Humanoid Robotics

Manipulation 2:
Motion Planning for Manipulation

Maren Bennewitz

Humanoid Motion Planning

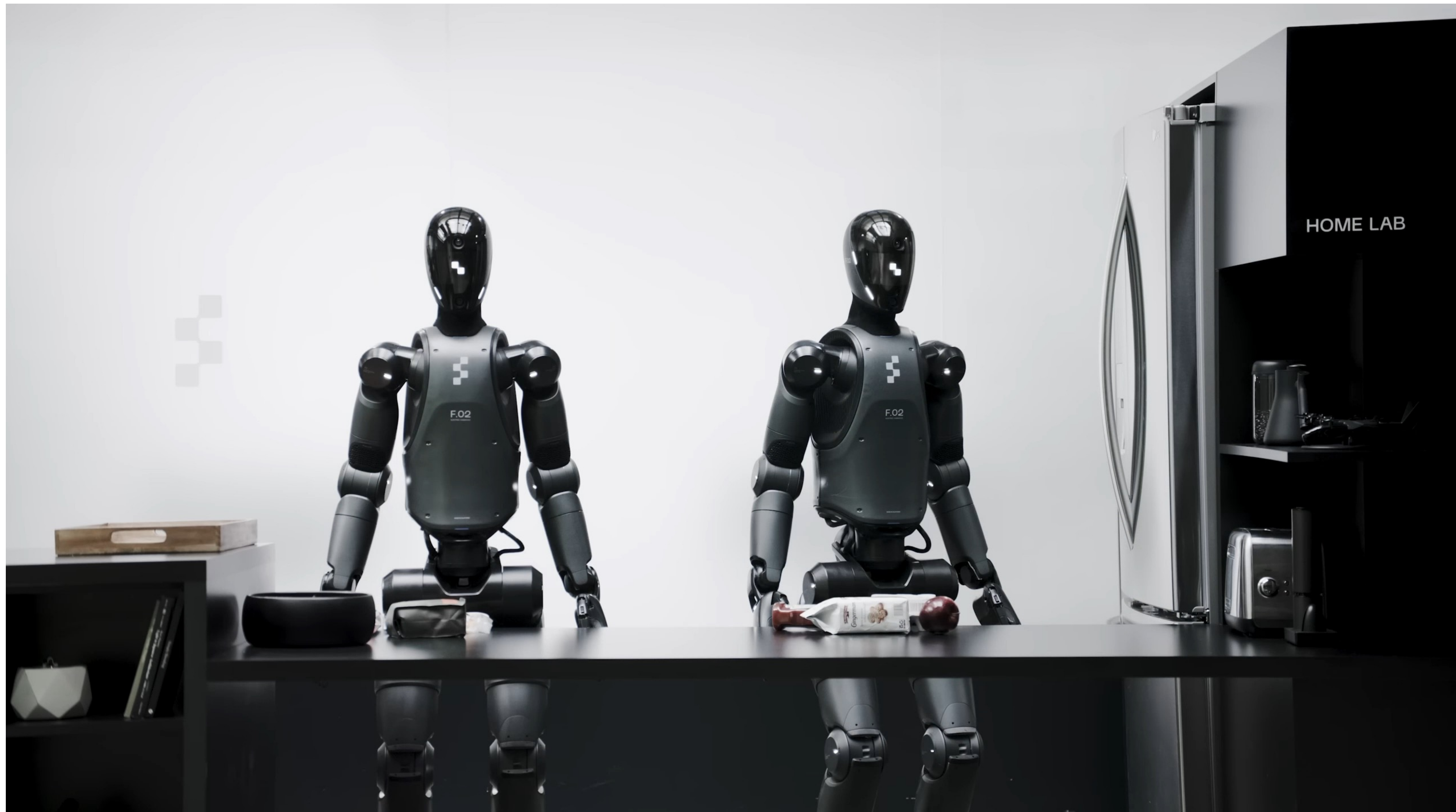
Different approaches based on the task

- Motion planning and trajectory generation for **manipulation** (upper limbs)
- Navigation and gait planning for **locomotion** (lower limbs)
- Whole-body controller ensures balancing during both tasks

Goal of This Chapter

- Introduction to basic concepts: path, trajectory, configuration space, task space
- Understanding of important components: collision detection, sampling-based planning, trajectory generation

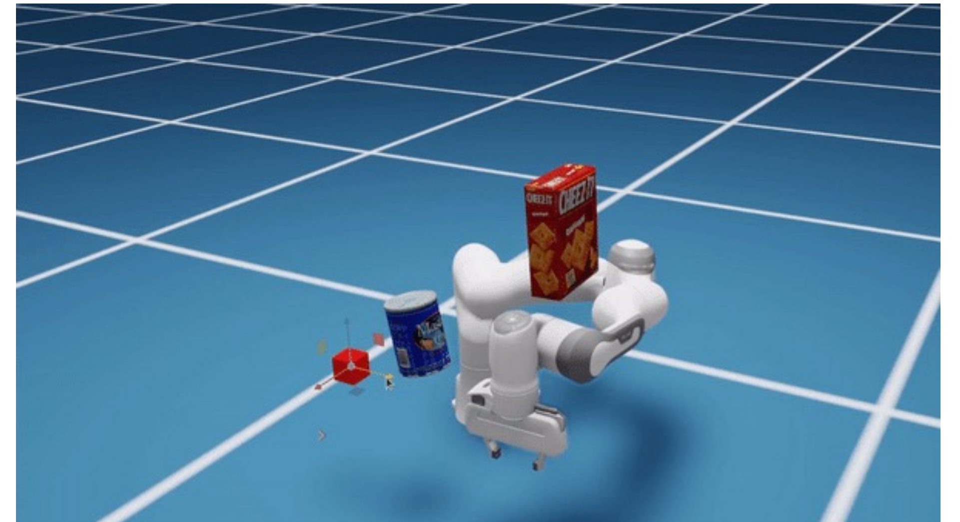
Motivation



Figure, “Introducing Helix”, 2025, www.youtube.com/watch?v=Z3yQHYNXPws

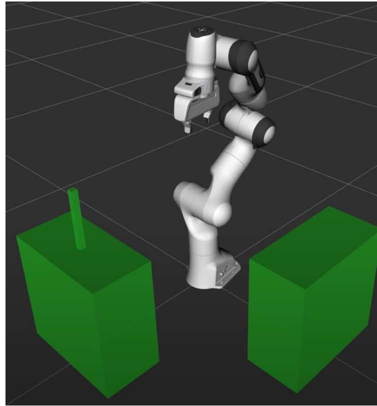
Problems to Solve

- How to **reach a target object** so that the arm can manipulate it?
- How to reach such that the arm motion is **collision-free** in a cluttered environment?
- How to reach such that the arm motion is smooth **smooth**?
- How to reach such that the path obeys **temporal constraints**?

