



Introduction to Embodied AI

Lecture 4 - Robotics III

Prof. He Wang



Logistics

Logistics

- Assignment 1: released last Friday night, due 03/22 (Saturday) 11:59PM.

☐ 论坛	描述	帖子总数	未读帖子	未读对我的回复	参与者总数
☐ Assignment 1 discussion		0	0	0	0

1. **Rotation functions:** You will create a simple toolbox for handling rotations.
2. **Toy forward kinematics:** Given a humanoid, you need to compute the position and orientation of its end effector.

Motion Planning

Agenda

- Problem Formulation
- Probabilistic Roadmap Method (PRM)
- Rapidly-exploring Random Trees (RRT)

Configuration Space

- Configuration space ($\mathcal{C}$ -space) is a subset of $\mathbb{R}^n$ containing all possible states of the system (state space in RL).
- $\mathcal{C}_{free} \subseteq \mathcal{C}$ contains all valid states.
- $\mathcal{C}_{obs} \subseteq \mathcal{C}$ represents obstacles.
- Examples:
 - All valid poses of a robot.
 - All valid joint values of a robot.
 - ...

Motion Planning

- Problem:
 - Given a configuration space $\mathcal{C}_{free}$
 - Given start state q_{start} and goal state q_{goal} in $\mathcal{C}_{free}$
 - Calculate a sequence of actions that leads from start to goal
- Challenge:
 - Need to avoid obstacles
 - Long planning horizon
 - High-dimensional planning space

Motion Planning

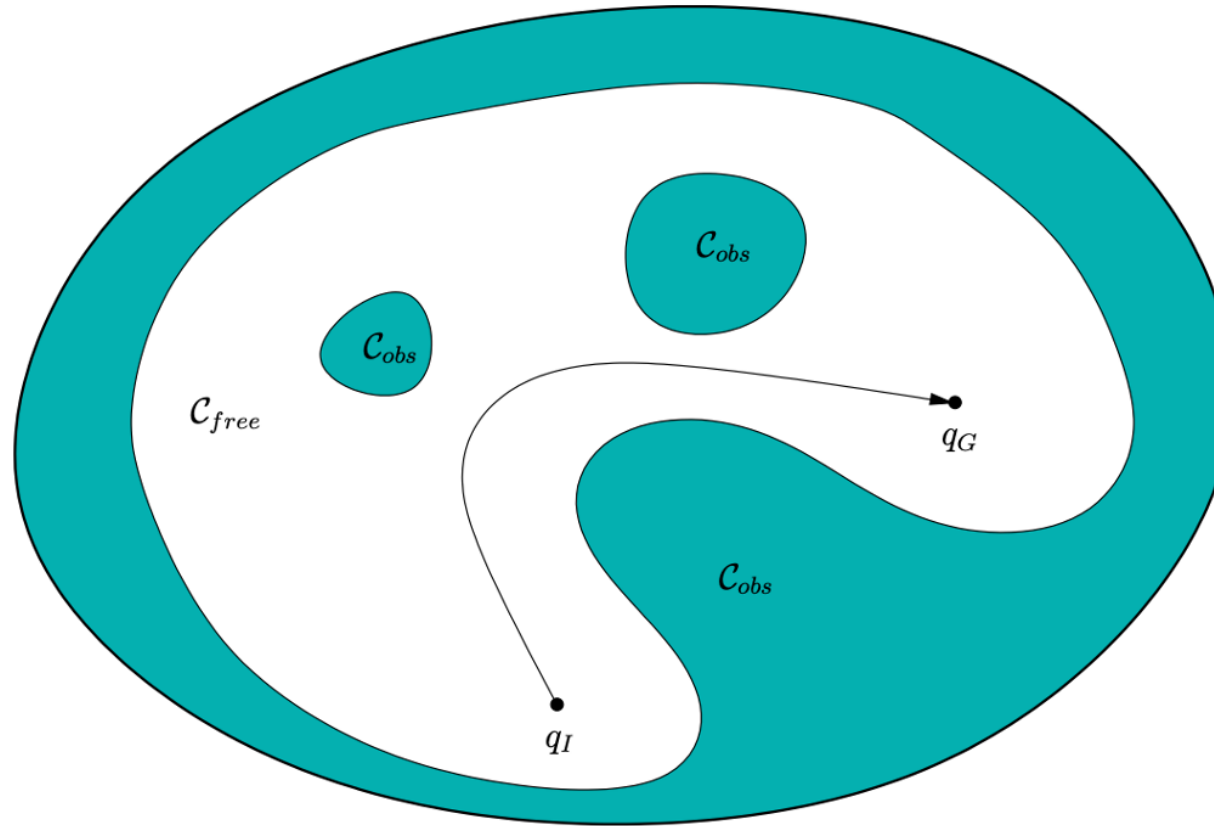
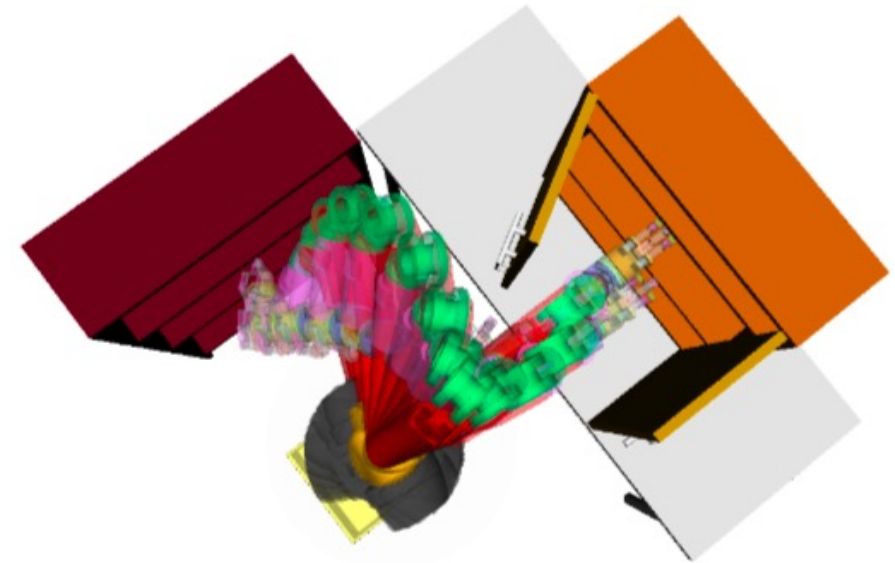


Figure 4.11: The basic motion planning problem is conceptually very simple using C-space ideas. The task is to find a path from q_I to q_G in $\mathcal{C}_{free}$. The entire blob represents $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$.

Examples

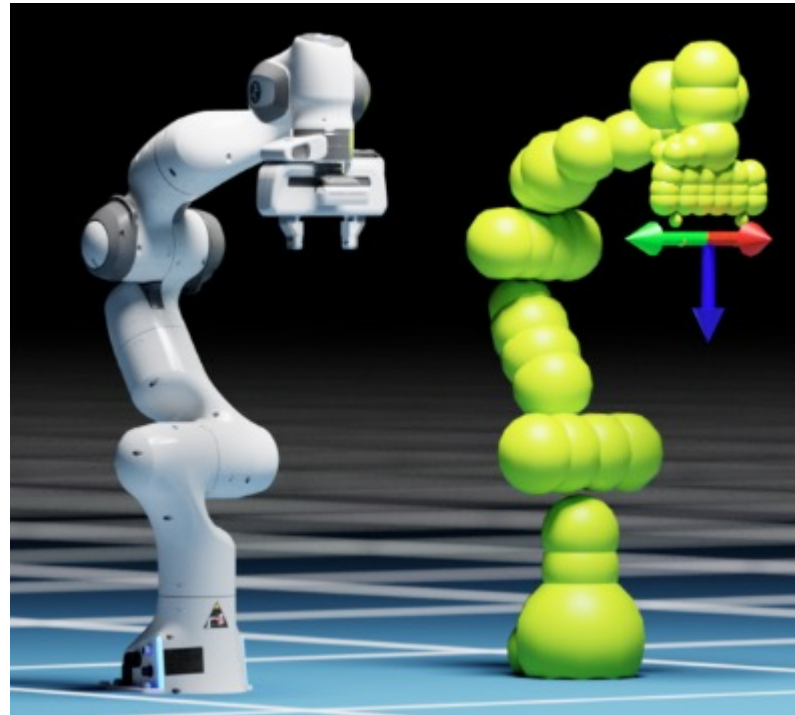


- Ratliff N, Zucker M, Bagnell J A, et al. *CHOMP: Gradient optimization techniques for efficient motion planning*, ICRA 2009
- Schulman, John, et al. *Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization*, RSS 2013

Collision modeling of robot

- Collision mesh of each link is defined in URDF
- Use forward-kinematics to get robot collision mesh from q
- Run collision check to see if q is in $\mathcal{C}_{free}$

Visual
mesh



Collision
mesh (collider)

Alternative: Convex Decomposition

- Convex decomposition:
 - Taking an arbitrarily complex concave triangle mesh and approximating it as a collection of convex objects.
 - Convex meshes collision checking are far more efficient than non-convex meshes



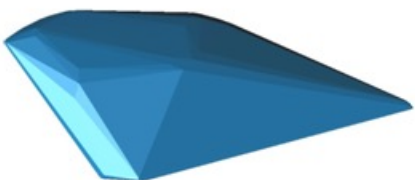
Alternative: Convex Decomposition

- Convex-Hull:
 - Get a single convex mesh, most efficient but not accurate
- Exact Convex Decomposition:
 - NP-hard, non practical since they produce a high number of clusters
- **Approximate Convex Decomposition(ACD):**
 - Determine a partition of the mesh triangles with a minimal number of clusters, while ensuring that each cluster has a concavity lower than a user defined threshold.

