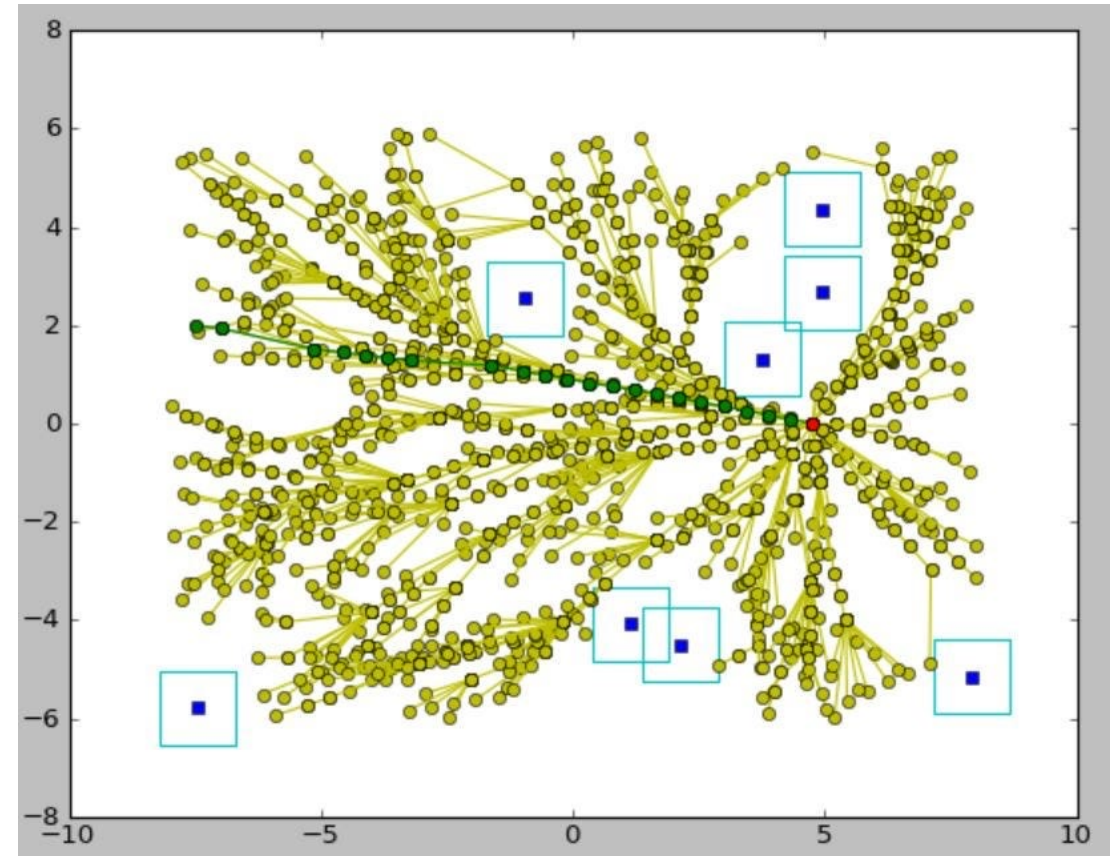
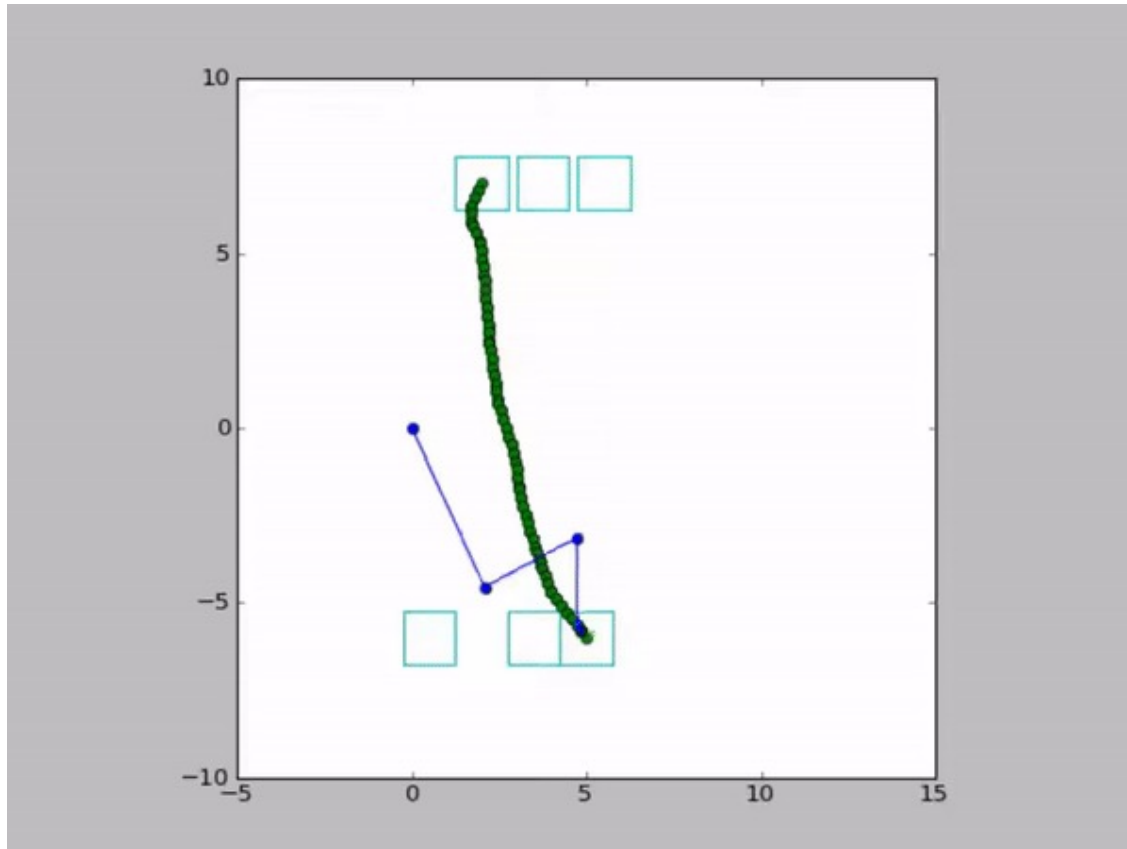
RRT



RRT*



RRT*



Theoretical properties of RRT*

■ Probabilistically Complete

Theorem 23 (Probabilistic completeness of RRG and RRT*) *The RRG and RRT* algorithms are probabilistically complete. Furthermore, for any robustly feasible path planning problem $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$, there exist constants $a > 0$ and $n_0 \in \mathbb{N}$, both dependent only on $\mathcal{X}_{\text{free}}$ and $\mathcal{X}_{\text{goal}}$, such that*

$$\mathbb{P}(\{V_n^{\text{RRG}} \cap \mathcal{X}_{\text{goal}} \neq \emptyset\}) > 1 - e^{-an}, \quad \forall n > n_0,$$

and

$$\mathbb{P}(\{V_n^{\text{RRT}^*} \cap \mathcal{X}_{\text{goal}} \neq \emptyset\}) > 1 - e^{-an}, \quad \forall n > n_0.$$

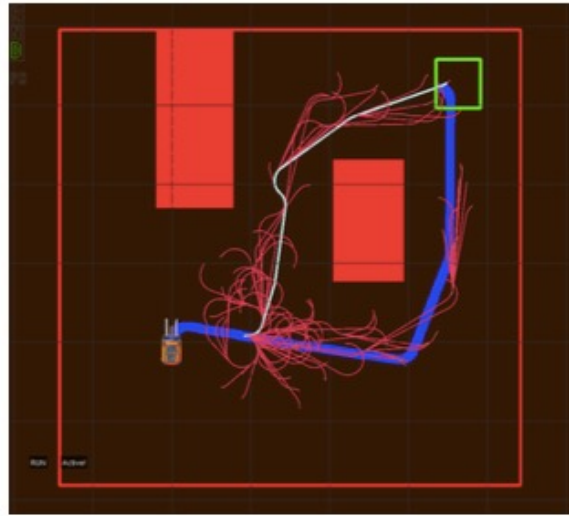
■ Asymptotically optimal

Theorem 38 (Asymptotic optimality of RRT*) *If $\gamma_{\text{RRT}^*} > (2(1+1/d))^{1/d} \left(\frac{\mu(\mathcal{X}_{\text{free}})}{\zeta_d}\right)^{1/d}$, then the RRT* algorithm is asymptotically optimal.*

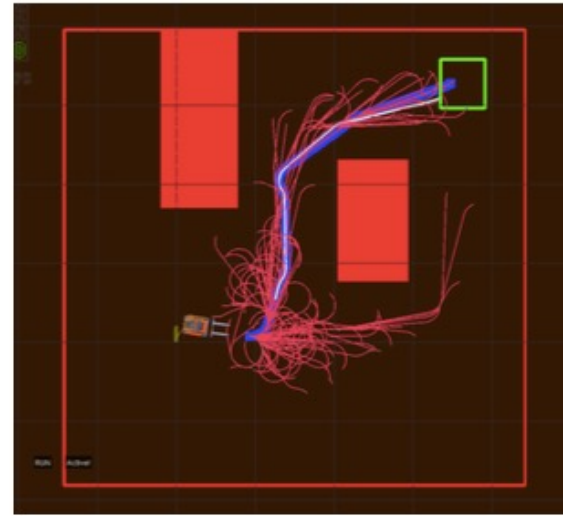
■ Efficient

Lemma 42 (PRM*, RRG, and RRT*) $M_n^{\text{PRM}^*}, M_n^{\text{RRG}}, M_n^{\text{RRT}^*} \in O(\log n)$.

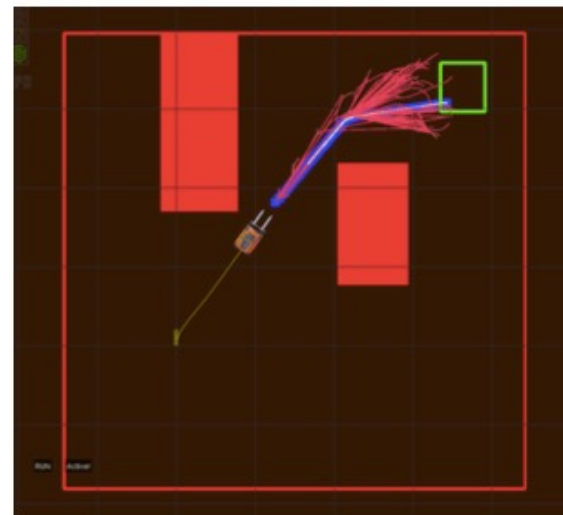
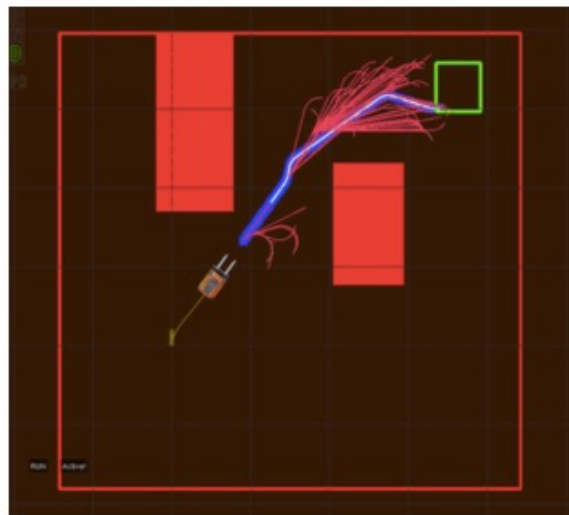
Real Time RRT*



(a) RRT* run 1



(b) RRT* run 1



Combining the best of A* and RRT*

Batch Informed Trees (BIT*)

Sampling-based Optimal Planning
via the Heuristically Guided Search
of Implicit Random Geometric Graphs

Jonathan D. Gammell¹,
Siddhartha S. Srinivasa², and Timothy D. Barfoot¹



¹ Institute for Aerospace Studies
UNIVERSITY OF TORONTO

²

Carnegie Mellon
THE ROBOTICS INSTITUTE

Post Processing for Motion Planning

Randomized motion planners tend to find not so great paths for execution: very jagged, often much longer than necessary.