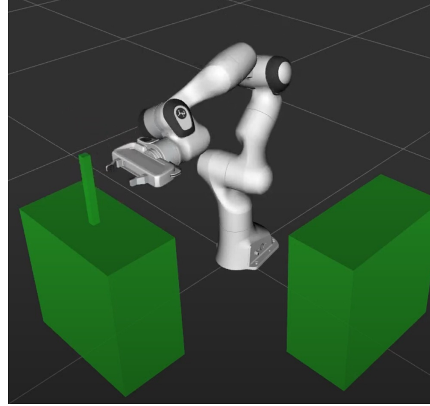


[Sundaralingam et al., "Curobo: Parallelized Collision-Free Robot Motion Generation", ICRA, 2023]

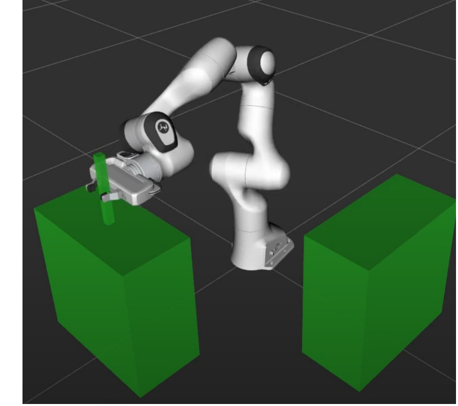
Motion Phases for Grasping Tasks



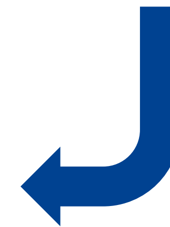
Initial Pose

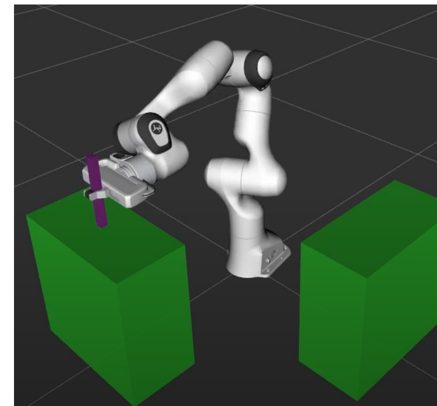


Pre-Grasp Pose

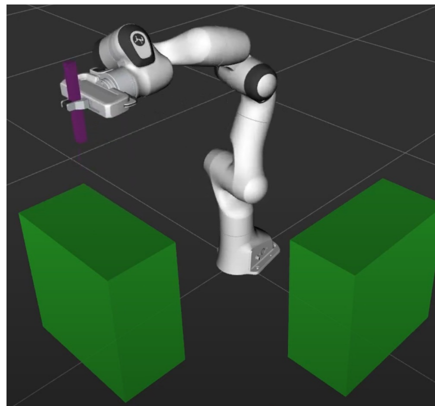


Grasp Pose

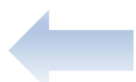




Gripper Close



Post-Grasp Pose



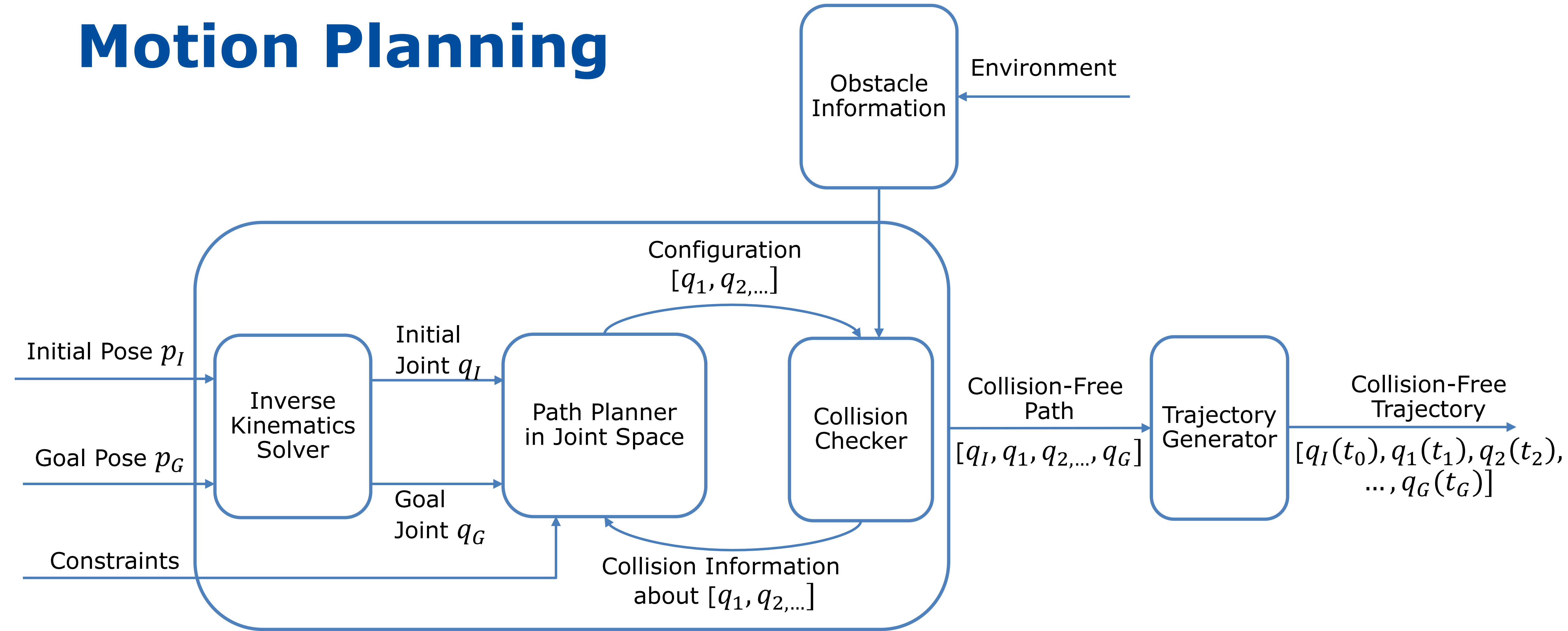
Place & Release

["MoveIt! Pick and Place Demo", 2018, www.youtube.com/watch?v=QBJPxx_63Bs]