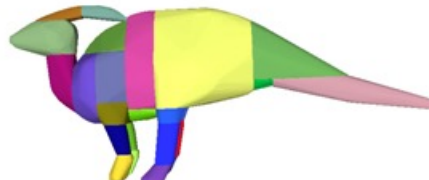
Original Mesh



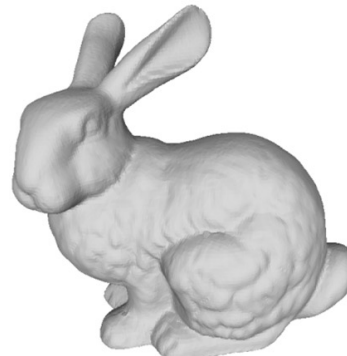
Convex-Hull



Approximate Convex
Decomposition



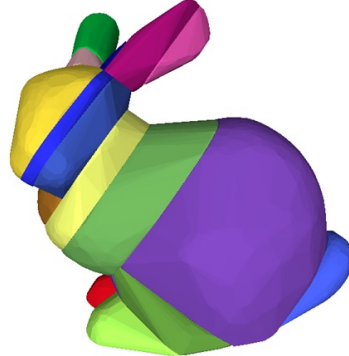
Original Mesh



Exact Convex
Decomposition
(7611 parts)



Approximate Convex
Decomposition
(20 parts)



Sample-based Algorithm

- The key idea is to explore a smaller subset of possibilities randomly without exhaustively exploring all possibilities.
- Pros:
 - Probabilistically complete
 - Solve the problem after knowing partial of $\mathcal{C}_{free}$
 - Apply easily to high-dimensional $\mathcal{C}$ -space
- Cons:
 - Requires to find path between two close points
 - Does not work well when the connection of $\mathcal{C}_{free}$ is bad
 - Never optimal

Probabilistic Roadmap Method (PRM)

Probabilistic Roadmap(PRM)

- The algorithm contains two stages:
 - Map construction phase
 - Randomly sample states in $\mathcal{C}_{free}$
 - Connect every sampled state to its neighbors
 - Connect the start and goal state to the graph
 - Query phase
 - Run path finding algorithms like Dijkstra

Rejection Sampling

- Aim to sample uniformly in $\mathcal{C}_{free}$.
- Method
 - Sample uniformly over $\mathcal{C}$.
 - Reject the sample not in the feasible area.

Pipeline

Input: n : number of sampled nodes in the roadmap, k : number of closest neighbours to examine for each configuration, q_{start}, q_{goal} .

$V \leftarrow \{q_{start}, q_{goal}\};$

$E \leftarrow \emptyset;$

while $|V| < n$ **do**

repeat

$q \leftarrow$ a random configuration in C .

until q is in C_{free} ;

end

foreach $q \in V$ **do**

$N_q \leftarrow$ the k closest neighbours of q chosen from V according to a distance function;

foreach $q' \in N_q$ **do**

if $(q, q') \notin E$ and $(q, q') \in C_{free}$ **then**

$E \leftarrow E \cup \{(q, q')\}$

end

end

end

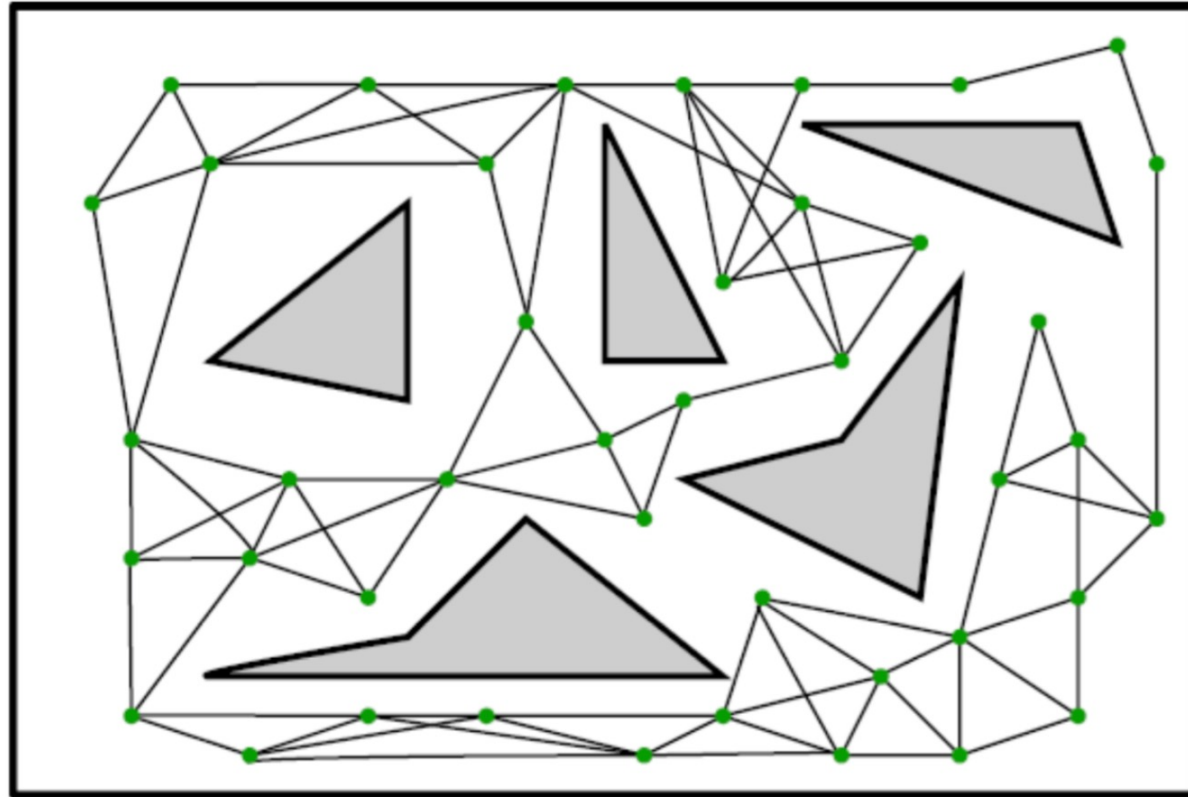
Find a path from q_{start} to q_{goal} with Dijkstra algorithm;

Challenges

- Connect neighboring points:
 - In general it requires solving dynamics
- Collision checking:
 - It takes a lot of time to check if the edges are in the configuration space.

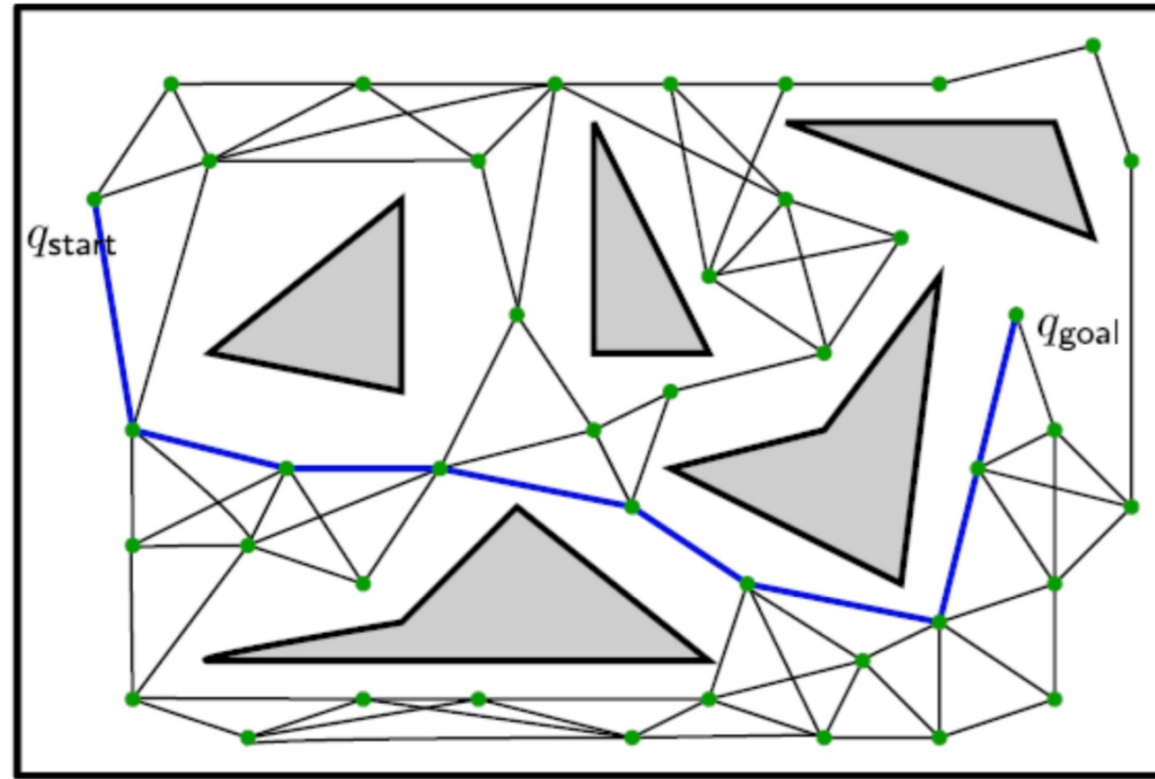
Example

PRM generates a graph $G = (V, E)$ such that every edge is in the configuration space without colliding with obstacles.