→ In practice: do smoothing before using the path

- Shortcutting:

- along the found path, pick two vertices x_{t_1} , x_{t_2} and try to connect them directly (skipping over all intermediate vertices)

- Nonlinear optimization for optimal control

- Allows to specify an objective function that includes smoothness in state, control, small control inputs, etc.

Maxim Likhachev on A*/D* vs PRM/RRT

- Sampling-based methods are typically much easier to get working. One of the great things about RRT is that it doesn't require careful discretization of the action space and instead takes advantage of an extend operator (i.e., local controller or an interpolation function) which naturally exists in most robotics systems
- For planning in a continuous space, when comparing a quick implementation of RRT and a quick implementation of Anytime version of A*, RRT is typically much faster due to sparse exploration of a space.
- A* and its variants are typically harder to implement because they require a) careful design of discretization of the state-space and action-space (to make sure edges land where they are supposed to land); b) careful design of the heuristic function to guide the search well.

Maxim Likhachev on A*/D* vs PRM/RRT

- A* and its variants (including anytime variants) typically generate better quality solutions and very consistent solutions (similar solutions for similar queries) which is beneficial in many domains.
- A* and its variants can often be made nearly as fast as RRT and sometimes even faster if one analyzes the robotic system well to derive a powerful heuristic function. Many robotic systems have natural low-dimensional manifolds (e.g., a 3D workspace for example) that can be used to derive such heuristic functions.
- A* and its variants can be applied to both discrete and continuous (as well as hybrid) systems, whereas sampling-based systems tend to be more suitable for continuous systems since they rely on the idea of sparse exploration. (Within the same point, it should be noted that A* and its variants apply to PRMs and its variants. PRM is just a particular graph representation of the environment.)

Maxim Likhachev on A*/D* vs PRM/RRT

- In summary, I think for continuous planning problems, A* and its variants require substantially more development efforts (careful analysis of the system to derive proper graph representation and a good heuristic function) but can result in a better performance (similar speed but better quality solutions and more consistent behavior).

Lecture Outline

From Motion Planning to Control

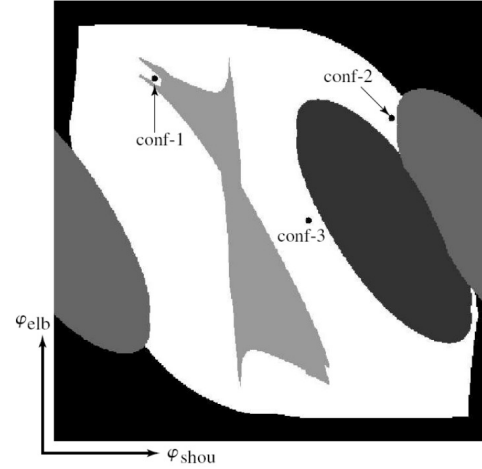
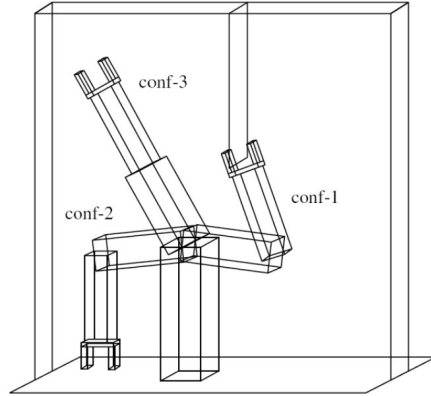


PID Control



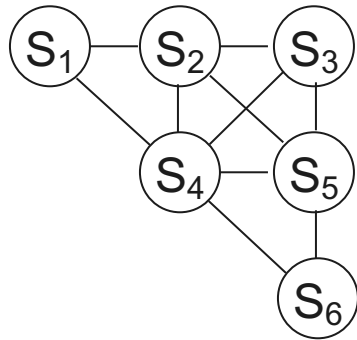
LQR

Motion Planning at a Glance

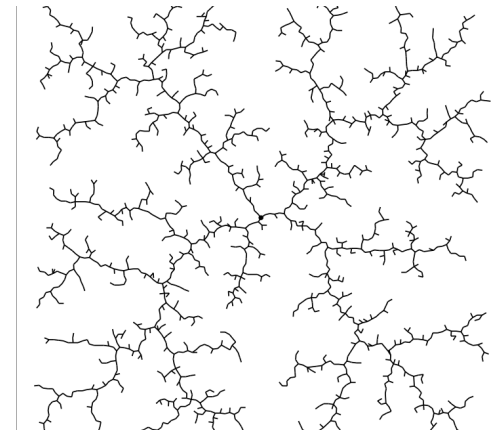


Returns a series of valid configurations from start to goal

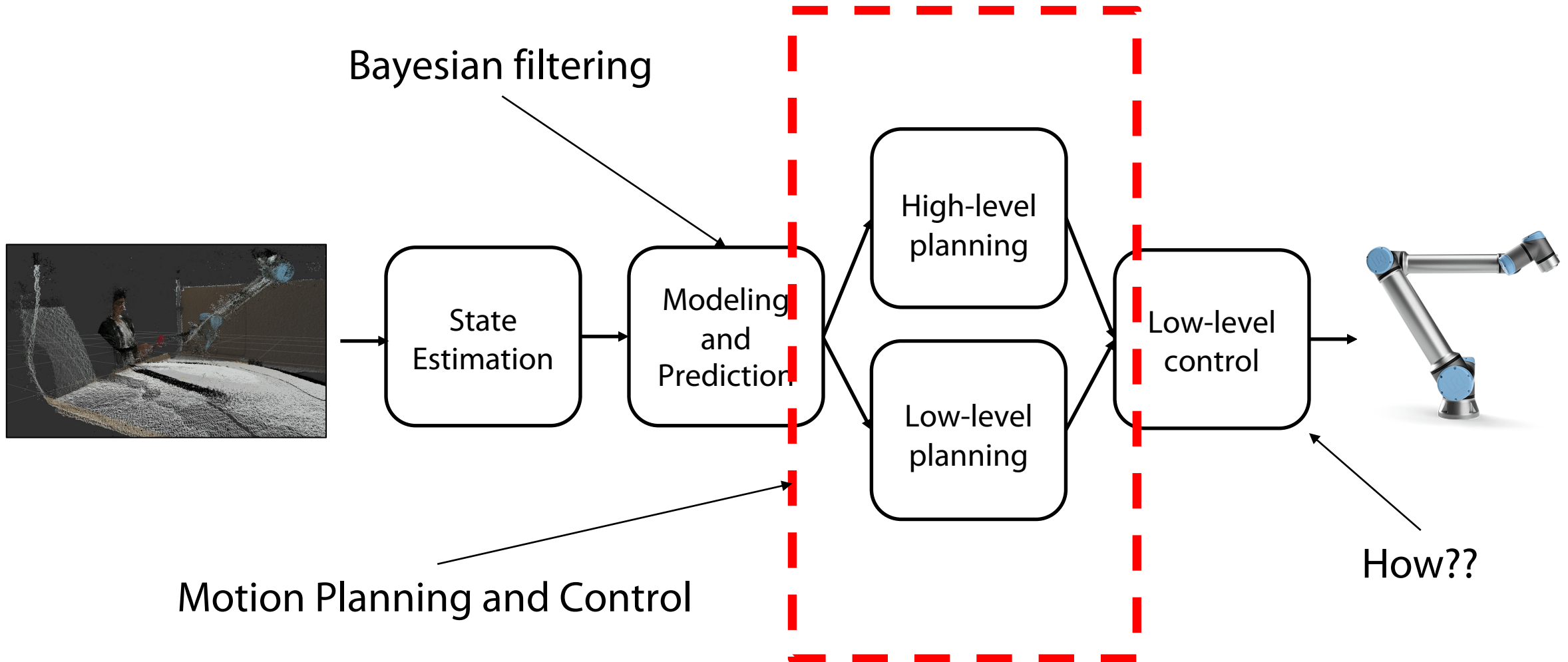
Deterministic Planning



Sampling Based Planning

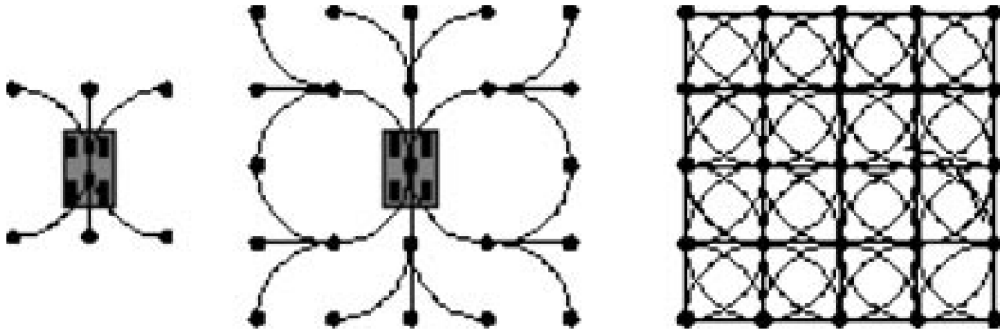


How do we actually execute on a robot?



Kinodynamic Motion Planning

Requires solving a 2 point boundary value problem on dynamics



s_0 and s_1 are connected if there exists a u such that $|f(s_0, u) - s_1| < \varepsilon$

$$\begin{aligned} \min_{u,x} \quad & \|u\| \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \quad \forall t \\ & u_t \in \mathcal{U}_t \\ & x_t \in \mathcal{X}_t \\ & x_0 = x_S \\ & x_T = x_G \end{aligned}$$

Gives us u for every connection, not just configurations

→ Can execute on the robot

Kinodynamic Motion Planning

$$\begin{aligned} \min_{u,x} \quad & \|u\| \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \quad \forall t \\ & u_t \in \mathcal{U}_t \\ & x_t \in \mathcal{X}_t \\ & x_0 = x_S \\ & x_T = x_G \end{aligned}$$

Gives us u for every connection, not just configurations

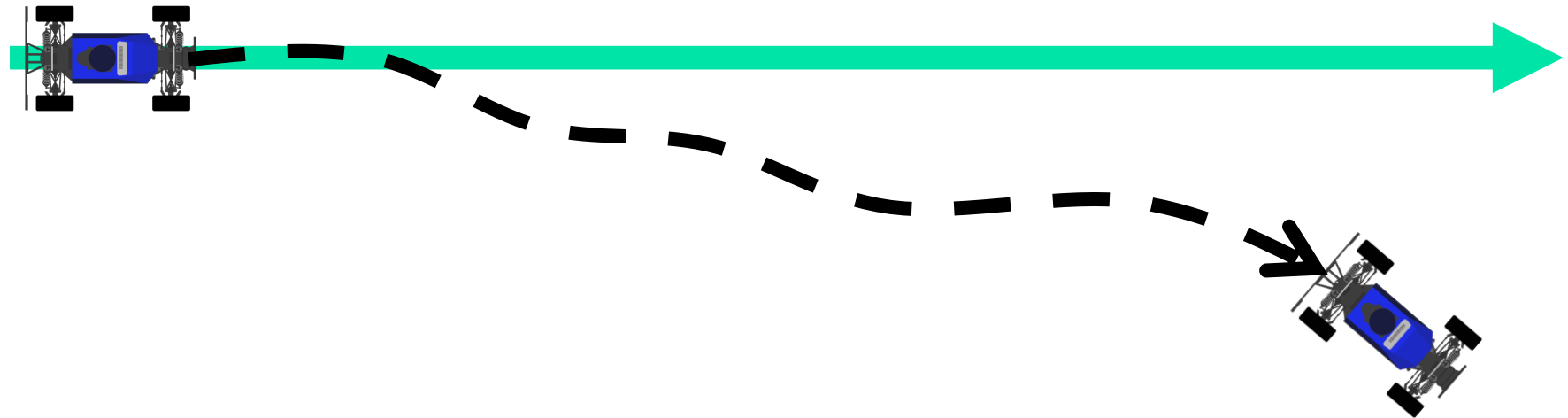
→ Can execute on the robot

Why is this not enough?