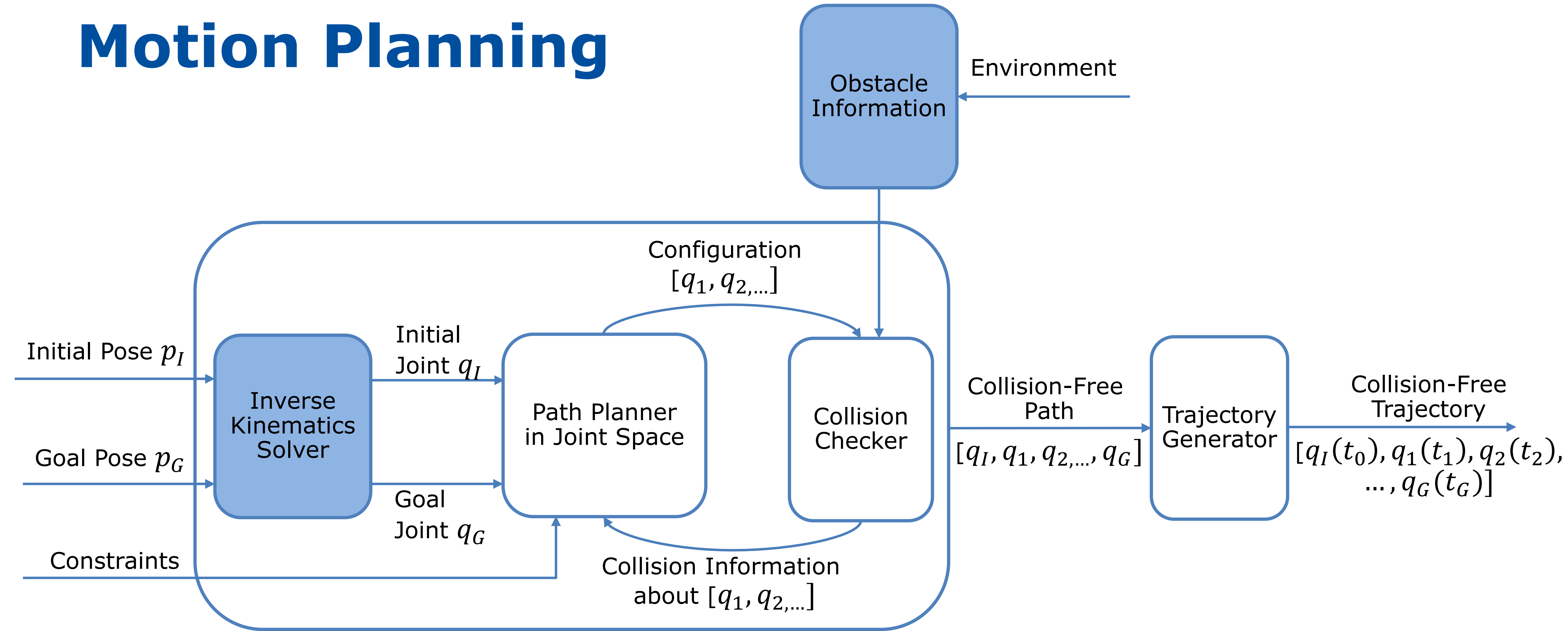
Steps in Manipulation Motion Planning

- Define **start** and **goal** end-effector poses
- Define **intermediate** poses if needed
- Add **constraints** if necessary
- Generate a **collision-free arm motion path**
- Parameterize a **trajectory** from the path