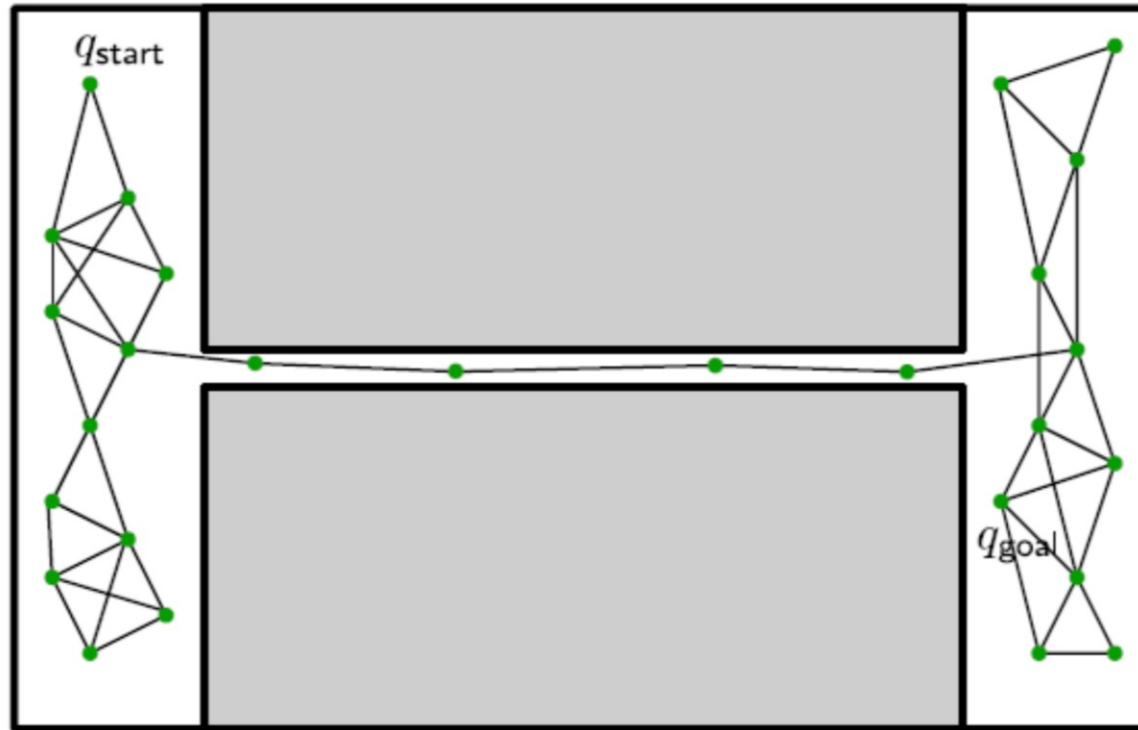
Example

Find the path from start state q_{start} to goal state q_{goal}



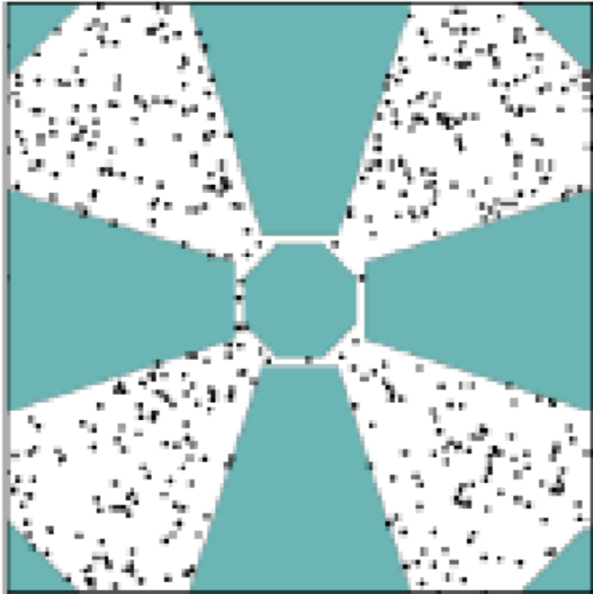
Limitations: Narrow Passages

It is unlikely to sample the points in the narrow bridge

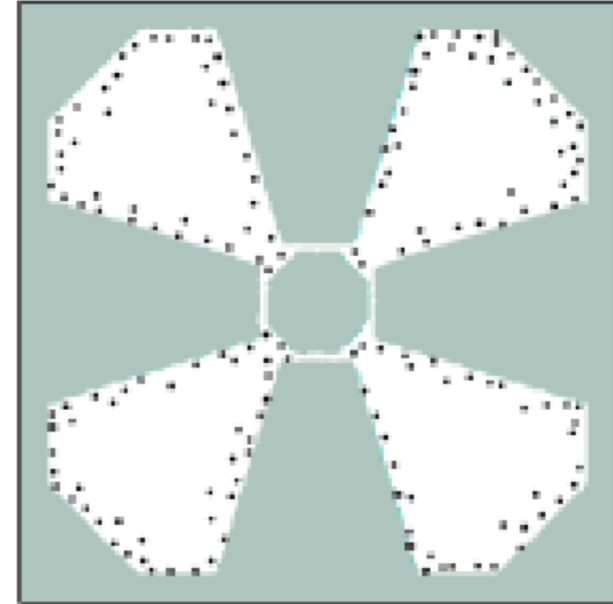


Gaussian Sampling

- Generate one sample q_1 uniformly in the configuration space
- Generate another sample q_2 from a Gaussian distribution $\mathcal{N}(q_1, \sigma^2)$
- If $q_1 \in \mathcal{C}_{free}$ and $q_2 \neq \mathcal{C}_{free}$ then add q_1



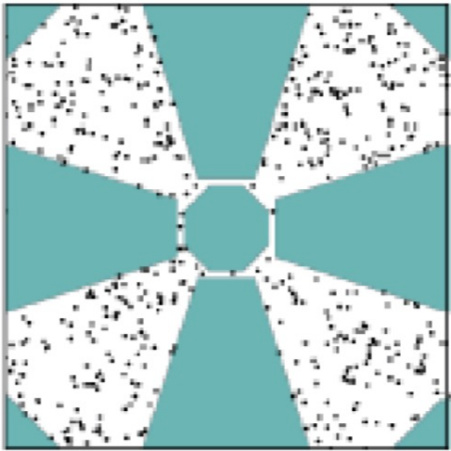
Uniform sampling



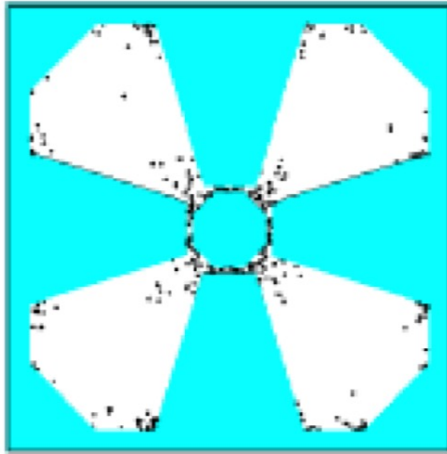
Gaussian sampling

Bridge Sampling

- Generate one sample q_1 uniformly in the configuration space
- Generate another sample q_2 from a Gaussian distribution $\mathcal{N}(q_1, \sigma^2)$
- $q_3 = \frac{q_1 + q_2}{2}$
- If q_1, q_2 are not in $\mathcal{C}_{free}$ then add q_3



Uniform sampling



Bridge sampling

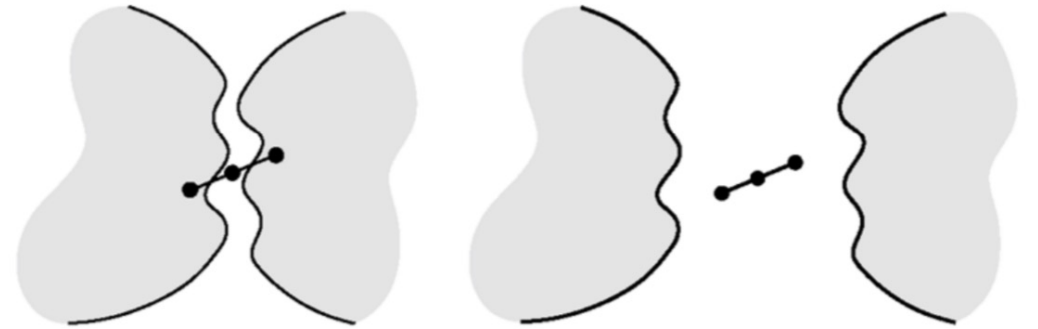


Fig. 2. Building short bridges is much easier in narrow passages (left) than in wide-open free space (right).

Rapidly-exploring Random Trees (RRT)

Rapidly-exploring Random Tree(RRT)

- RRT grows a tree rooted at the start state by using random samples from configuration space.
- As each sample is drawn, a connection is attempted between it and the nearest state in the tree. If the connection is in the configuration space, this results in a new state in the tree.

Extend Operation

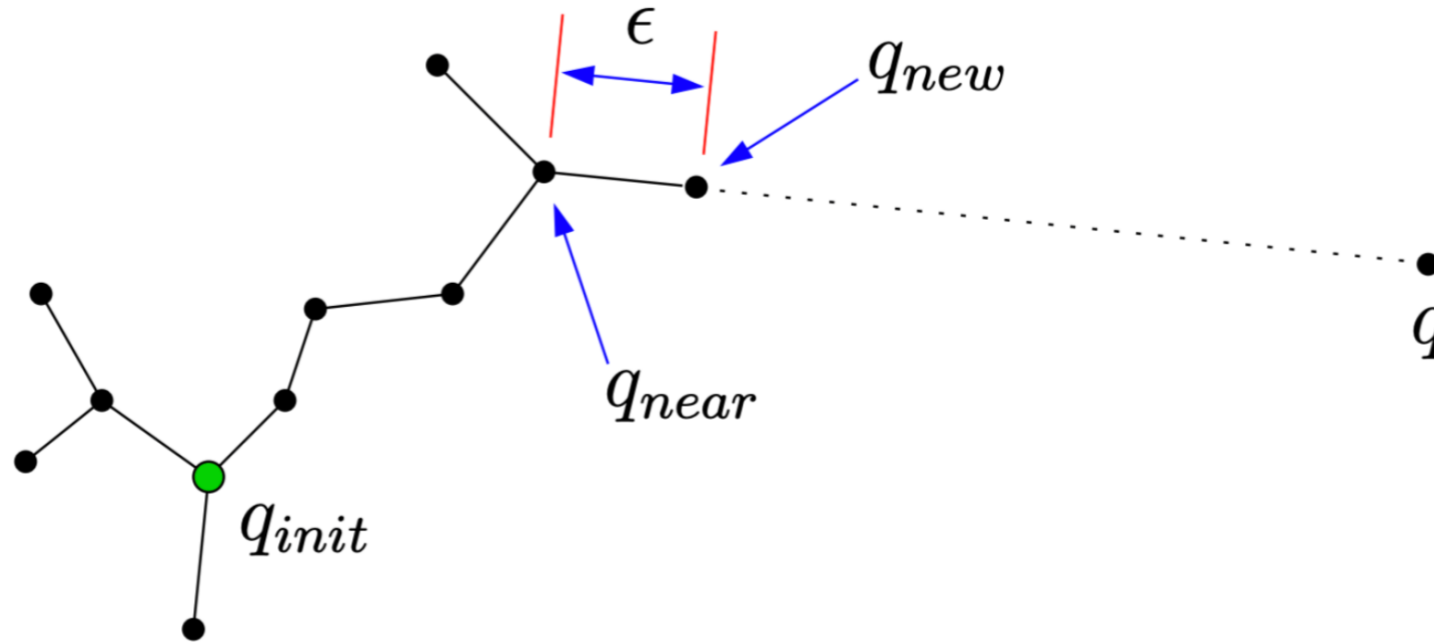


Figure 3: The EXTEND operation.