Pitfalls of Open-Loop Control

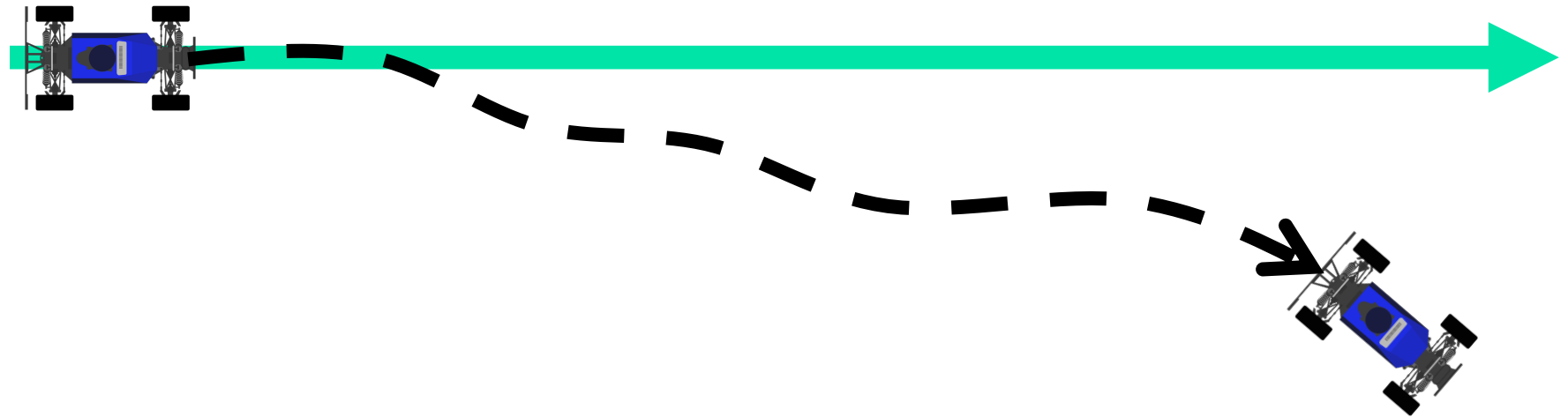
$$\begin{aligned} \min_{u,x} \quad & \|u\| \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \quad \forall t \\ & u_t \in \mathcal{U}_t \\ & x_t \in \mathcal{X}_t \\ & x_0 = x_S \\ & x_T = x_G \end{aligned}$$

- Drifts off the path with no corrections
- Doesn't account for costs like smoothness / preferences etc

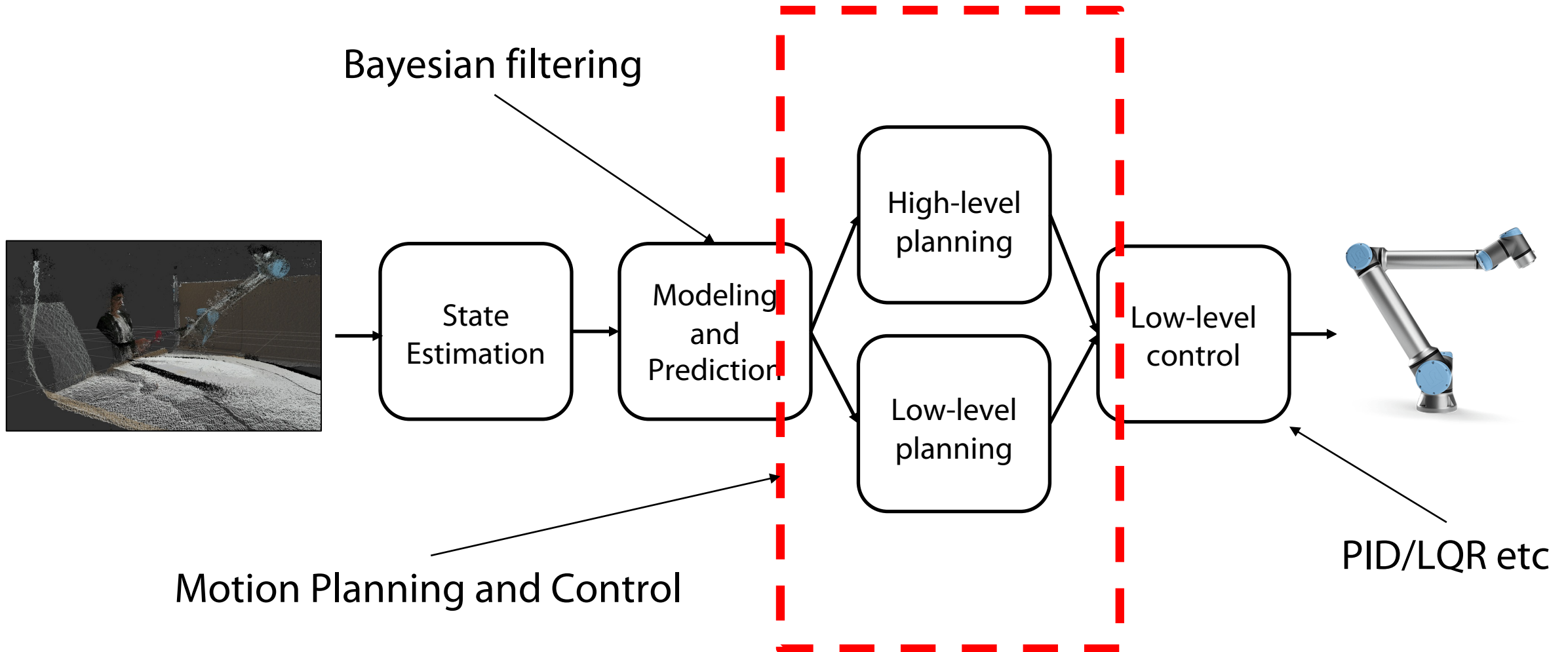


From Motion Planning to Control

- Feedback Control (eg PID) → ■ Drifts off the path with no corrections
- Optimal Control (eg LQR) → ■ Doesn't account for costs like smoothness / preferences etc



Where does this fit in the stack?

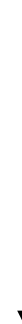


Lecture Outline

From Motion Planning to Control



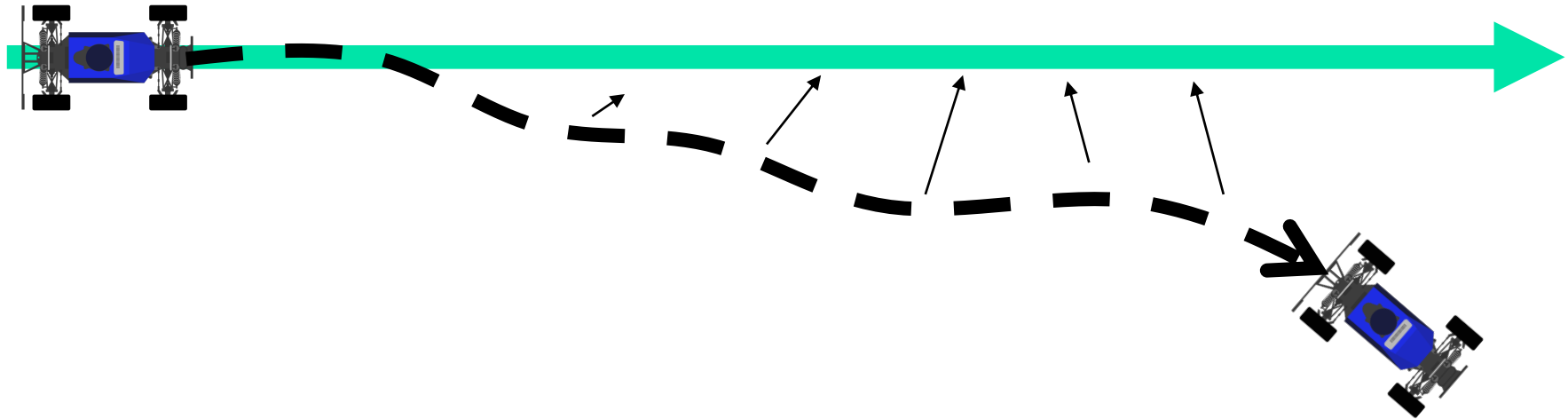
PID Control



LQR

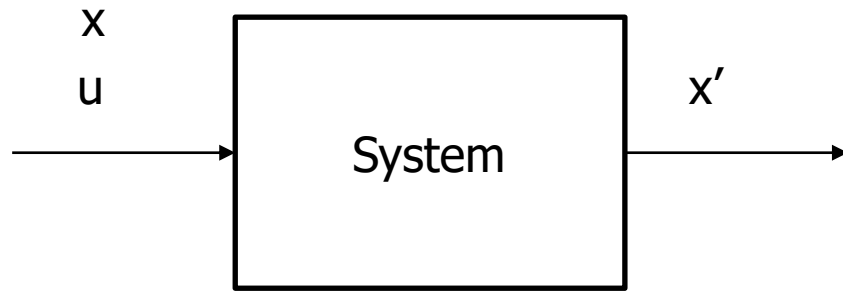
Goal of PID Control

- Obtain controller that stabilizes around a particular “setpoint”

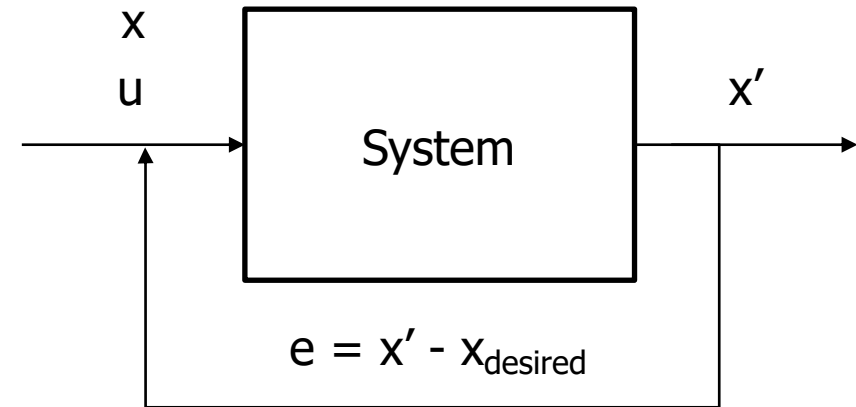


Going from Open Loop to Feedback Control

Open-loop control



Closed-loop control



Closed loop (feedback) control allows for correction of modeling errors and disturbances

How should we do feedback control?