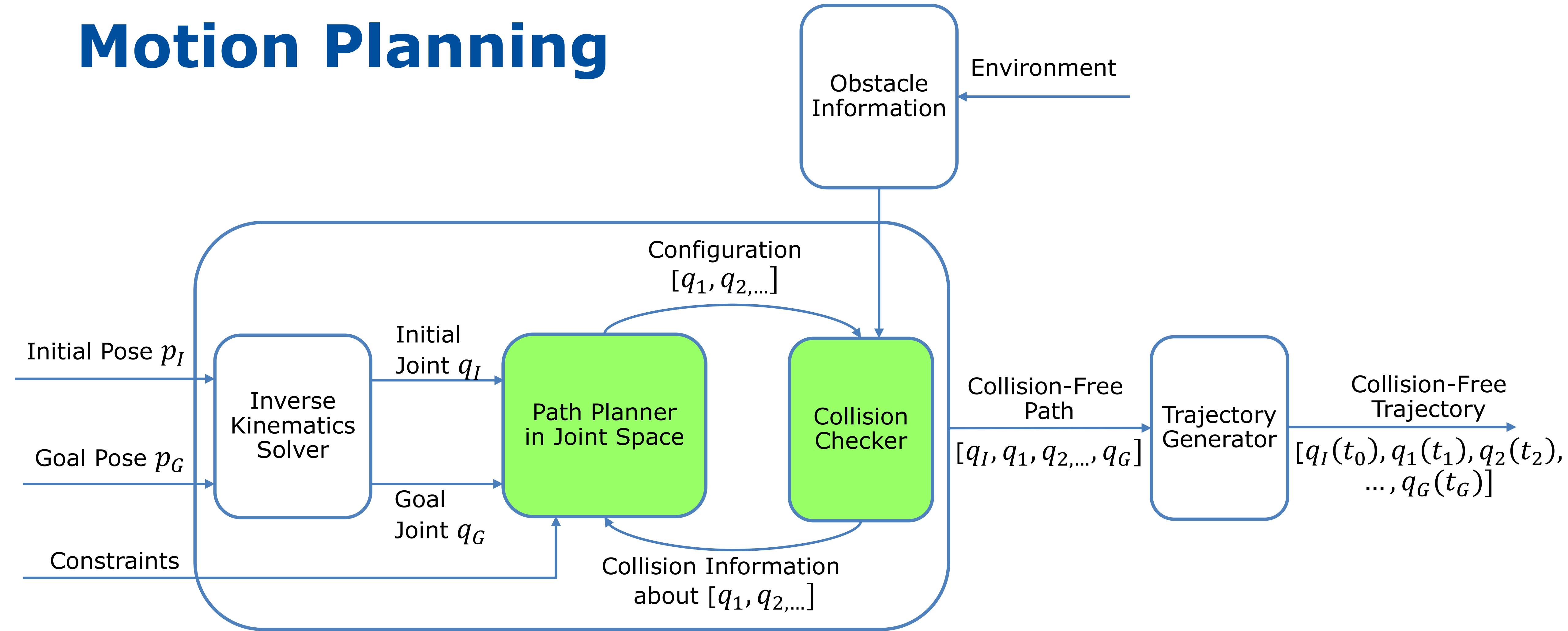
Motion Planning

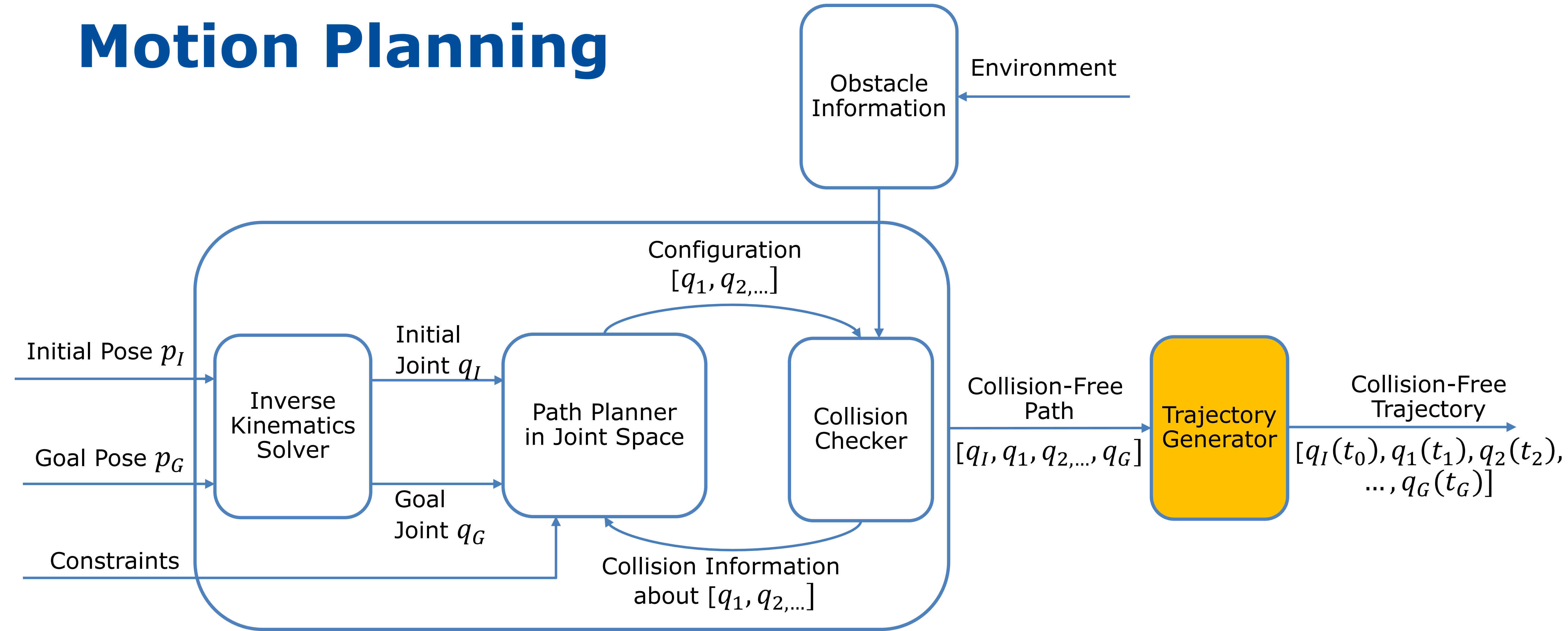


Motion Planning



Motion Planning





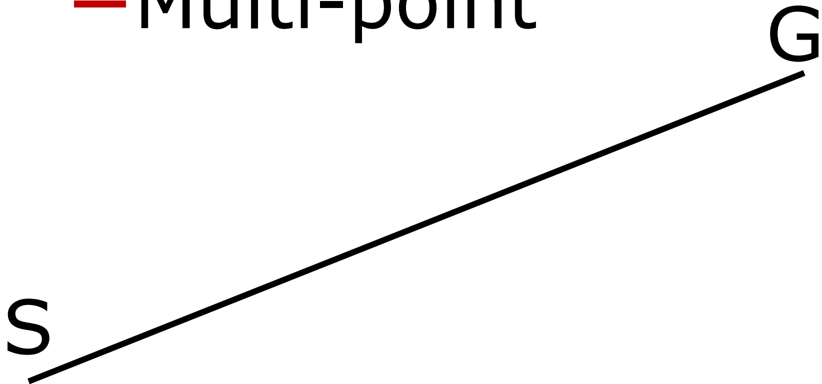
Motion Planning
Obstacle Information
Environment
Configuration $[q_1, q_{2,\dots}]$
Initial Pose p_I
Goal Pose p_G
Constraints
Inverse Kinematics Solver
Initial Joint q_I
Goal Joint q_G
Path Planner in Joint Space
Collision Checker
Collision Information about $[q_1, q_{2,\dots}]$
Collision-Free Path
$[q_I, q_1, q_{2,\dots}, q_G]$
Trajectory Generator
Collision-Free Trajectory
$[q_I(t_0), q_1(t_1), q_2(t_2), \dots, q_G(t_G)]$

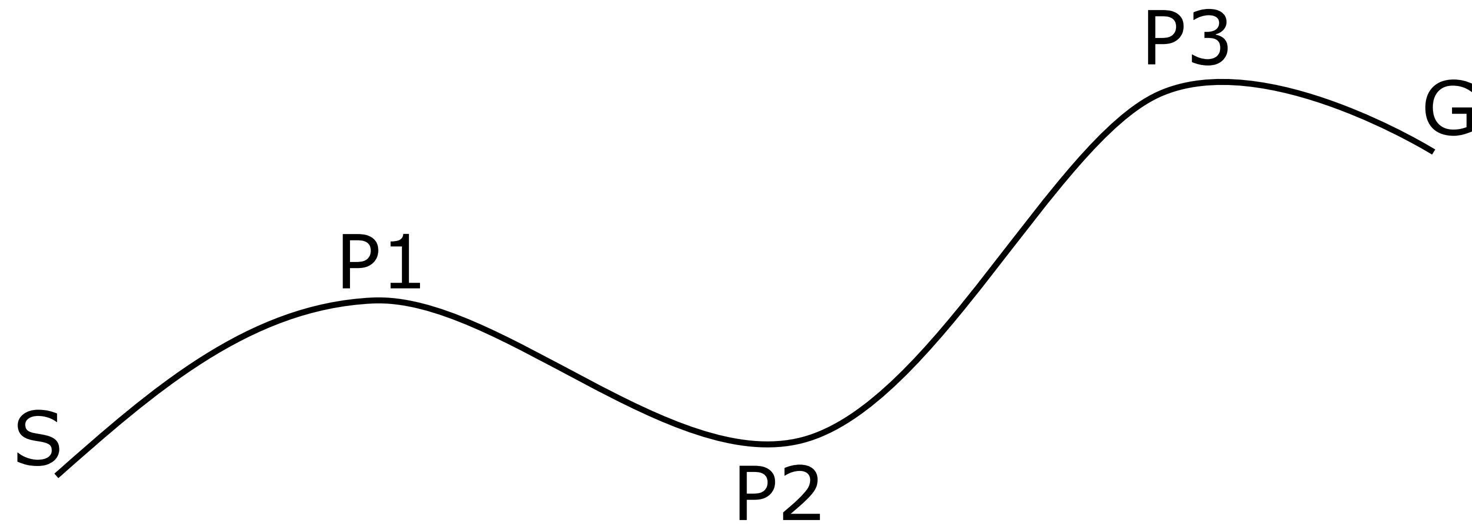
Concepts Needed for Motion Planning

- What is a path?
- What is a trajectory?
- What different kinds of robot spaces exist?
- How to plan a path?
- How to perform collision checking?
- How to generate a trajectory?