Pipeline

Input: n : number of sampled nodes in the tree, ϵ is the stepsize, β is the probability of sampling q_{goal} , q_{start} , q_{goal} .

$V \leftarrow \{q_{start}\};$

$E \leftarrow \emptyset;$

for $i = 1 \rightarrow n$ **do**

if $\text{rand}(0, 1) < \beta$ **then**

$q_{target} \rightarrow q_{goal}$

end

else

$q_{target} \rightarrow$ uniformly random sample from C_{free}

end

$q_{near} \rightarrow$ nearest neighbor of q_{target} in V ;

$q_{new} \rightarrow q_{near} + \frac{\epsilon}{|q_{near} - q_{target}|} (q_{target} - q_{near})$;

if $q_{new} \notin V$ and $q_{new} \in C_{free}$ and $(q_{near}, q_{new}) \in C_{free}$ **then**

$V \rightarrow V \cup \{q_{new}\};$

$E \rightarrow E \cup \{(q_{near}, q_{new})\}$;

end

end

Find a path from q_{start} to q_{goal} with Dijkstra algorithm;

Challenges

- Find nearest neighbor in the tree
 - We need to support online quick query
 - Examples: KD Trees
- Need to choose a good ϵ to expand the tree efficiently
 - Large ϵ : hard to generate new samples
 - Small ϵ : too many samples in the tree

RRT-Connect

- Grow two trees starting from q_{start} and q_{start} respectively instead of just one.
- Grow the trees towards each other rather than random configurations
- Use stronger greediness by growing the tree with multiple epsilon steps instead of a single one.

Kuffner, James J., and Steven M. LaValle. "RRT-connect: An efficient approach to single-query path planning." Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065). Vol. 2. IEEE, 2000.

Pseudo Code

```
RANDOM_CONFIG( $q_{goal}$ )
if  $rand(0, 1) < \beta$  then
    return  $q_{goal}$ 
else
    return random sample from  $C_{free}$ 
end if
```

```
EXTEND( $\mathcal{T}, q$ )
 $q_{near} \leftarrow \text{NEAREST\_NEIGHBOR}(\mathcal{T}, q)$ 
if  $|q_{near} - q| < \epsilon$  then
     $q_{new} \leftarrow q$ 
else
     $q_{new} \leftarrow q_{near} + \frac{\epsilon}{|q - q_{near}|}(q - q_{near})$ 
end if
if  $q_{new} \in C_{free}$  and  $(q_{near}, q_{new}) \in C_{free}$  then
     $\mathcal{T}.\text{add\_vertex}(q_{new})$ 
     $\mathcal{T}.\text{add\_edge}((q_{near}, q_{new}))$ 
    if  $q_{new} = q$  then
        return Reached
    else
        return Advanced
    end if
else
    return Trapped
end if
```

```
CONNECT( $\mathcal{T}, q$ )
repeat
     $S \leftarrow \text{EXTEND}(\mathcal{T}, q)$ 
until  $S \neq \text{Advanced}$ 
return  $S$ 
```

```
RRT_CONNECT( $q_{start}, q_{goal}$ )
 $\mathcal{T}_a.\text{init}(q_{start}), \mathcal{T}_b.\text{init}(q_{goal})$ 
for  $k = 1$  to  $K$  do
     $q_{rand} \leftarrow \text{RANDOM\_CONFIG}(q_{goal})$ 
    if  $\text{EXTEND}(\mathcal{T}_a, q_{rand}) \neq \text{Trapped}$  then
        if  $\text{CONNECT}(\mathcal{T}_b, q_{new}) = \text{Reached}$  then
            return  $\text{PATH}(\mathcal{T}_a, \mathcal{T}_b)$ 
        end if
    end if
     $\text{SWAP}(q_{start}, q_{goal}), \text{SWAP}(\mathcal{T}_a, \mathcal{T}_b)$ 
end for
return Failure
```

Shortcutting

- Algorithms like RRT and RRT connect is **not asymptotically optimal**.
- PRM is asymptotically optimal but requires **massive sampling**.
- PRM and RRT often produce jerky, unnatural paths
- Shortcutting is a very useful post-processing heuristic that solves these problems.

Shortcutting

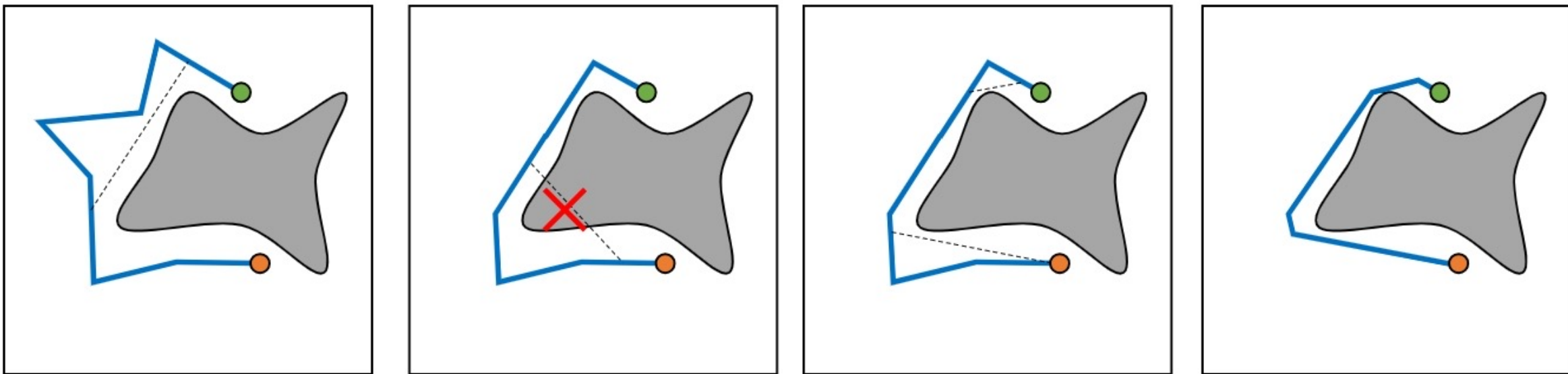


Figure 11. A shortcutting heuristic can quickly smooth out the jerkiest parts of paths that are generated by a sampling-based planner.

In order to do so, two random points are sampled along the path. With the path being a curve $y(s) : [0, 1] \rightarrow \mathcal{C}$ parameterized over the unit interval, we sample two parameters $u, v \in [0, 1]$. If $Visible(y(u), y(v))$ is true, then we replace the portion of the path between u and v with the straight line path. Otherwise, the sampling process begins again. This repeats for some number of iterations.

Shortcutting

- Shortcutting is only a local optimization technique, but it is very fast. In fact, when the time limit is short, this can give much better paths than PRM and RRT!
- However, shortcutting quickly stops making progress, and the asymptotically-optimal techniques end up winning out.
- We can overcome the "stalling out" problem by combining shortcutting with random-restarts.

Shortcutting

Algorithm Shortcutting(y, N)

1. **for** n iterations **do**
2. Let $a \leftarrow y(u)$ with $u \sim \text{Uniform}([0, 1])$.
3. Let $b \leftarrow y(v)$ with $v \sim \text{Uniform}([0, 1])$.
4. If $u > v$, swap u with v and a with b .
5. **if** $\text{Visible}(a, b)$ **then**
6. Replace the portion of y between u and v with $\overline{ab}$.
7. Reparameterize y to have unit domain.
8. **return** y .

Algorithm Shortcutting with Random Restarts($T_{plan}, N_{shortcut}, N_{refine}, T_{total}$)