Compute control action based on instantaneous error

$$u = K(\mathbf{x}, e)$$

control

state error

(steering angle, speed)

Apply control action, robot moves a bit, compute new error, repeat

Different laws have different trade-offs

Bang-bang control

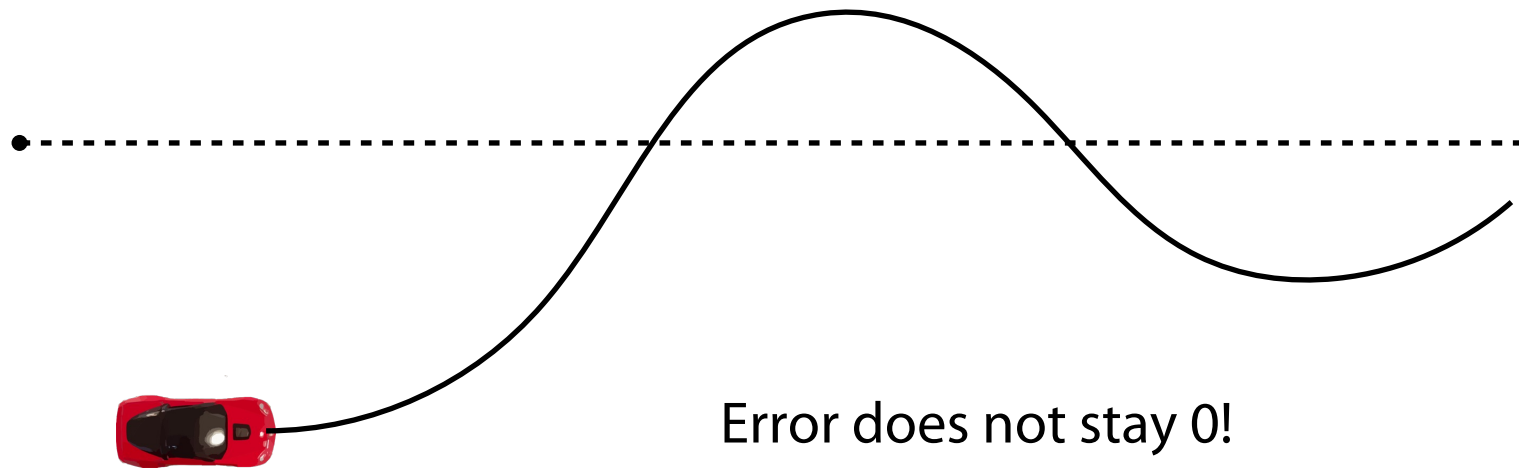
Simple control law - choose between hard left and hard right



$$u = \begin{cases} u_{max} & \text{if } e_{ct} < 0 \\ -u_{max} & \text{otherwise} \end{cases}$$

Bang-bang control

What happens when we run this control?



Need to adapt the magnitude of control proportional to the error ...

Proportional-Integral-Derivative (PID) Controller

- One of the most popular controllers in practice!
- Used widely in industrial control since 1900s (regulating temp., speed, etc.)

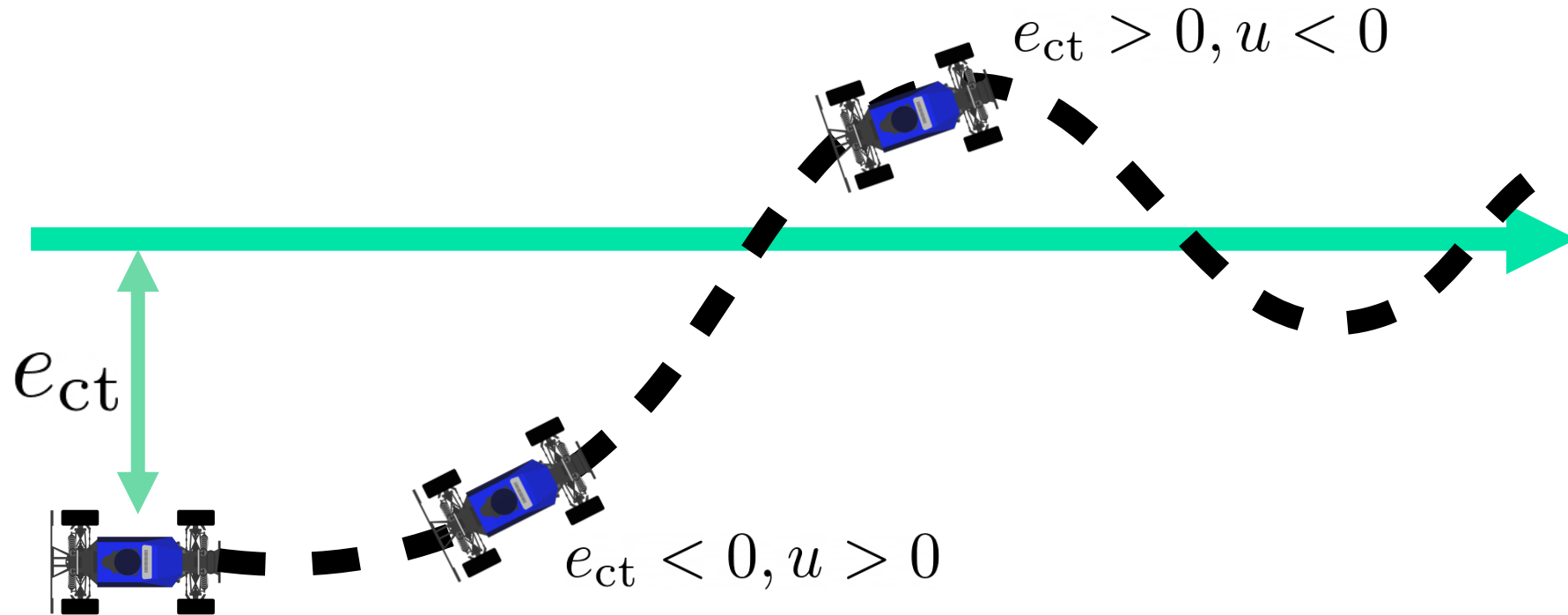
$$u = - \left(K_p e_{ct} + K_i \int e_{ct} dt + K_d \dot{e}_{ct} \right)$$

PROPORTIONAL
(PRESENT)

INTEGRAL
(PAST)

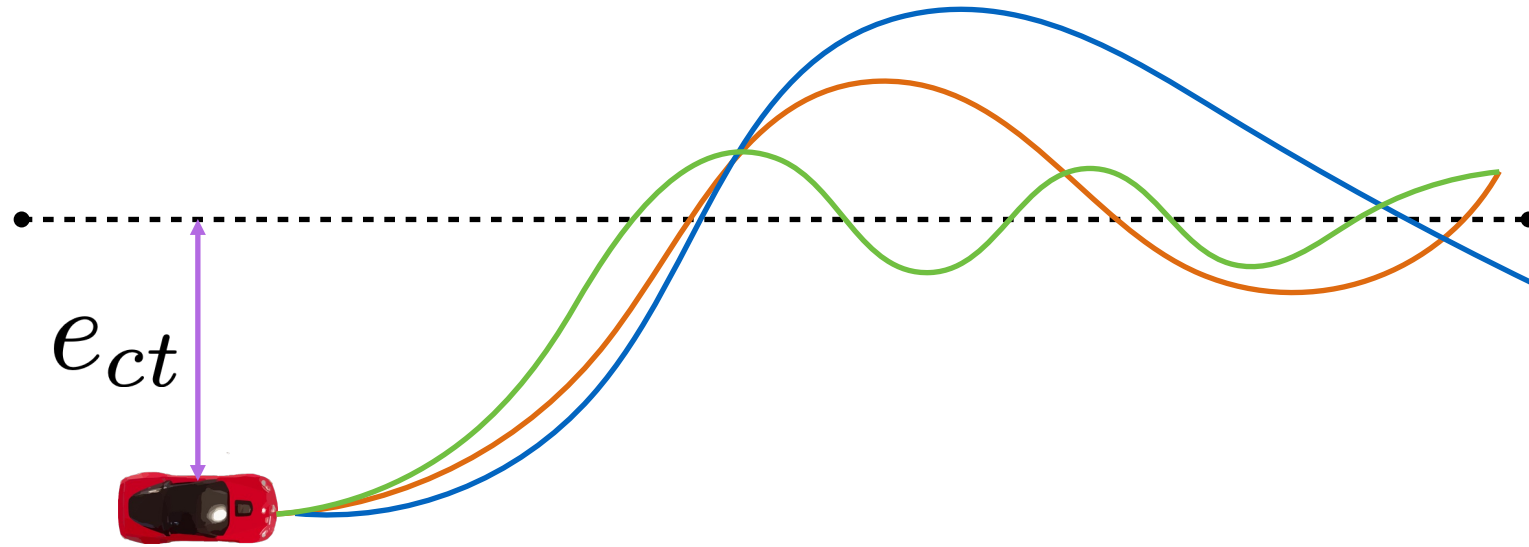
DERIVATIVE
(FUTURE)

Proportional Control



$$u = -K_p e_{ct}$$

The proportional gain matters!

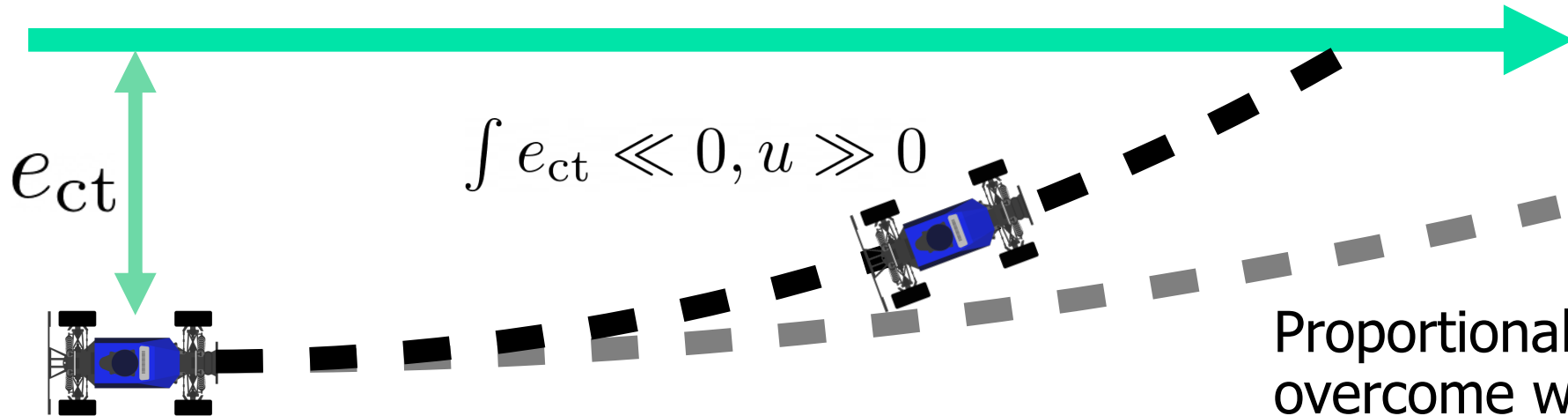
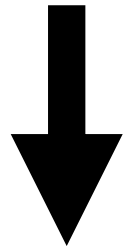


What happens when gain is low?

What happens when gain is high?