Path

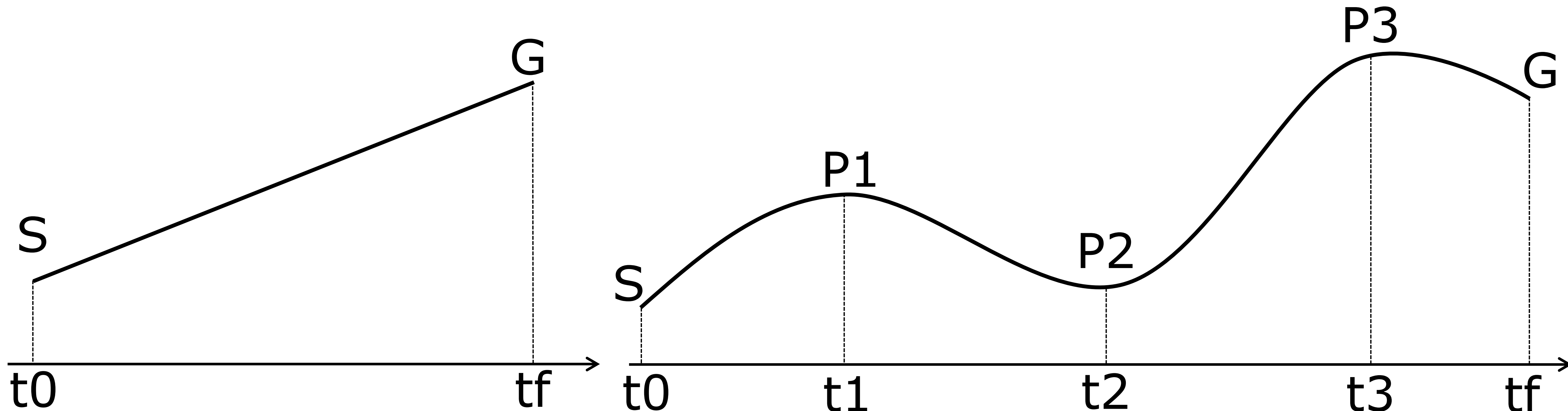
- Defines geometric sequence of positions
- Lacks timing and dynamic information
- Can be
  - Point to point
  - Multi-point





Trajectory

- Adds time parameterization to path
  - Initial and final times
  - Time optimality
- Specifies velocity, acceleration, jerk or torque along path



Robot Spaces

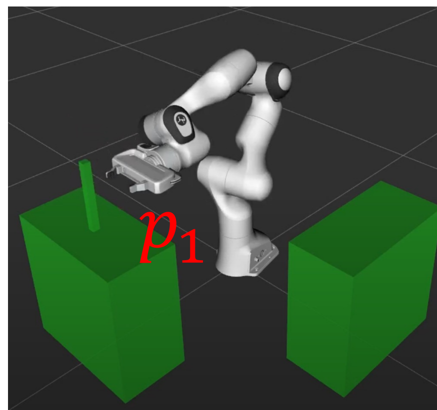
- Robots operate in multi-dimensional spaces
  - **Configuration space** (joint space): Space formed by the combination of robot joint angles
  - **Task space** (Cartesian space): Space described by end-effector position ($\mathbf{R}^3$) and orientation ($\mathbf{SO}(3)$), full ($\mathbf{SE}(3)$)
  - **Workspace**: Actual physical region that the end-effector can reach ($\mathbf{R}^2$ for mobile robot base, $\mathbf{R}^3$ for arms)
- Real-world tasks are specified in the **task space**
- However, robots are controlled in **configuration space**
- Obstacle regions are typically given **workspace**

Task Space

- More intuitive than joint space for manipulation planning
- Controls end-effector pose (position, orientation)
- Enables direct control of robot's environmental interaction
- Crucial for grasping, tool use, and human-robot collaboration

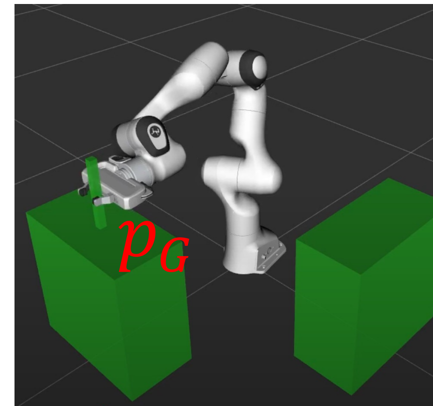
Task Space Motion: Pre-Grasp to Grasp

- End-effector must linearly approach the object
- Interpolate in task space from p_1 to p_G keeping the gripper orientation fixed
- Compute inverse kinematics
  $[q_1 = IK(p_1), q_2 = IK(p_2), \dots, q_G = IK(p_G)]$



Pre-Grasp Pose

Task Space Motion



Grasp Pose

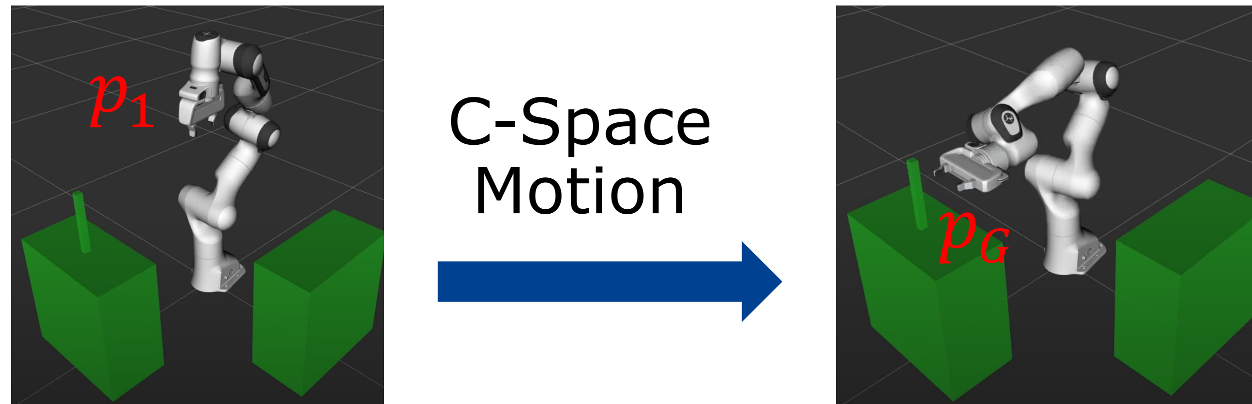
["MoveIt! Pick and Place Demo", 07/2018, www.youtube.com/watch?v=QBJPxx_63Bs]

Configuration Space (C-Space)

- Represents the space of the robot’s joint angle configurations
- For a robot with n joints, its configuration space is an n-dimensional space
- High-dimensional, capturing all possible configurations
- Essential for collision checking and motion planning