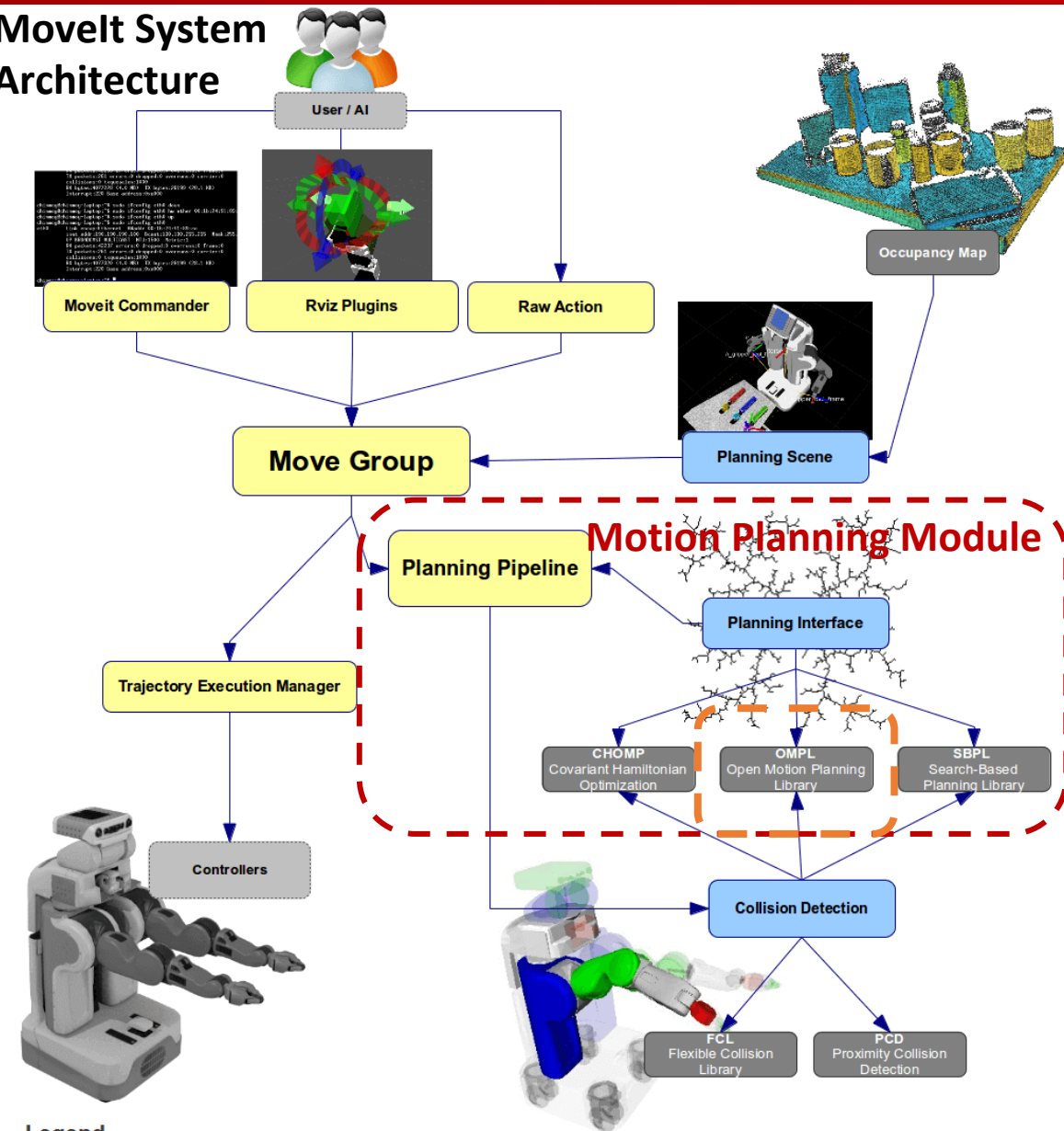
1. $P \leftarrow \text{RRT-Init}()$
2. Let $y_{best} \leftarrow \text{RRT-Plan}(P, T)$.
3. If no path is found, return "failure"
4. $P \leftarrow \text{RRT-Init}()$ (reset planner)
5. **while** the time spent planning is less than T_{total} **do**
6. $y_{best} \leftarrow \text{Shortcut}(y_{best}, N_{refine})$ (continue to improve best path)
7. $y \leftarrow \text{RRT-Plan}(P, T_{plan})$
8. **if** path y is found **then**
9. $P \leftarrow \text{RRT-Init}()$ (reset planner)
10. $y \leftarrow \text{Shortcut}(y, N_{shortcut})$
11. **if** $\text{Cost}(y) < \text{Cost}(y_{best})$ **then** $y_{best} \leftarrow y$ (better path found)
12. **return** y_{best} .

Example: OMPL

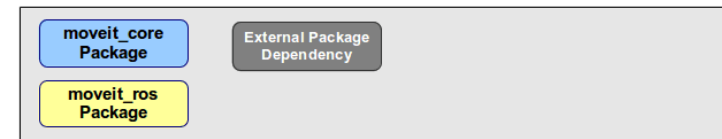
The Open Motion Planning Library

- Default planning library in ROS-MoveIt
- A powerful collection of state-of-the-art **sampling-based** motion planning algorithms

MoveIt System Architecture



Legend



Example: OMPL

The Open Motion Planning Library (OMPL)

- **Geometric planners**

- Only accounts for the geometric and kinematic constraints of the system. It is assumed that any feasible path can be turned into a dynamically feasible trajectory.
- RRT (**RRTConnect (default)**, RRT*,...), PRM (LazyPRM, PRM*, LazyPRM*,...)

- **Control-based planners**

- If the system under consideration is subject to differential constraints, then a control-based planner is used. These planners rely on state propagation rather than simple interpolation to generate motions.
- RRT, Sparse Stable RRT, ...

Example: OMPL

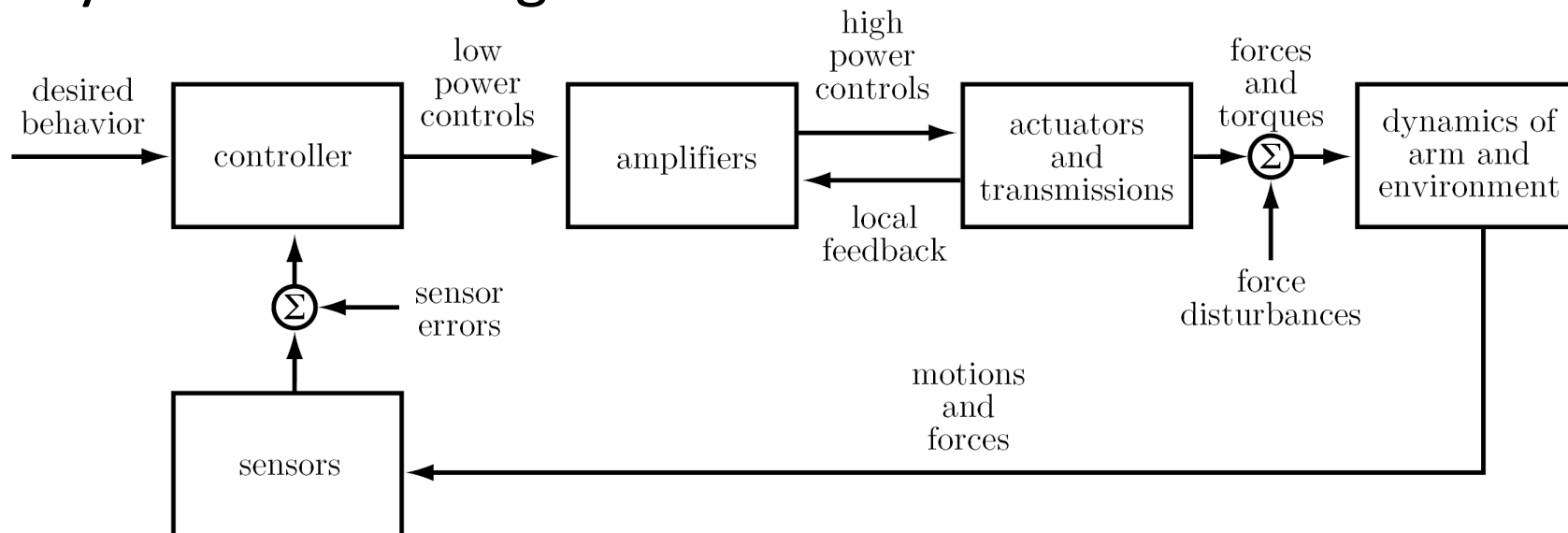
The Open Motion Planning Library (OMPL)

- **RRTConnect demo**
 - OMPL: RRTConnect demo: <https://www.youtube.com/watch?v=5IFyw0xo1qg>
 - RRTConnect planner: <https://www.youtube.com/watch?v=buFMSEhD65I>
 - RRT Connect animation: <https://www.youtube.com/watch?v=cxyRDIBAIR0>
 - MATLAB tutorial: <https://www.youtube.com/watch?v=5IFyw0xo1qg>

Control System

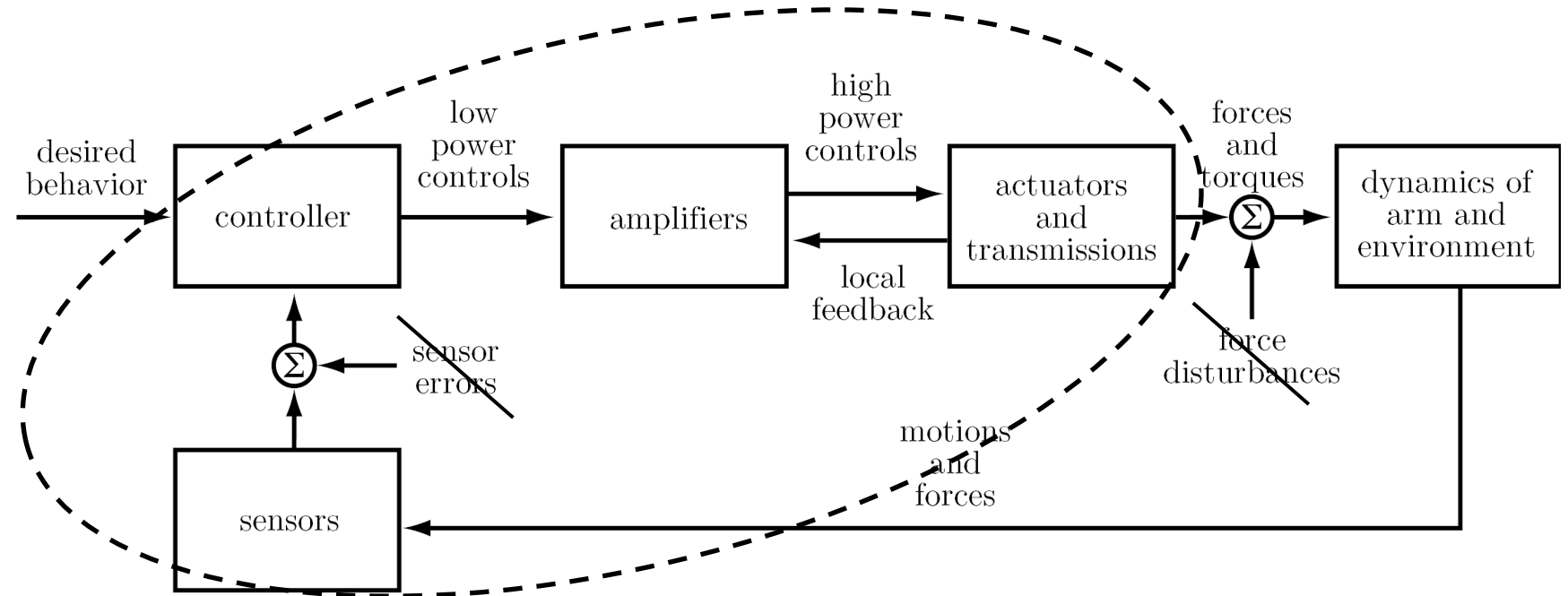
Control System

- Main components:
 - Sensor (e.g. joint angles, velocities, torques sensor)
 - Actuator (e.g. joint motor)
 - **Controller**
- Control system block diagram:

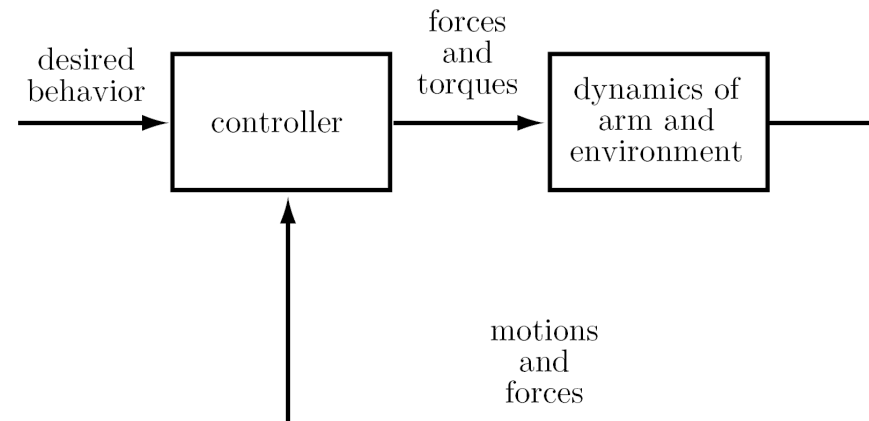


The Controller

- Block diagram:



- Simplified block diagram:

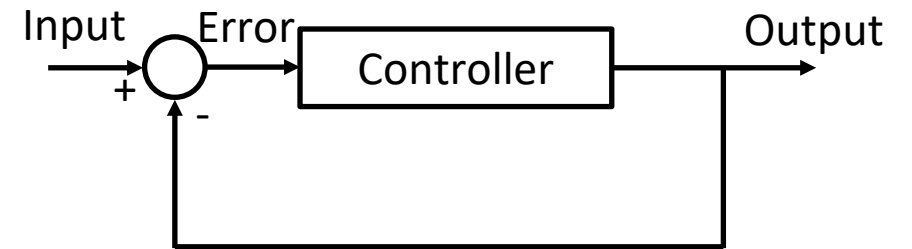


The Control Systems

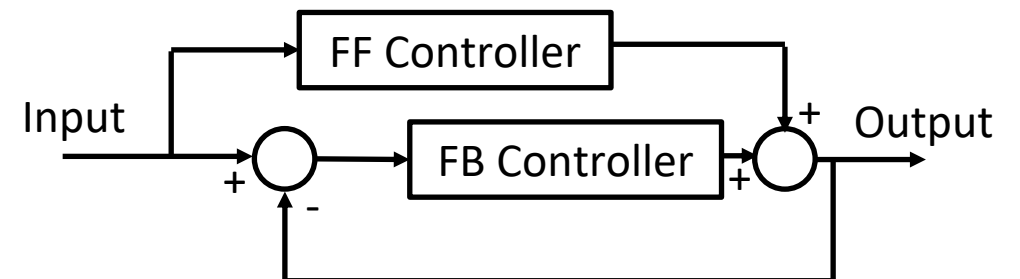
- Open loop (feedforward, FF) control:
 - Does not have state estimators
 - Does not have feedback: issues with noise and disturbance
 - Use heuristics to achieve desired state



- **Close-loop (feedback, FB) control:**
 - Achieve desired state
 - Counteract disturbance
 - Stabilize unstable systems



- FF & FB control:

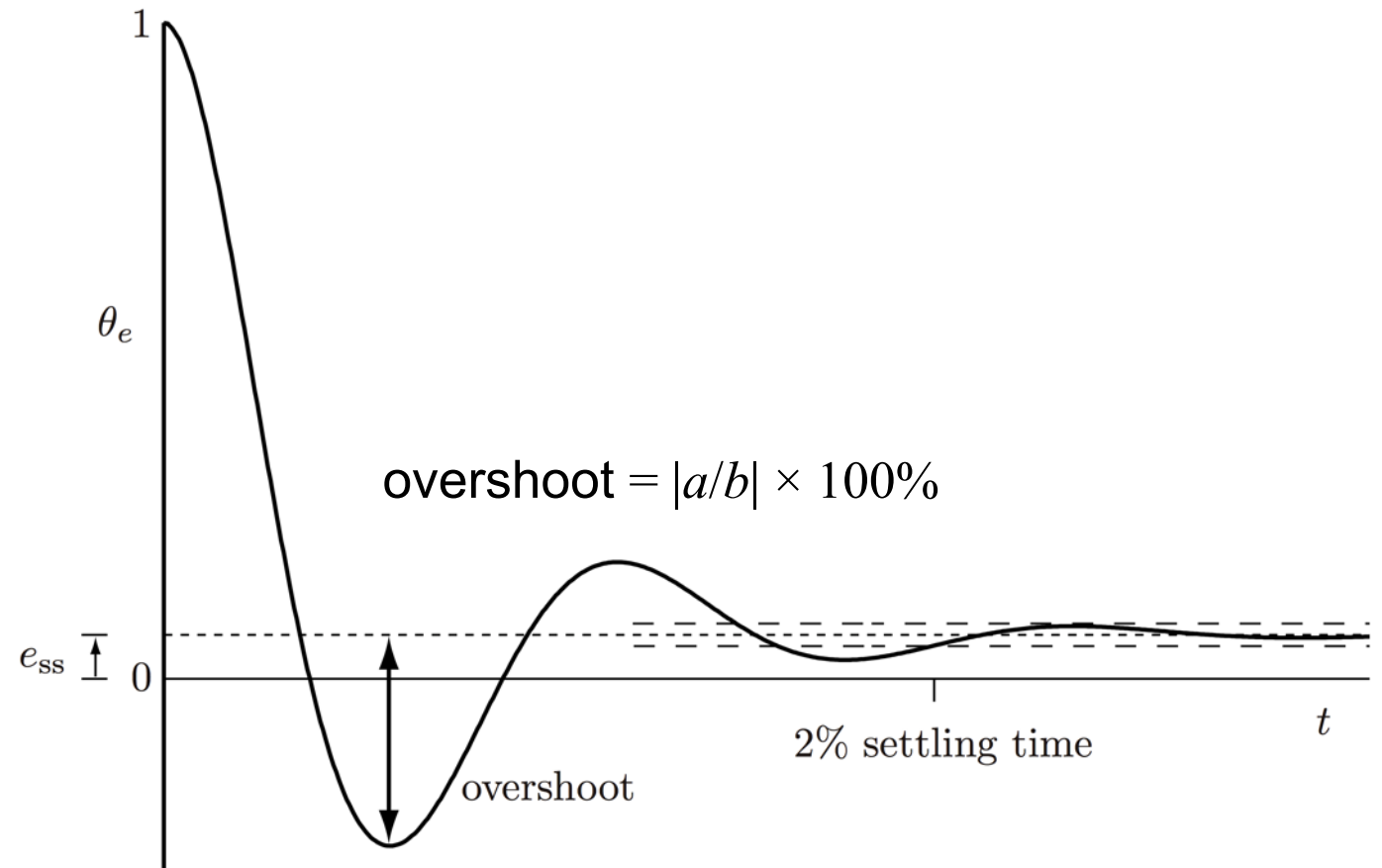


Effectiveness of a Control System

- Get to the desired state as quickly as possible and stay in that state
- We want to system to:
 - Reach Steady-State –minimize the steady state error
 - Minimize transient response –reach steady state fast
 - Be stable in steady state –minimize oscillations around the steady state

Error Response

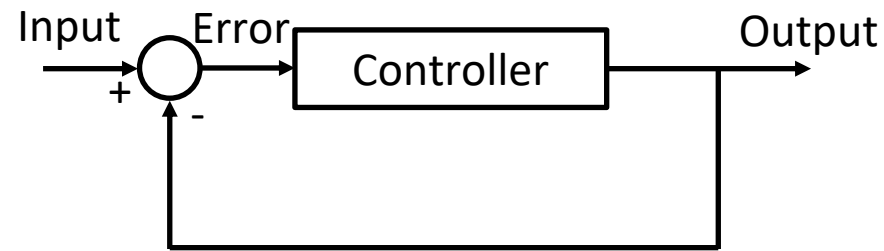
- Error function:
 - Desired state: θ_d
 - Current state: θ
 - Error: $\theta_e = \theta_d - \theta$
- Steady-State error response:
 - $e_{ss} = \lim_{t \rightarrow \infty} \theta_e(t)$
- Transient error response:
 - Overshoot
 - Settling time



PID Controller

Controller

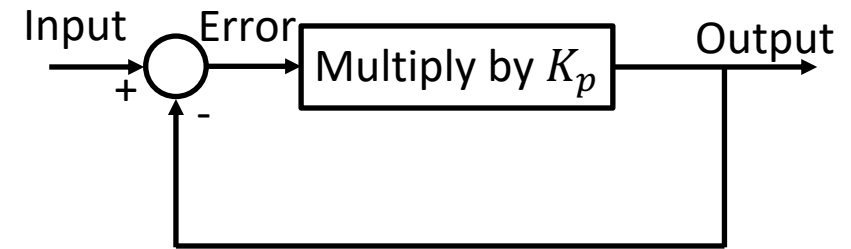
- How to convert the error signal into a command?



- Output: signal to the motor (velocity command, force/torque command, etc.)

Proportional Control

- Make the control signal proportional to the magnitude of the error in the current state
- Proportional (P) Control:
 - $P = K_p \theta_e(t)$



Proportional Control

- If control signal changes derivative of state (velocity signal):

$$\dot{\theta}(t) = K_p \theta_e(t)$$

- We have:

$$\dot{\theta}_d(t) - \dot{\theta}_e(t) = K_p \theta_e(t)$$

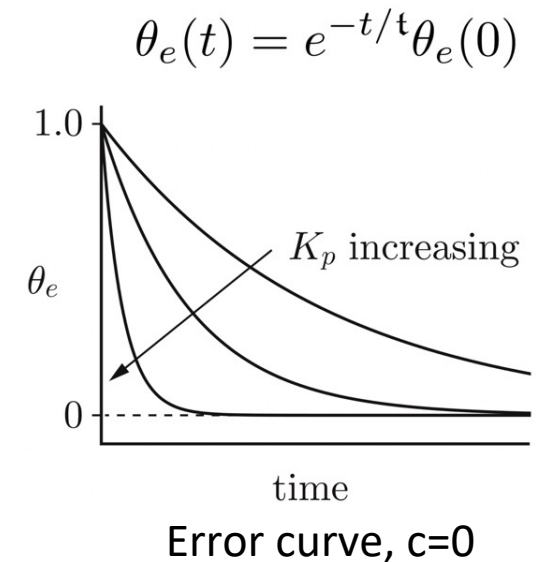
- If desired state moves with constant velocity:

$$\dot{\theta}_d(t) = c$$

- Solve the ODE to get error response:

$$\theta_e(t) = \frac{c}{K_p} + \left(\theta_e(0) - \frac{c}{K_p} \right) e^{-K_p t}$$

- State converges to target when $c=0$, otherwise there is steady-state error.