Proportional Integral (PI) Control

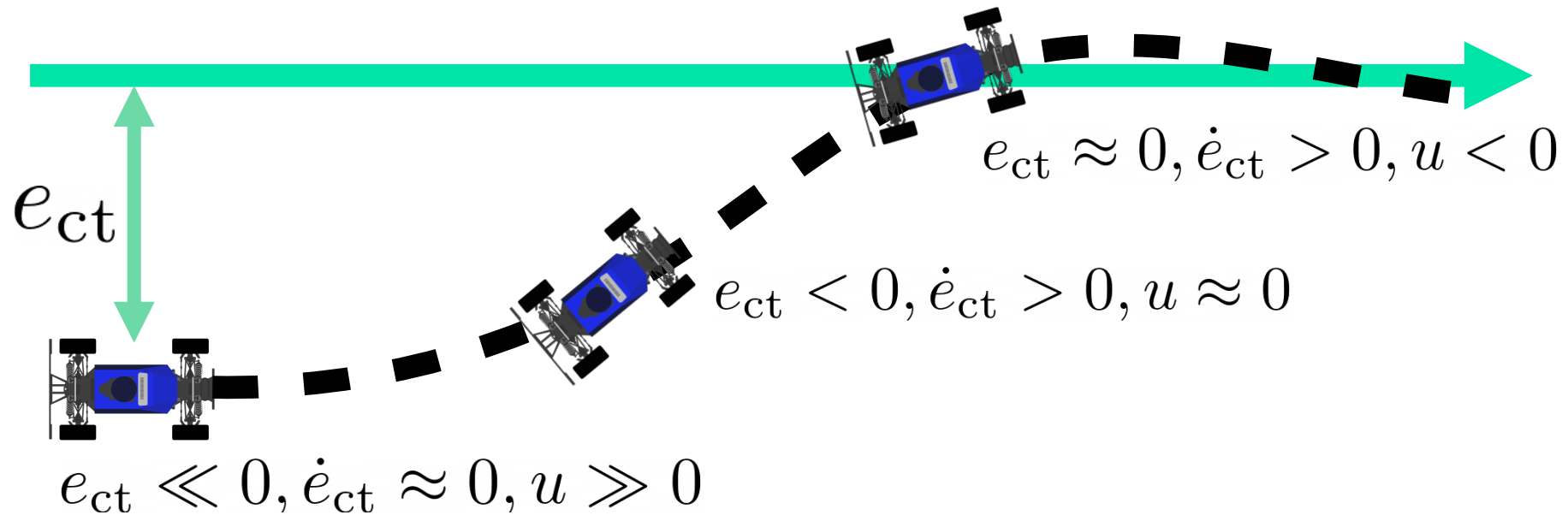
WIND



$$u = - \left(K_p e_{ct} + K_i \int e_{ct} dt \right)$$

Proportional Derivative (PD) Control

we want to minimize the magnitude of the overshoot



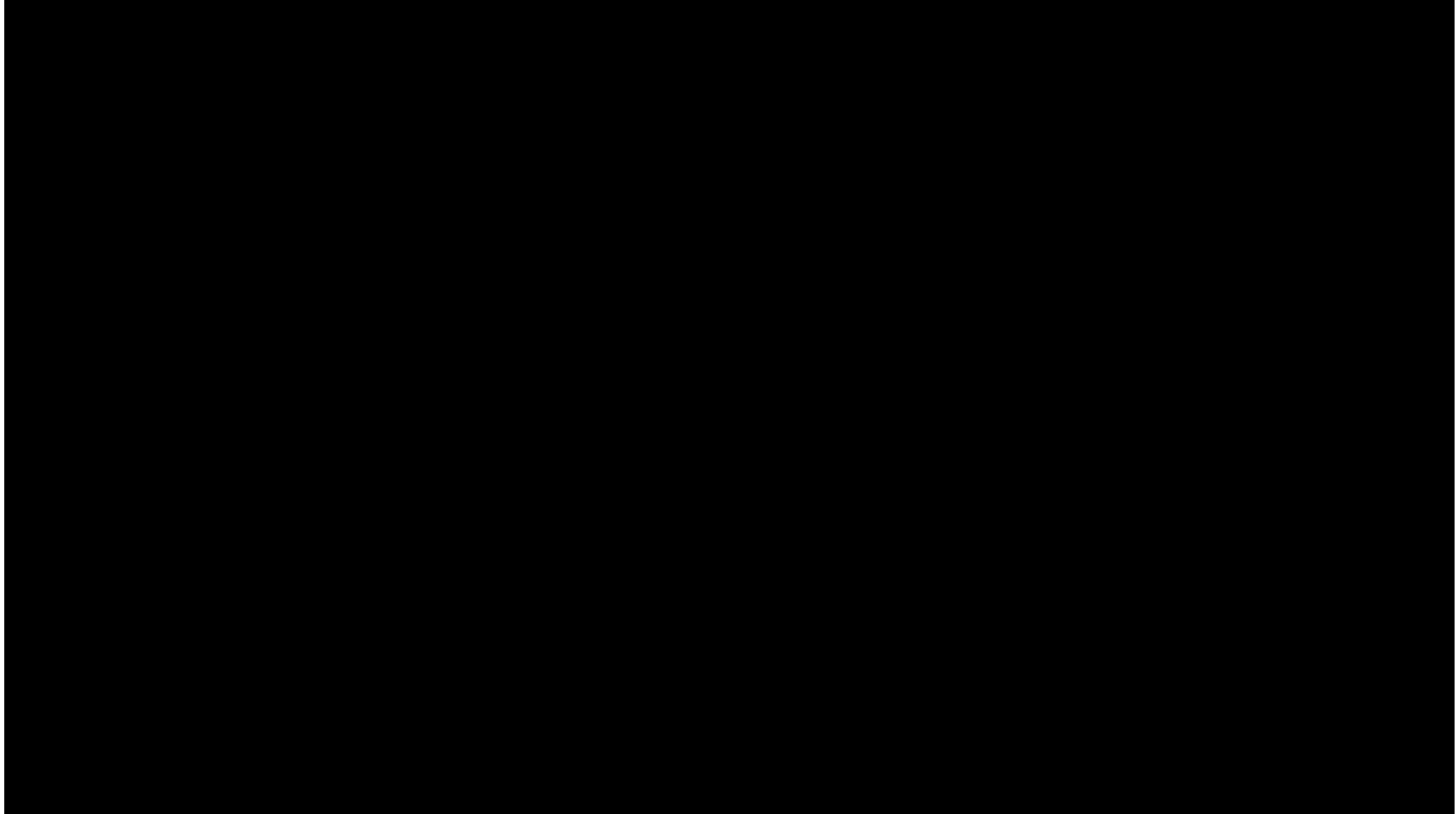
$$u = - (K_p e_{ct} + K_d \dot{e}_{ct})$$

PID Intuition

$$u = - \left(\underbrace{K_p e_{ct}}_{\substack{\text{PROPORTIONAL} \\ \text{(PRESENT)}}} + \underbrace{K_i \int e_{ct} dt}_{\substack{\text{INTEGRAL} \\ \text{(PAST)}}} + \underbrace{K_d \dot{e}_{ct}}_{\substack{\text{DERIVATIVE} \\ \text{(FUTURE)}}} \right)$$

- Proportional: minimize the current error!
- Integral: if I'm accumulating error, try harder!
- Derivative: if I'm going to overshoot, slow down!

PID Control in Action

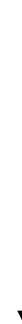


Lecture Outline

From Motion Planning to Control



PID Control



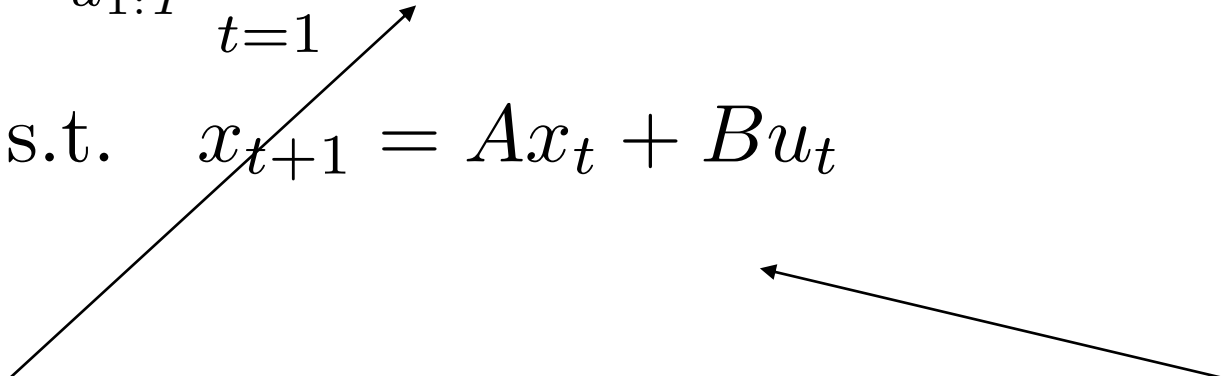
LQR

Goal of Optimal Control

- Minimize sum of costs, subject to dynamics and other constraints

$$\min_{u_{1:T}} \sum_{t=1}^T g(x_t, u_t) + G(x_T, u_T)$$

s.t. $x_{t+1} = Ax_t + Bu_t$

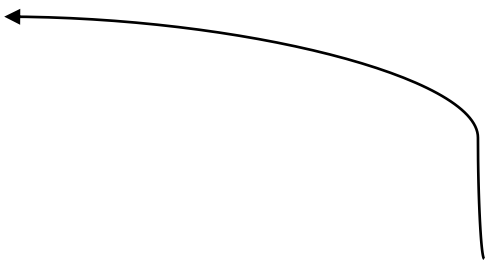
The diagram consists of two arrows. One arrow starts from the text 'Can be costs like smoothness, preferences, speed' and points diagonally upwards and to the right, ending at the summation term of the cost function in the equation above. The second arrow starts from the text 'Can be constraints like velocity/acceleration bounds' and points diagonally upwards and to the left, ending at the state transition equation below the cost function.