Example: Initial to Pre-Grasp

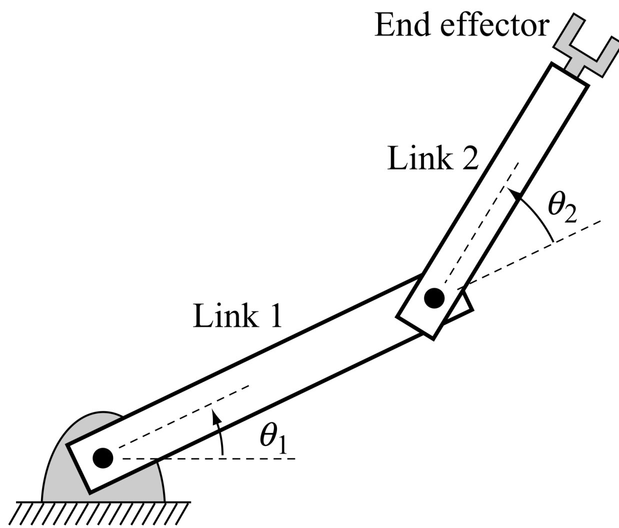
- Start and goal poses (p_1, p_G) defined in task space
- Compute inverse kinematics $q_1 = IK(p_1), q_G = IK(p_G)$
- Then, compute a path from q_1 to q_G in joint space
- In case of obstacles, generate a collision-free path $[q_1, q_2, \ldots, q_G]$ in joint space

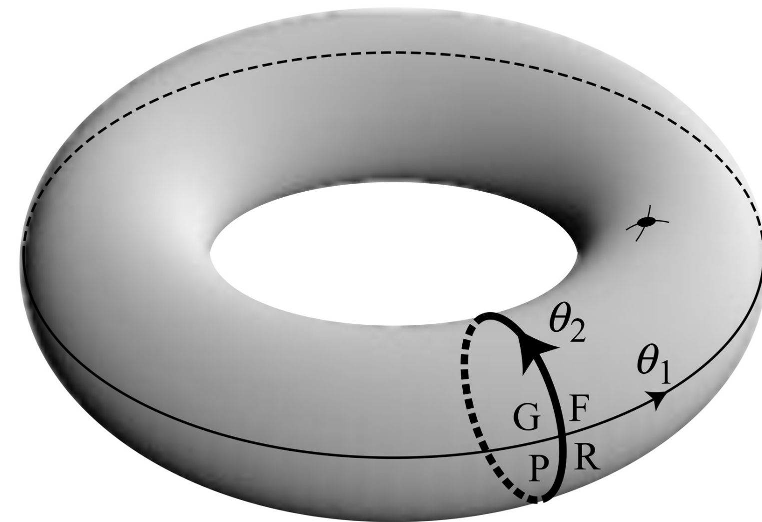


["MoveIt! Pick and Place Demo", 07/2018, www.youtube.com/watch?v=QBJPxx_63Bs]

C-Space of a Two-Joint Planar Arm

- Consider a 2-joint planar arm with no joint limits
- Each joint angle θ_i corresponds to a point on circle S^1
- C-Space is $S^1 \text{x}\, S^1 = T^2$ corresponding to a 2D torus
- Configuration q in C-space consists of 2 angles $q = (\theta_1, \theta_2)$

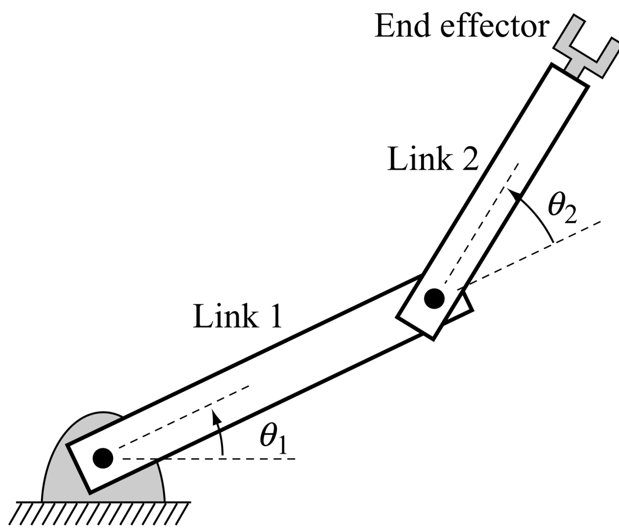


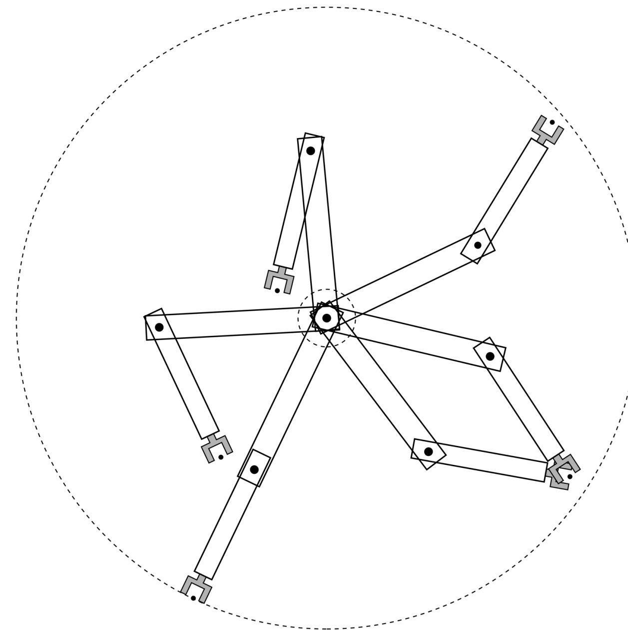


[Choset et al., *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT press, 2005]

Workspace of a Two-Joint Planar Arm

- For the 2-joint planar arm, the workspace is a 2D torus, i.e., a subset of $\mathbb{R}^2$
- All points in the 2D torus are reachable with two different configurations: elbow-up or elbow-down