Proportional Control

- If control signal changes the second derivative of state (force or torque signal):

$$\ddot{\theta}(t) = K_p \theta_e(t)$$

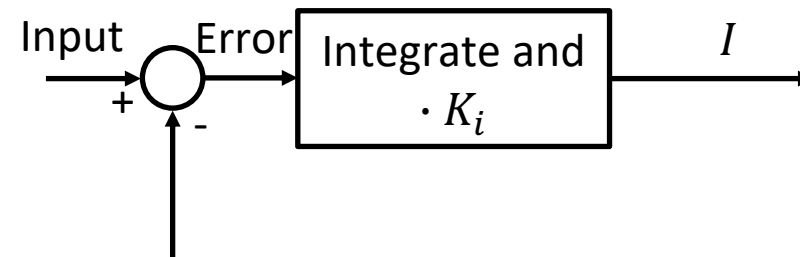
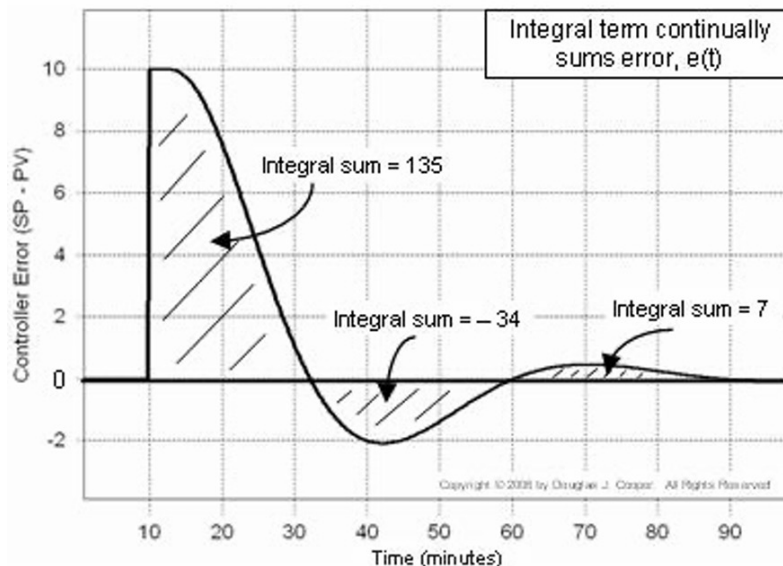
- The state oscillates and is not stable
- Can we do better?
 - Combine proportional control with other control strategies.

Integral Signal

- A factor in the output proportional to the duration and magnitude of the Error Signal.

$$I = K_i \int_0^t \theta_e(\tau) d\tau$$

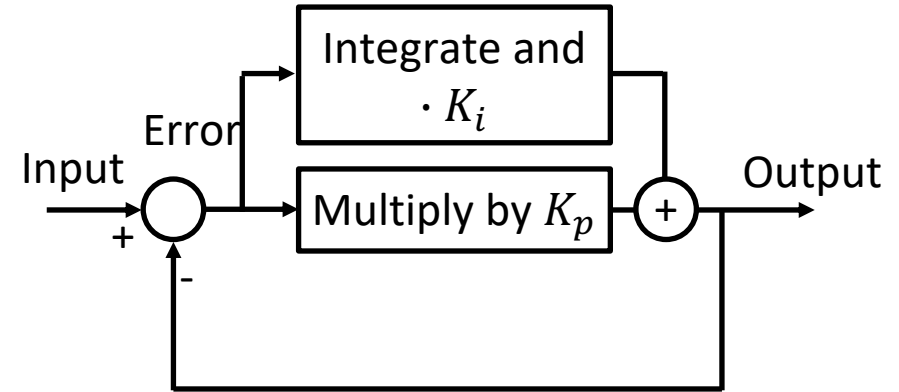
- The more prolonged the error and the greater the amount, the larger the integral output.



Proportional-Integral Control

- Combine P and I (PI Control):

$$PI = K_p \theta_e(t) + K_i \int_0^t \theta_e(\tau) d\tau$$



- If control signal changes derivative of state (velocity signal):

$$\dot{\theta}_d(t) - \dot{\theta}_e(t) = K_p \theta_e(t) + K_i \int_0^t \theta_e(\tau) d\tau$$

- We have:

$$\ddot{\theta}_e(t) + K_p \dot{\theta}_e(t) + K_i \theta_e(t) = \ddot{\theta}_d(t)$$

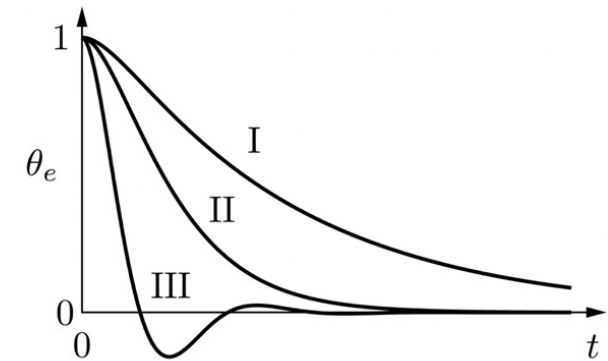
Proportional-Integral Control

$$\ddot{\theta}_e(t) + K_p \dot{\theta}_e(t) + K_i \theta_e(t) = \ddot{\theta}_d(t)$$

- If $\ddot{\theta}_d(t) = 0$, we have this ODE*:

$$\ddot{\theta}_e + K_p \dot{\theta}_e + K_i \theta_e = 0$$

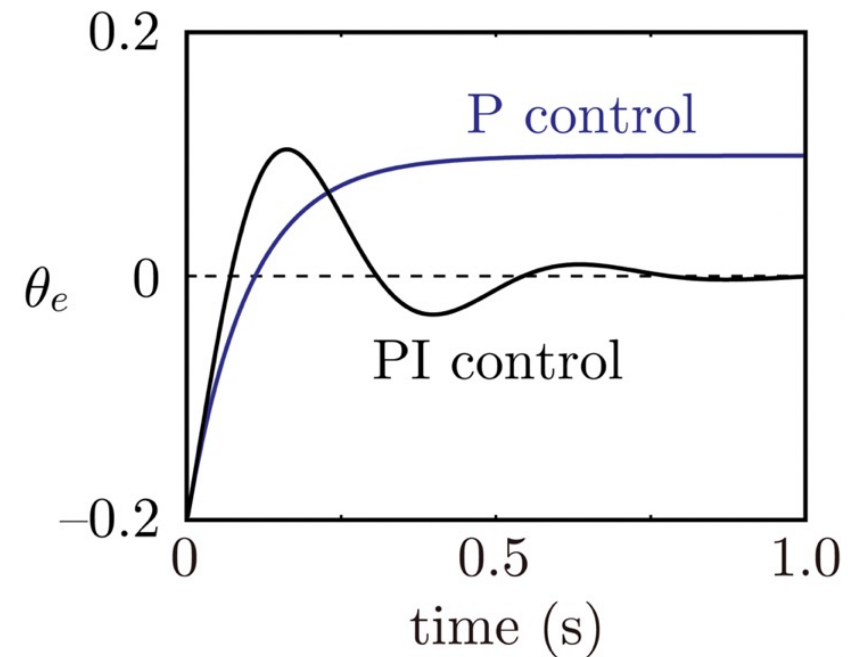
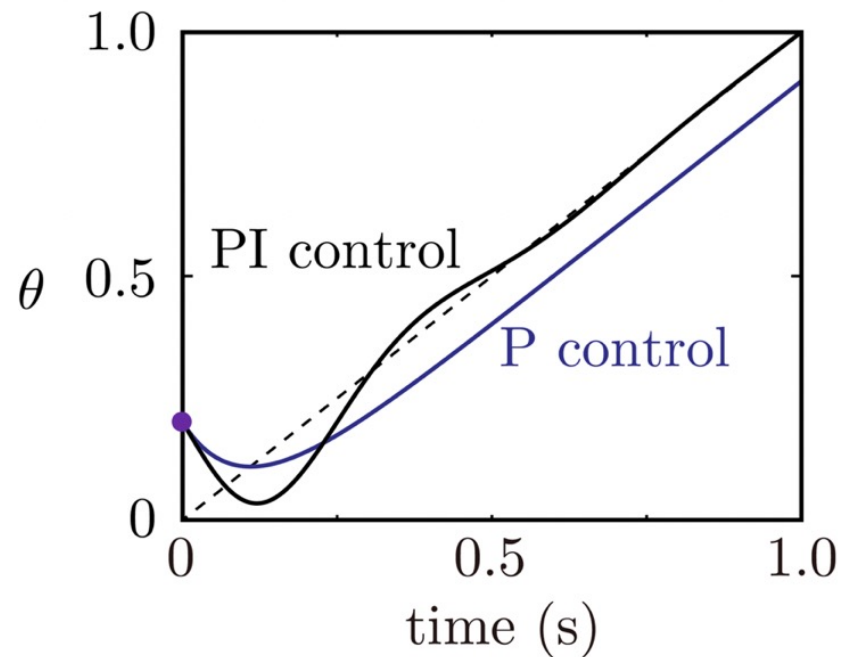
- The solutions have three different forms:
 - I. Over damped
 - II. Critically damped
 - III. Under damped



*:Second-order linear homogeneous differential equation with constant coefficients

Comparing P and PI

- P control eliminates steady-state error only for setpoint control
- PI control can eliminate steady-state error for constant velocity control, but not for arbitrary trajectories