Can be costs like smoothness, preferences, speed

Can be constraints like velocity/acceleration bounds

From Motion Planning to PID to Optimal Control

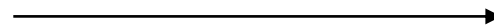
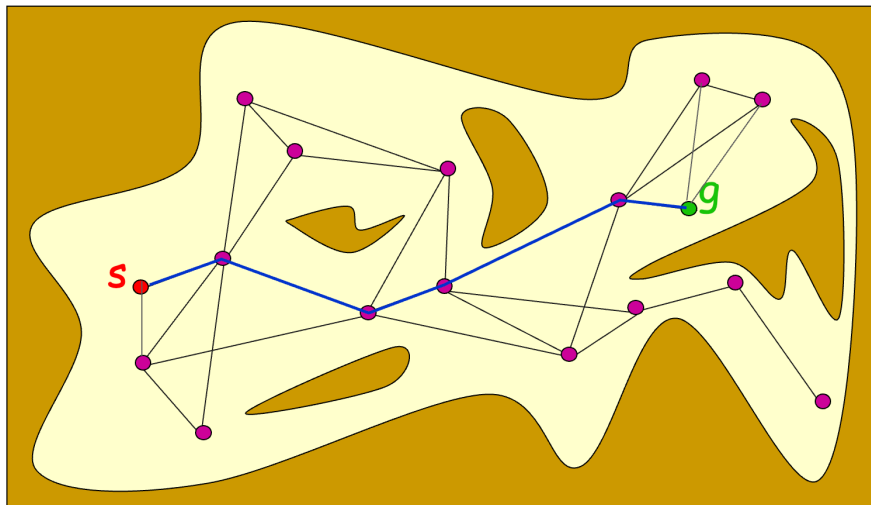
- Allows for integration of other costs/preferences
- Can be parameterized as closed loop

$$\begin{aligned} \min_{K_{1:t}} \quad & \sum_{t=1}^T g(x_t, u_t) + G(x_T, u_T) \\ \text{s.t.} \quad & x_{t+1} = Ax_t + Bu_t \\ & u_t = K_t(x_t) \end{aligned}$$


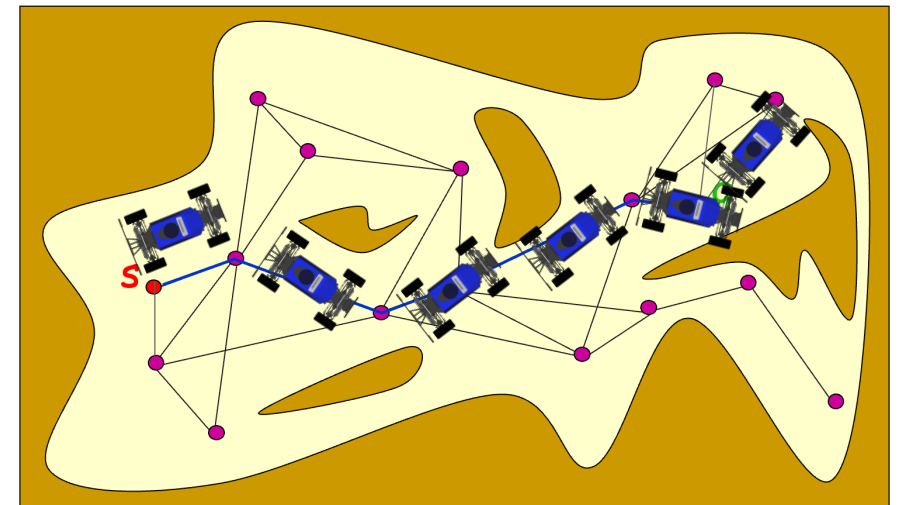
- Allows for "optimal" closed-loop controllers, stable under disturbances

From Motion Planning to PID to Optimal Control

- Motion planning and optimal control are not orthogonal → complementary
- Can set costs for optimal control to track motion planning solution



Seed cost for OC



Tractable Optimal Control Problem - LQR

- Optimal Control for Linear Dynamical Systems and Quadratic Cost (aka LQ setting, or LQR setting)
 - Very special case: can solve continuous state-space optimal control problem exactly and only requires performing linear algebra operations
 - Running time: $O(H n^3)$

Note 1: Great reference [optional] Anderson and Moore, Linear Quadratic Methods

Note2 : Strong similarity with Kalman filtering, which is able to compute the Bayes' filter updates exactly even though in general there are no closed form solutions and numerical solutions scale poorly with dimensionality.

Linear Quadratic Regulator (LQR)

The LQR setting assumes a linear dynamical system:

$$x_{t+1} = Ax_t + Bu_t,$$

x_t : state at time t

u_t : input at time t

It assumes a quadratic cost function:

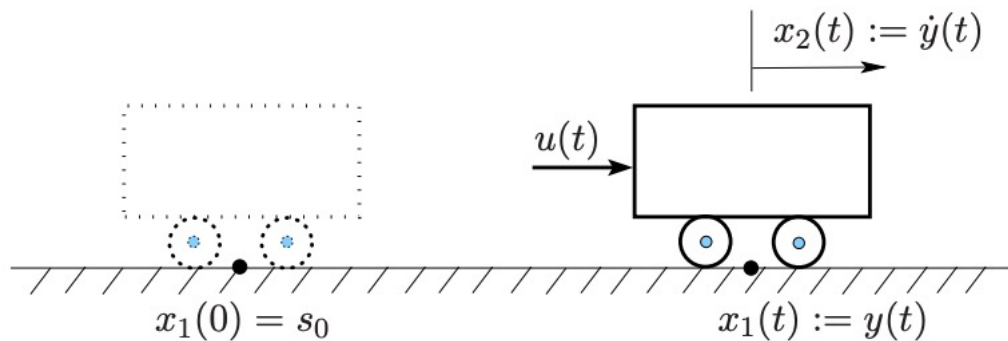
$$g(x_t, u_t) = x_t^\top Qx_t + u_t^\top Ru_t$$

with $Q \succ 0, R \succ 0$.

For a square matrix X we have $X \succ 0$ if and only if for all vectors z we have $z^\top Xz > 0$. Hence there is a non-zero cost for any state different from the all-zeros state, and any input different from the all-zeros input.

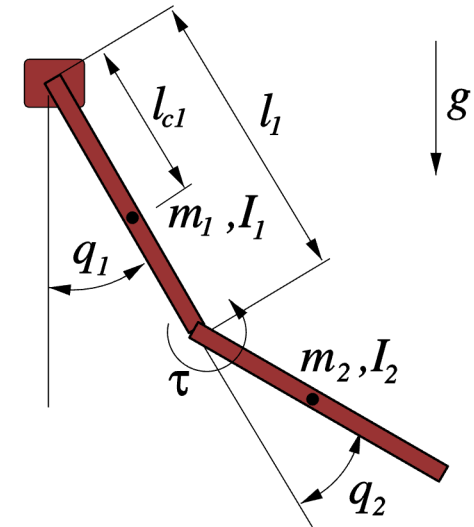
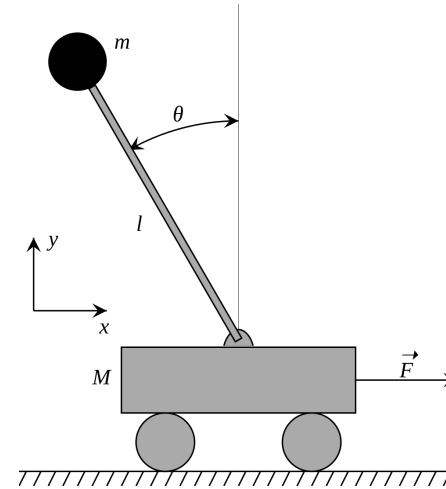
Examples of LQR systems

Double Integrator

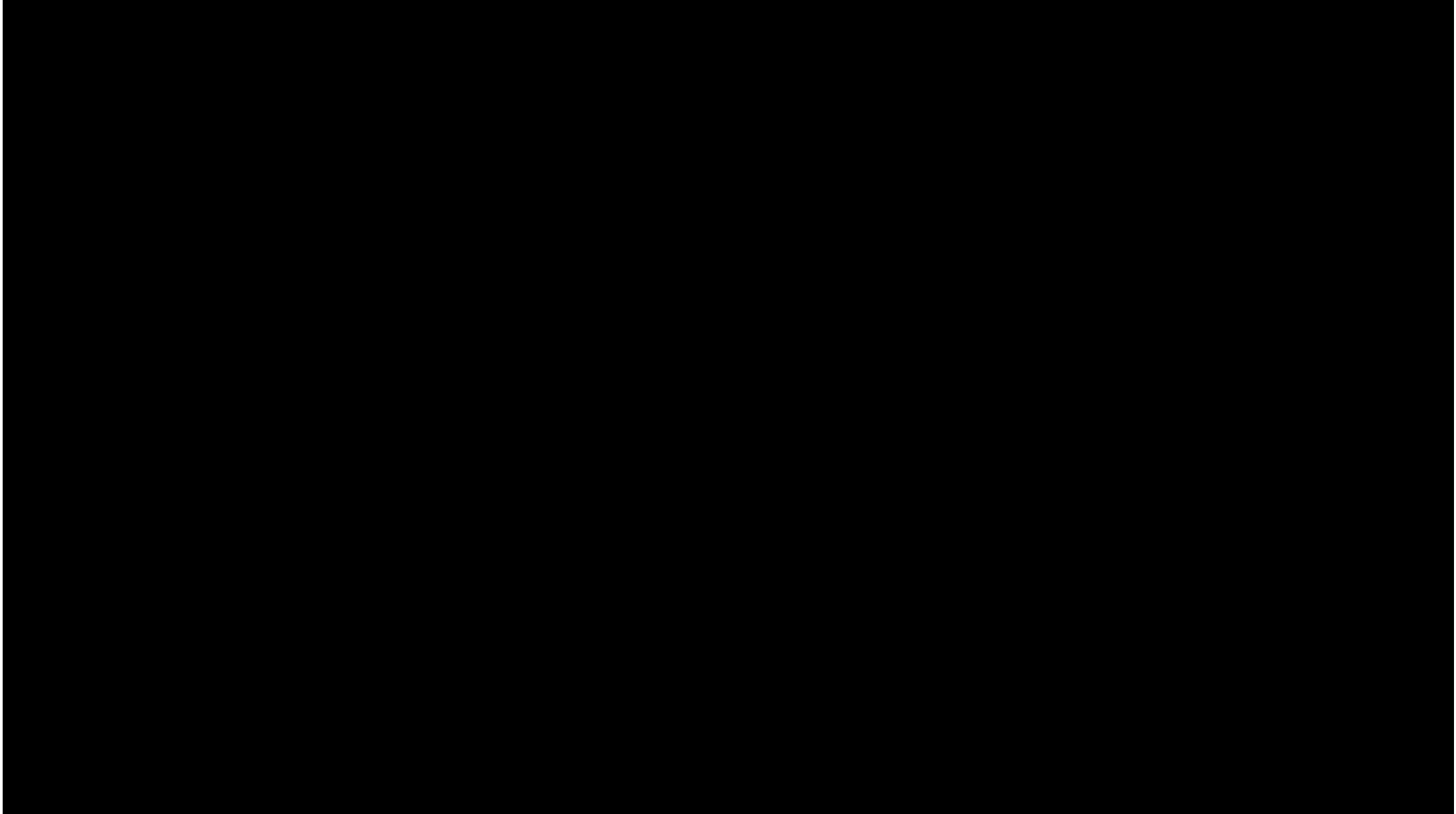


$$\ddot{q} = u(t) \quad \dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{u}(t)$$
$$y = q(t) \quad \mathbf{y}(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}(t).$$

Many systems can be easily linearized



Extension to Non-Linear Systems



LQR Problem Statement

- Minimize sum of costs subject to linear dynamics constraints

$$\min_{u_{1:T}} \sum_{t=1}^T g(x_t, u_t) + G(x_T, u_T)$$

$$\text{s.t. } x_{t+1} = Ax_t + Bu_t$$

$$\min_{K_{1:t}} \sum_{t=1}^T g(x_t, u_t) + G(x_T, u_T)$$

$$\text{s.t. } x_{t+1} = Ax_t + Bu_t$$

$$u_t = K_t(x_t)$$

Recursive Solution to LQR Problem

- Back-up step for $i+1$ steps to go:

$$J_{i+1}(s) = \min_u g(s, u) + \sum_{s'} P(s'|s, u) J_i(s')$$

- LQR:

$$\begin{aligned} J_{i+1}(x) &= \min_u x^\top Qx + u^\top Ru + \sum_{x'=Ax+Bu} J_i(x') \\ &= \min_u [x^\top Qx + u^\top Ru + J_i(Ax + Bu)] \end{aligned}$$

LQR value iteration: J_1

$$J_{i+1}(x) \leftarrow \min_u [x^\top Qx + u^\top Ru + J_i(Ax + Bu)]$$

Initialize $J_0(x) = x^\top P_0x$.

$$\begin{aligned} J_1(x) &= \min_u [x^\top Qx + u^\top Ru + J_0(Ax + Bu)] \\ &= \min_u [x^\top Qx + u^\top Ru + (Ax + Bu)^\top P_0(Ax + Bu)] \quad (1) \end{aligned}$$

To find the minimum over u , we set the gradient w.r.t. u equal to zero:

$$\nabla_u [\dots] = 2Ru + 2B^\top P_0(Ax + Bu) = 0,$$

$$\text{hence: } u = -(R + B^\top P_0B)^{-1} B^\top P_0Ax \quad (2)$$

$$\begin{aligned} (2) \text{ into } (1): J_1(x) &= x^\top P_1x \\ \text{for: } P_1 &= Q + K_1^\top RK_1 + (A + BK_1)^\top P_0(A + BK_1) \\ K_1 &= -(R + B^\top P_0B)^{-1} B^\top P_0A. \end{aligned}$$

LQR value iteration: J_1 (ctd)

- In summary:

$$\begin{aligned}J_0(x) &= x^\top P_0 x \\x_{t+1} &= Ax_t + Bu_t \\g(x, u) &= u^\top Ru + x^\top Qx\end{aligned}$$

$$\begin{aligned}J_1(x) &= x^\top P_1 x \\ \text{for: } P_1 &= Q + K_1^\top RK_1 + (A + BK_1)^\top P_0 (A + BK_1) \\ K_1 &= -(R + B^\top P_0 B)^{-1} B^\top P_0 A.\end{aligned}$$

- $J_1(x)$ is quadratic, just like $J_0(x)$.