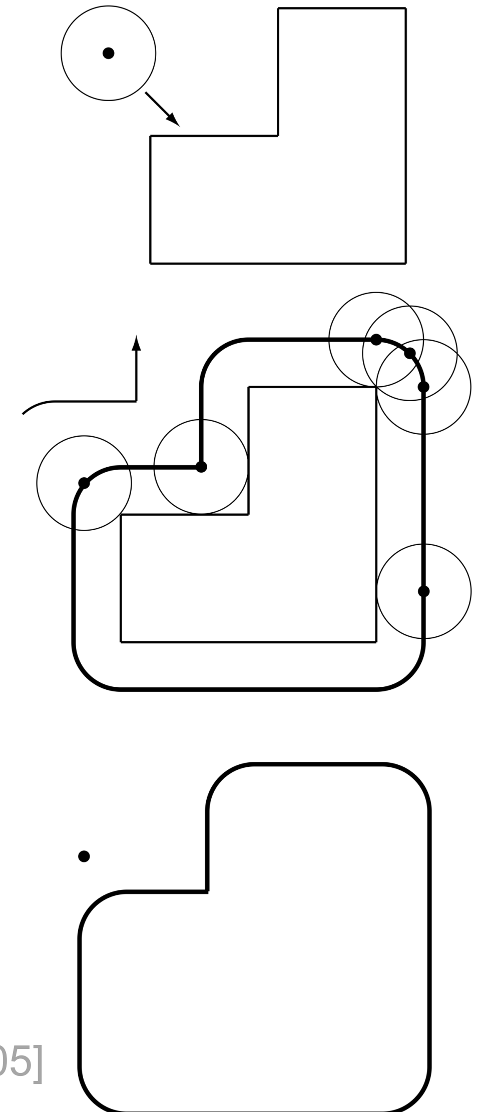
[Choset et al., *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT press, 2005]

C-Space Obstacles and Free Space

- Typically, complete description of the robot's geometry and of its reachable workspace is provided
- Let W be the entire physical environment space
- Let $O \subset W$ represent the **workspace obstacle region**
- Let $A(q) \subset W$ denote set of points occupied by the robot when in configuration $q \in C$
- **C-space obstacle**: $C_{obs} = \{q \in C | A(q) \cap O \neq \emptyset\}$
- **Free C-space**: $C_{free} = C \setminus C_{obs}$

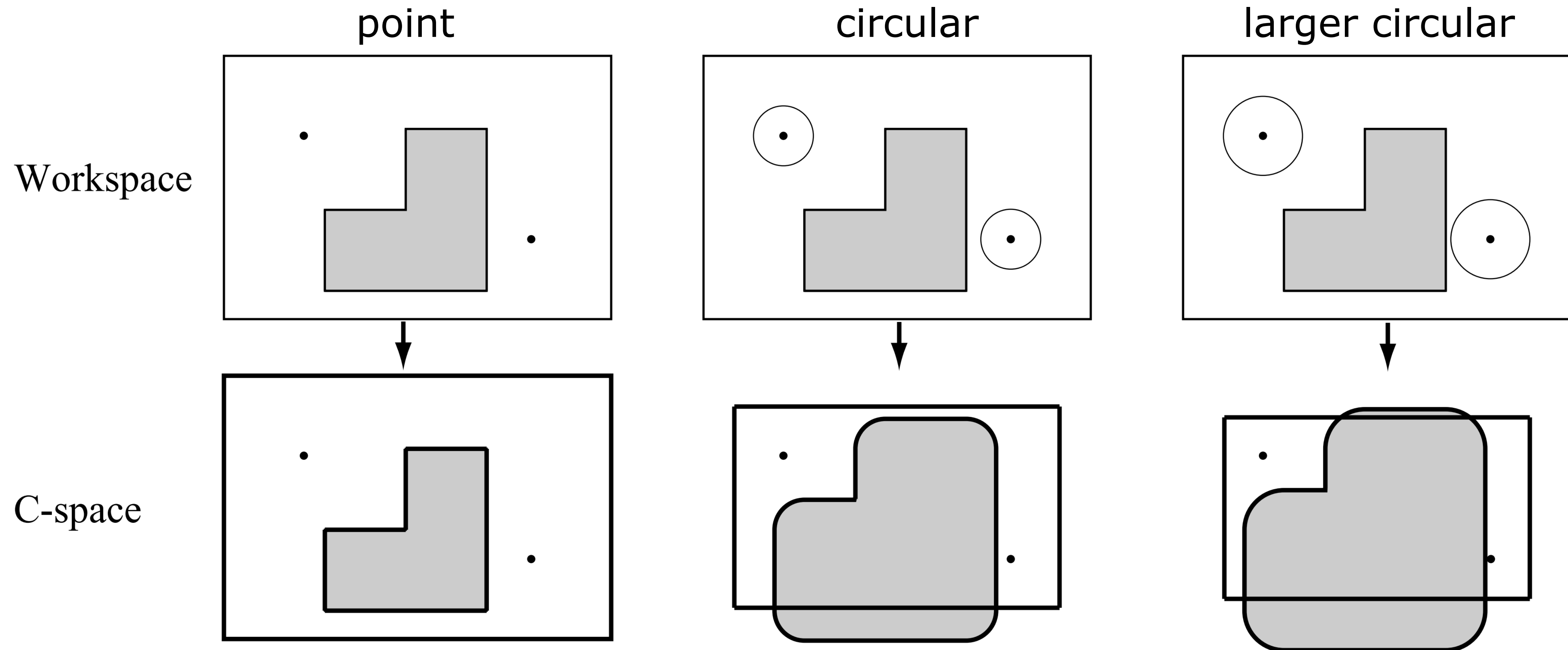
Workspace Obstacles to C-Space Obstacles

- Consider **circular mobile robot** with single polygonal obstacle as shown
- C-space obstacle found by “sliding” the robot around the obstacle
- Motion planning for circular robot in top figure is equivalent to motion planning for point in C-space



[Choset et al., *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT press, 2005]

Workspace and C-Spaces for Different Mobile Robots



[Choset et al., *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT press, 2005]

What about Transforming Workspace Obstacles to C-Space for n-Joint Arms?

- For circular mobile robots, converting workspace obstacles to C-space is relatively trivial due to
  - Symmetry of the robot
  - Workspace and C-space being low-dimensional $\mathbb{R}^2$
- Robot arms have **workspace in $\mathbb{R}^3$** and **task space in SE(3)**
- **C-space is T^n** with n number of joints
- Hence, conversion of workspace obstacles to C-space is **computationally infeasible**

Geometric Path Planning Problem

Given

- Robot's configuration space C
- Robot's workspace W
- Obstacle region $O \subset W$
- Initial configuration $\mathrm{q_I} \in C_{free}$
- Goal configuration $\mathrm{q_G} \in C_{free}$

Goal

For the query $(\mathrm{q_I}, \mathrm{q_G})$, compute a collision-free path $[q_I, q_1, q_{2,\ldots}, q_G]$ in the configuration space

Motion Planning Complexity

- Not easy to compute C_{obs} and C_{free}
- Exponential dependence on C-space dimensionality
- Two approaches for motion planning: Combinatorial and Sampling