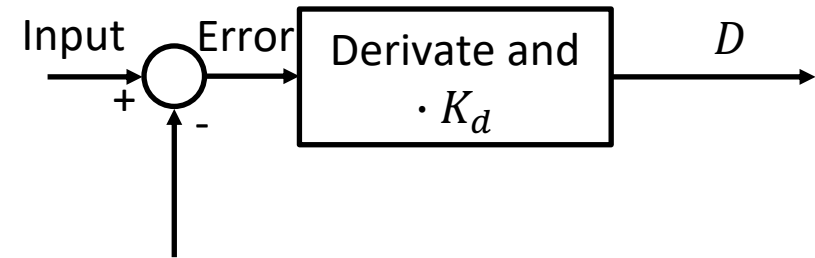
In real cases we mostly use setpoint (target velocity=0)

Derivative Signal

- Proportional to the rate of change of the error

$$D = K_d \frac{d}{dt} \theta_e(t)$$

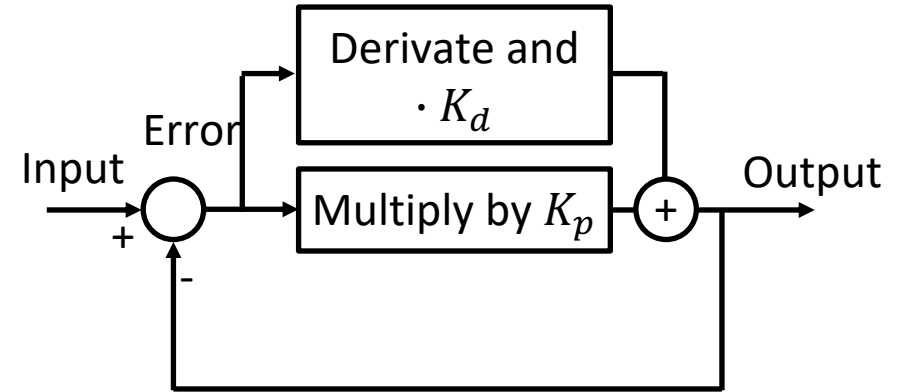
- Increase response time
 - Greater control when the error is increasing
 - Gentler control when the error is decreasing
- Accounts for “future behavior” or “trend”
- Attempts to reduce overshoot



Proportional-Derivative Control

- Combine P and D (PD Control):

$$PD = K_p \theta_e(t) + K_d \frac{d}{dt} \theta_e(t)$$



- If control signal changes the second derivative of state:

$$\ddot{\theta}_d(t) - \ddot{\theta}_e(t) = K_p \theta_e(t) + K_d \frac{d}{dt} \theta_e(t)$$

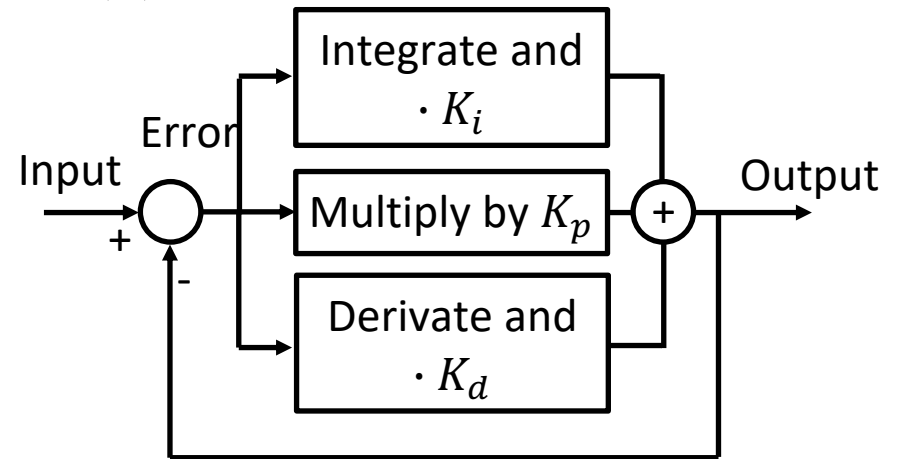
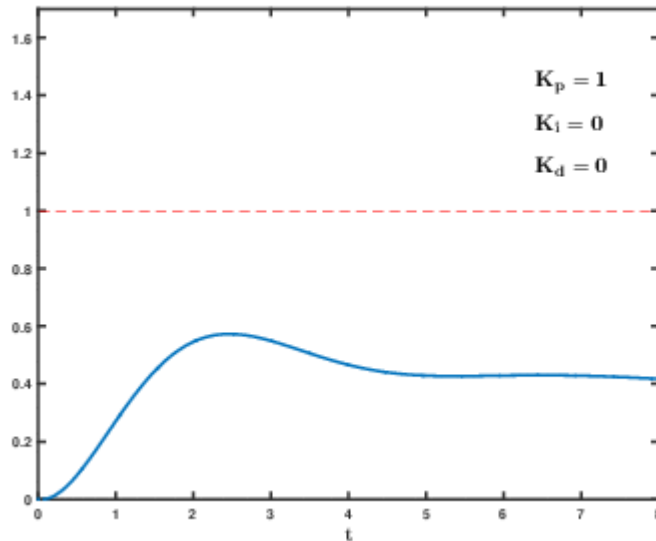
- We have (similar to velocity control of PI):

$$\ddot{\theta}_e(t) + K_d \dot{\theta}_e(t) + K_p \theta_e(t) = \ddot{\theta}_d(t)$$

PID Control

- Putting all together:

$$PID = K_p \theta_e(t) + K_i \int_0^t \theta_e(\tau) d\tau + K_d \frac{d}{dt} \theta_e(t)$$



- PID control are often used in controlling position or velocity, and not often used to directly control force

Tuning PID

Effects of *increasing* a parameter independently

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

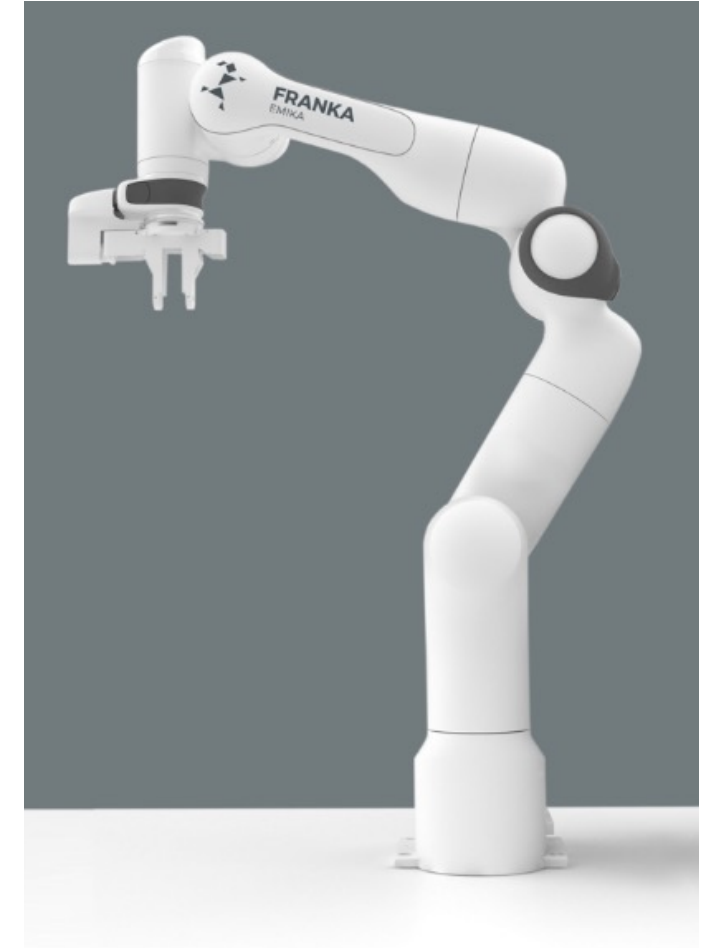
Kiam Heong Ang; Chong, G.; Yun Li (2005). "PID control system analysis, design, and technology". *IEEE Transactions on Control Systems Technology*. 13 (4): 559–576.
Jinghua Zhong (Spring 2006). "PID Controller Tuning: A Short Tutorial" (PDF). Archived from the original on 2015-04-21. Retrieved 2011-04-04.
https://en.wikipedia.org/wiki/PID_controller#cite_note-24

Controller in Robot Arms: Internal PD controller

- DoF control:
 - Control each degree of freedom (joint) using PD controller
 - Joint parameters: stiffness, damping, ...
 - Control force:

$$\begin{aligned} & \text{force} \\ &= \text{stiffness} * (\text{targetPosition} - \text{Position}) + \text{damping} * (\text{targetVelocity} - \text{Velocity}) \end{aligned}$$

- The *stiffness* and *damping* can be regarded as k_p and k_d in PID controller



Examples in SAPIEN

Drive the robot with the PhysX internal PD controller

```
active_joints = robot.get_active_joints()
if use_internal_drive:
    for joint_idx, joint in enumerate(active_joints):
        joint.set_drive_property(stiffness=20, damping=5)
        joint.set_drive_target(target_qpos[joint_idx])
    # Or you can directly set joint targets for an articulation
    # robot.set_drive_target(target_qpos)
```

SAPIEN provides builtin PhysX **drives** (controllers) to control either the position or speed of a joint. For each active joint (with non-zero degree of freedom), we can call `set_drive_property(...)` to set its drive properties: `stiffness` and `damping`. The drive is a **proportional derivative drive**, which applies a force as follows:

$$\text{force} = \text{stiffness} * (\text{targetPosition} - \text{position}) + \text{damping} * (\text{targetVelocity} - \text{velocity})$$

<https://sapien.ucsd.edu/docs>

Examples in Isaac Gym

`DOF_MODE_POS` and `DOF_MODE_VEL` engage a PD controller that can be tuned using stiffness and damping parameters. The controller will apply DOF forces that are proportional to `posError * stiffness + velError * damping`.

`DOF_MODE_POS` is used for position control, typically with both stiffness and damping set to non-zero values:



Summary of Robotics

- Kinematics: conversion between Cartesian space and joint space
- Dynamics: skipped
- Motion Planning: get a geometrically valid path
- Control: ensures the motion follow the planned trajectory



Robot Vision and Learning

— From a Perspective of Embodied AI

Next week: Lecture 4,
3D Vision