→ Value iteration update is the same for all times and can be done in closed form for this particular continuous state-space system and cost!

$$\begin{aligned}J_2(x) &= x^\top P_2 x \\ \text{for: } P_2 &= Q + K_2^\top RK_2 + (A + BK_2)^\top P_1 (A + BK_2) \\ K_2 &= -(R + B^\top P_1 B)^{-1} B^\top P_1 A.\end{aligned}$$

Value iteration solution to LQR

Set $P_0 = 0$.
for $i = 1, 2, 3, \dots$

$$\begin{aligned}K_i &= -(R + B^\top P_{i-1} B)^{-1} B^\top P_{i-1} A \\P_i &= Q + K_i^\top R K_i + (A + B K_i)^\top P_{i-1} (A + B K_i)\end{aligned}$$

The optimal policy for a i -step horizon is given by:

$$\pi(x) = K_i x$$

The cost-to-go function for a i -step horizon is given by:

$$J_i(x) = x^\top P_i x.$$

LQR assumptions revisited

$$\begin{aligned}x_{t+1} &= Ax_t + Bu_t \\g(x_t, u_t) &= x_t^\top Qx_t + u_t^\top Ru_t\end{aligned}$$

= for keeping a linear system at the all-zeros state while preferring to keep the control input small.

- Extensions make it more generally applicable:
 - Affine systems
 - Systems with stochasticity
 - Penalization for change in control inputs
 - Linear time varying (LTV) systems
 - Trajectory following for non-linear systems

LQR Ext0: Affine systems

$$\begin{aligned}x_{t+1} &= Ax_t + Bu_t + c \\g(x_t, u_t) &= x_t^\top Qx_t + u_t^\top Ru_t\end{aligned}$$

- Optimal control policy remains linear, optimal cost-to-go function remains quadratic
- Two avenues to do derivation:
 - 1. Re-derive the update, which is very similar to what we did for standard setting
 - 2. Re-define the state as: $z_t = [x_t; 1]$, then we have:

$$z_{t+1} = \begin{bmatrix} x_{t+1} \\ 1 \end{bmatrix} = \begin{bmatrix} A & c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ 1 \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_t = A'z_t + B'u_t$$

LQR Ext1: stochastic system

$$\begin{aligned}x_{t+1} &= Ax_t + Bu_t + w_t \\g(x_t, u_t) &= x_t^\top Qx_t + u_t^\top Ru_t \\w_t, t = 0, 1, \dots &\text{are zero mean and independent}\end{aligned}$$

- Exercise: work through similar derivation as we did for the deterministic case, but which will now have expectations.
- Result:
 - Same optimal control policy
 - Cost-to-go function is almost identical: has one additional term which depends on the variance in the noise (and which cannot be influenced by the choice of control inputs)

LQR Ext2: non-linear systems

Nonlinear system: $x_{t+1} = f(x_t, u_t)$

We can keep the system at the state x^* iff

$$\exists u^* \text{ s.t. } x^* = f(x^*, u^*)$$

Linearizing the dynamics around x^* gives:

$$x_{t+1} \approx f(x^*, u^*) + \underbrace{\frac{\partial f}{\partial x}(x^*, u^*)}_{\mathbf{A}}(x_t - x^*) + \underbrace{\frac{\partial f}{\partial u}(x^*, u^*)}_{\mathbf{B}}(u_t - u^*)$$

Equivalently:

$$x_{t+1} - x^* \approx \mathbf{A}(x_t - x^*) + \mathbf{B}(u_t - u^*)$$

Let $z_t = x_t - x^*$, let $v_t = u_t - u^*$, then:

$$z_{t+1} = \mathbf{A}z_t + \mathbf{B}v_t, \quad \text{cost} = z_t^\top \mathbf{Q}z_t + v_t^\top \mathbf{R}v_t \quad [= \text{standard LQR}]$$

$$v_t = \mathbf{K}z_t \Rightarrow u_t - u^* = \mathbf{K}(x_t - x^*) \Rightarrow u_t = u^* + \mathbf{K}(x_t - x^*)$$

LQR Ext3: Penalize for Change in Control Inputs

- Standard LQR:

$$\begin{aligned}x_{t+1} &= Ax_t + Bu_t \\g(x_t, u_t) &= x_t^\top Qx_t + u_t^\top Ru_t\end{aligned}$$

- When run in this format on real systems: often high frequency control inputs get generated. Typically highly undesirable and results in poor control performance.
- Why?
- Simple special case which works well in practice: penalize for change in control inputs. ---- How ??

LQR Ext3: Penalize for Change in Control Inputs

- Standard LQR:

$$\begin{aligned}x_{t+1} &= Ax_t + Bu_t \\g(x_t, u_t) &= x_t^\top Qx_t + u_t^\top Ru_t\end{aligned}$$

- How to incorporate the change in controls into the cost/reward function?
 - Soln. method A: explicitly incorporate into the state by augmenting the state with the past control input vector, and the difference between the last two control input vectors.
 - Soln. method B: change of control input variables.

LQR Ext3: Penalize for Change in Control Inputs

- Standard LQR:

$$\begin{aligned}x_{t+1} &= Ax_t + Bu_t \\g(x_t, u_t) &= x_t^\top Qx_t + u_t^\top Ru_t\end{aligned}$$

- Introducing change in controls Δu :

$$\begin{aligned}\begin{bmatrix} x_{t+1} \\ u_t \end{bmatrix} &= \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} x_t \\ u_{t-1} \end{bmatrix} + \begin{bmatrix} B \\ I \end{bmatrix} \Delta u_t \\ \underbrace{\begin{bmatrix} x_{t+1} \\ u_t \end{bmatrix}}_{x'_{t+1}} &= \underbrace{\begin{bmatrix} A & B \\ 0 & I \end{bmatrix}}_{A'} \underbrace{\begin{bmatrix} x_t \\ u_{t-1} \end{bmatrix}}_{x'_t} + \underbrace{\begin{bmatrix} B \\ I \end{bmatrix}}_{B'} \underbrace{\Delta u_t}_{u'_t}\end{aligned}$$

$$\text{cost} = -(x'^\top Q' x' + \Delta u^\top R' \Delta u)$$

$$Q' = \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix}$$

R' = penalty for change in controls

[If $R'=0$, then “equivalent” to standard LQR.]

LQR Ext4: Linear Time Varying (LTV) Systems

$$\begin{aligned}x_{t+1} &= A_t x_t + B_t u_t \\g(x_t, u_t) &= x_t^\top Q_t x_t + u_t^\top R_t u_t\end{aligned}$$

LQR Ext4: Linear Time Varying (LTV) Systems

Set $P_0 = 0$.

for $i = 1, 2, 3, \dots$

$$K_i = -(R_{H-i} + B_{H-i}^\top P_{i-1} B_{H-i})^{-1} B_{H-i}^\top P_{i-1} A_{H-i}$$

$$P_i = Q_{H-i} + K_i^\top R_{H-i} K_i + (A_{H-i} + B_{H-i} K_i)^\top P_{i-1} (A_{H-i} + B_{H-i} K_i)$$

The optimal policy for a i -step horizon is given by:

$$\pi(x) = K_i x$$

The cost-to-go function for a i -step horizon is given by:

$$J_i(x) = x^\top P_i x.$$

LQR Ext5: Trajectory Following for Non-Linear Systems

- A state sequence $x_0^*, x_1^*, \dots, x_H^*$ is a feasible target trajectory if and only if

$$\exists u_0^*, u_1^*, \dots, u_{H-1}^* : \forall t \in \{0, 1, \dots, H-1\} : x_{t+1}^* = f(x_t^*, u_t^*)$$

- Problem statement:

$$\min_{u_0, u_1, \dots, u_{H-1}} \sum_{t=0}^{H-1} (x_t - x_t^*)^\top Q (x_t - x_t^*) + (u_t - u_t^*)^\top R (u_t - u_t^*)$$

$$\text{s.t. } x_{t+1} = f(x_t, u_t)$$

- Transform into linear time varying case (LTV):

$$x_{t+1} \approx f(x_t^*, u_t^*) + \underbrace{\frac{\partial f}{\partial x}(x_t^*, u_t^*)}_{A_t} (x_t - x_t^*) + \underbrace{\frac{\partial f}{\partial u}(x_t^*, u_t^*)}_{B_t} (u_t - u_t^*)$$

$$x_{t+1} - x_{t+1}^* \approx A_t (x_t - x_t^*) + B_t (u_t - u_t^*)$$

LQR Ext5: Trajectory Following for Non-Linear Systems

- Transformed into linear time varying case (LTV):

$$\min_{u_0, u_1, \dots, u_{H-1}} \sum_{t=0}^{H-1} (x_t - x_t^*)^\top Q (x_t - x_t^*) + (u_t - u_t^*)^\top R (u_t - u_t^*)$$

$$\text{s.t. } x_{t+1} - x_{t+1}^* = A_t (x_t - x_t^*) + B_t (u_t - u_t^*)$$

- Now we can run the standard LQR back-up iterations.
- Resulting policy at i time-steps from the end:

$$u_{H-i} - u_{H-i}^* = K_i (x_{H-i} - x_{H-i}^*)$$

- The target trajectory need not be feasible to apply this technique, however, if it is infeasible then there will an offset term in the dynamics:

$$x_{t+1} - x_{t+1}^* = f(x_t, u_t) - x_{t+1}^* + A_t (x_t - x_t^*) + B_t (u_t - u_t^*)$$

Most General Case

- How about this general optimal control problem?

$$\min_{u_0, \dots, u_H} \sum_{t=0}^H g(x_t, u_t)$$

Iteratively Apply LQR

Initialize the algorithm by picking either (a) A control policy $\pi^{(0)}$ or (b) A sequence of states $x_0^{(0)}, x_1^{(0)}, \dots, x_H^{(0)}$ and control inputs $u_0^{(0)}, u_1^{(0)}, \dots, u_H^{(0)}$. With initialization (a), start in Step (1). With initialization (b), start in Step (2).

Iterate the following:

- (1) Execute the current policy $\pi^{(i)}$ and record the resulting state-input trajectory $x_0^{(i)}, u_0^{(i)}, x_1^{(i)}, u_1^{(i)}, \dots, x_H^{(i)}, u_H^{(i)}$.
- (2) Compute the LQ approximation of the optimal control problem around the obtained state-input trajectory by computing a first-order Taylor expansion of the dynamics model, and a second-order Taylor expansion of the cost function.
- (3) Use the LQR back-ups to solve for the optimal control policy $\pi^{(i+1)}$ for the LQ approximation obtained in Step (2).
- (4) Set $i = i + 1$ and go to Step (1).

Iterative LQR in Standard LTV Format

Standard LTV is of the form:

$$\begin{aligned} z_{t+1} &= A_t z_t + B_t v_t \\ g(z, v) &= z^\top Q z + v^\top R v \end{aligned}$$

Linearizing f around $(x_t^{(i)}, u_t^{(i)})$ from the roll-out in iteration i of the iterative LQR algorithm gives us:

$$x_{t+1} \approx f(x_t^{(i)}, u_t^{(i)}) + \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t - x_t^{(i)}) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t - u_t^{(i)})$$

Keeping in mind that $x_{t+1}^{(i)} = f(x_t^{(i)}, u_t^{(i)})$ gives us:

$$x_{t+1} - x_{t+1}^{(i)} \approx \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)})(x_t - x_t^{(i)}) + \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)})(u_t - u_t^{(i)})$$

Hence we get the standard format if using:

$$\begin{aligned} z_t &= \begin{bmatrix} x_t - x_t^{(i)} & 1 \end{bmatrix}^\top \\ v_t &= (u_t - u_t^{(i)}) \\ A_t &= \begin{bmatrix} \frac{\partial f}{\partial x}(x_t^{(i)}, u_t^{(i)}) & 0 \\ 0 & 1 \end{bmatrix} \\ B_t &= \begin{bmatrix} \frac{\partial f}{\partial u}(x_t^{(i)}, u_t^{(i)}) \\ 0 \end{bmatrix} \\ Q_t &= \begin{bmatrix} \frac{\partial^2 g}{\partial x^2}(x^{(i)}) & \frac{\partial g}{\partial x}(x^{(i)}) \\ \left(\frac{\partial g}{\partial x}(x^{(i)})\right)^\top & 2g(x^{(i)}) \end{bmatrix} \\ R_t &= \begin{bmatrix} \frac{\partial^2 g}{\partial u^2}(u^{(i)}) & \frac{\partial g}{\partial u}(u^{(i)}) \\ \left(\frac{\partial g}{\partial u}(u^{(i)})\right)^\top & 2g(u^{(i)}) \end{bmatrix} \end{aligned}$$

for simplicity and with some abuse of notation we assumed $g(x,u) = g(x) + g(u)$

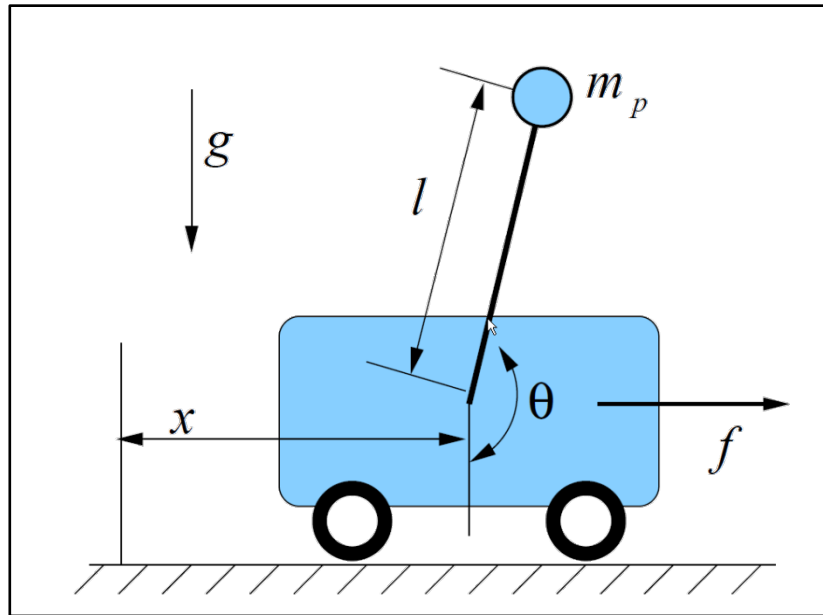
Iteratively Apply LQR: Practicalities

- f is non-linear, hence this is a non-convex optimization problem. Can get stuck in local optima! Good initialization matters.
- g could be non-convex: Then the LQ approximation can fail to have positive-definite cost matrices.
 - Practical fix: if Q_t or R_t are not positive definite \rightarrow increase penalty for deviating from current state and input $(x^{(i)}_t, u^{(i)}_t)$ until resulting Q_t and R_t are positive definite.

Can We Do Even Better?

- Yes!
- At convergence of iLQR, we end up with linearizations around the (state,input) trajectory the algorithm converged to
- In practice: the system could not be on this trajectory due to perturbations / initial state being off / dynamics model being off / ...
- Solution: at time t when asked to generate control input u_t , we could re-solve the control problem using iLQR over the time steps t through H
- Replanning entire trajectory is often impractical \rightarrow in practice: replan over horizon h .
= ***receding horizon control***
 - This requires providing a cost to go $J^{(t+h)}$ which accounts for all future costs. This could be taken from the offline iLQR run

Cart-pole

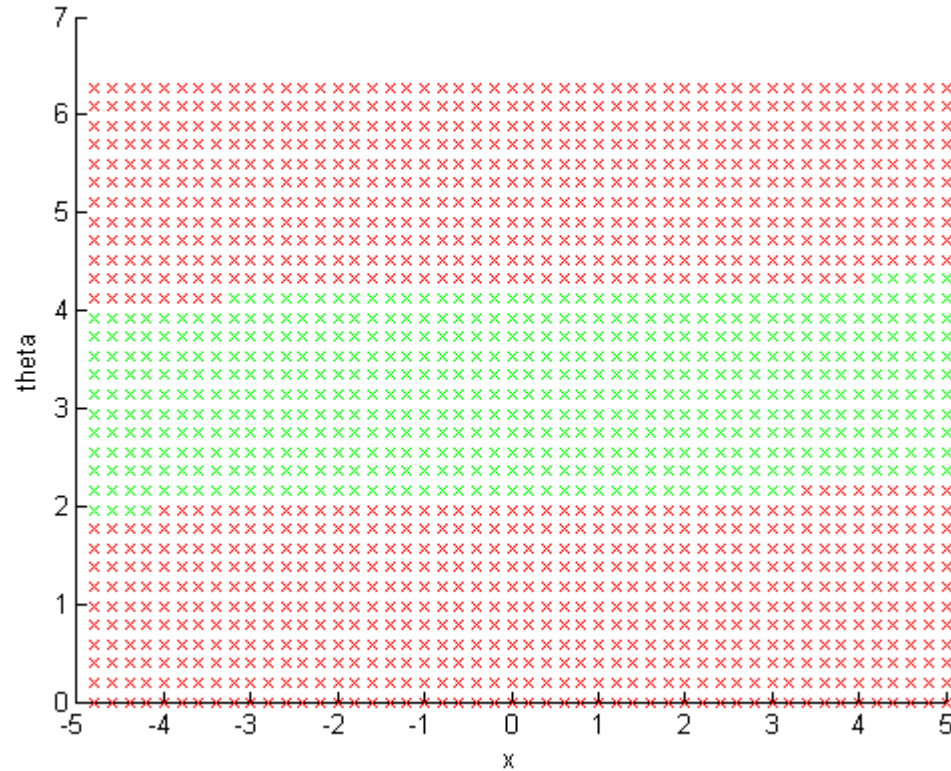


$$H(q)\ddot{q} + C(q, \dot{q}) + G(q) = B(q)u$$

$$\begin{aligned} H(q) &= \begin{bmatrix} m_c + m_p & m_p l \cos \theta \\ m_p l \cos \theta & m_p l^2 \end{bmatrix} \\ C(q, \dot{q}) &= \begin{bmatrix} 0 & -m_p l \dot{\theta} \sin \theta \\ 0 & 0 \end{bmatrix} \\ G(q) &= \begin{bmatrix} 0 \\ m_p g l \sin \theta \end{bmatrix} \\ B &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{aligned}$$

Cart-pole --- LQR

Results of running LQR for the linear time-invariant system obtained from linearizing around $[0;0;0;0]$. The cross-marks correspond to initial states. Green means the controller succeeded at stabilizing from that initial state, red means not.

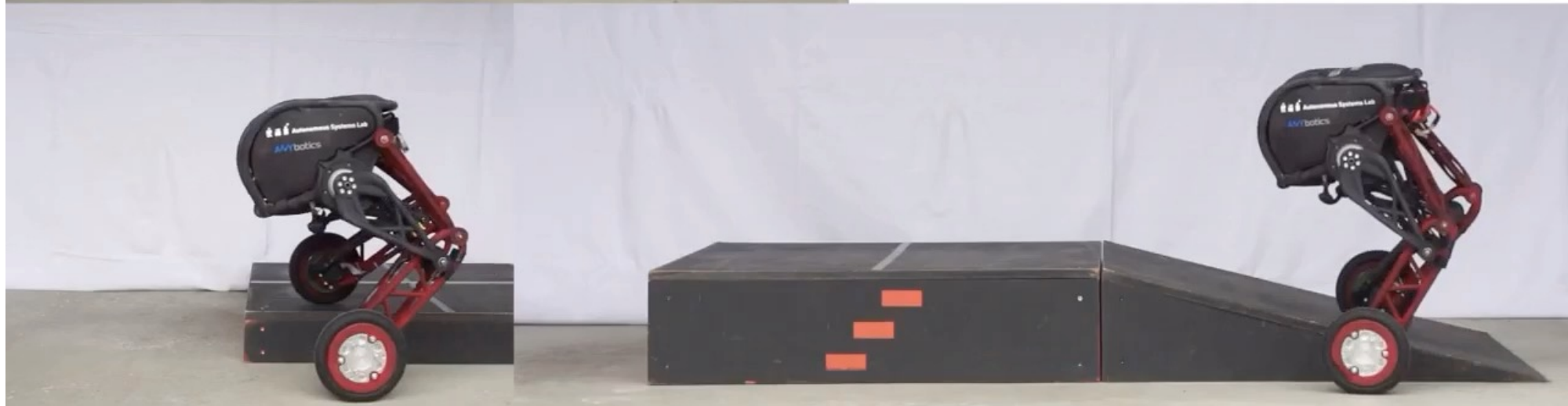


$$Q = \text{diag}([1;1;1;1]); R = 0; [x, \text{theta}, \text{xdot}, \text{thetadot}]$$

LQR in Action



Overcoming challenging indoor environments.



Learning Linear Dynamics Latent Spaces

- ***Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images***
Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, Martin Riedmiller
<https://arxiv.org/abs/1506.07365>
- ***Deep Spatial Autoencoders for Visuomotor Learning***
Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, Pieter Abbeel
<https://arxiv.org/abs/1509.06113>
- ***SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning***
Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J. Johnson, Sergey Levine
<https://arxiv.org/abs/1808.09105>

Recap: Course Overview

Filtering/Smoothing

Localization

Mapping

SLAM

Search

Motion Planning

TrajOpt

Stability/Certification

MDPs and RL

Imitation Learning

Solving POMDPs