Motion Planning: Combinatorial Algorithms

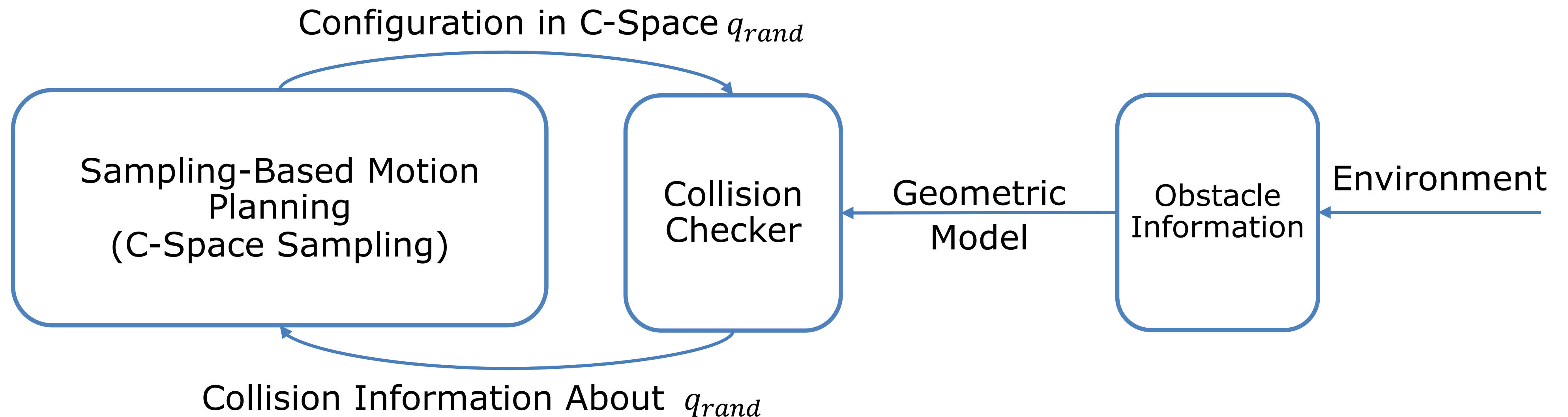
- **Complete**, i.e., either find a solution or will correctly report that no solution exists
- **Exact**, i.e., find paths through C-space w/o resorting to approximations
- However, **NP-hard**

Motion Planning: Sampling-Based Methods

- **Weaker guarantee**: Will find a solution eventually if one exists, but no guarantee on failure report in finite time in case none exists
- **Approximate**: Uses approximation of C-space for collision checking

Sampling-Based Motion Planning

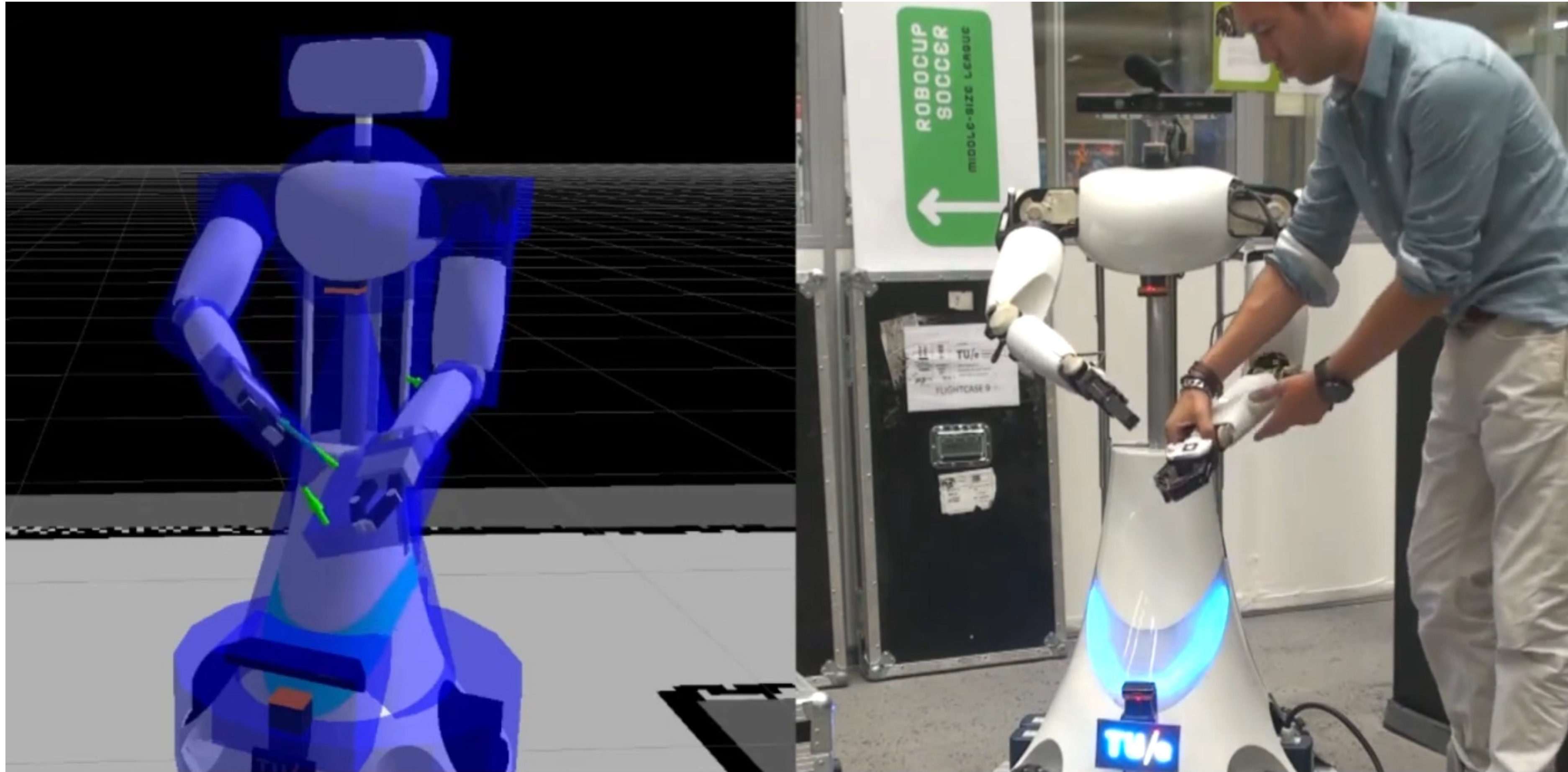
- Avoids explicit construction of C_{obs}
- Instead, performs search that **probes C-space with sampling**
- Collision checking **without exact geometric models**



Geometric Models

- Representations for known objects, i.e., robot and known obstacles
  - Primitives (rectangle, cylinder, box, sphere)
  - Meshes
- Representations for unknown objects, i.e., sensed obstacles
  - Point clouds
  - Occupancy maps
- See Chapter 3 for more details on 3D world representations

Collision Checking with Primitives



[Tech United Eindhoven, “Reactive Collision Avoidance With the AMIGO Robot”, 2016, www.youtube.com/watch?v=7GcLU9I65eM]

Collision Detection

- For a particular configuration $q \in T^n$, check if $q \in C_{free}$ or $q \in C_{obs}$
- Collision detection can be a continuous or Boolean function
- Boolean function $\phi: C \rightarrow \{TRUE, FALSE\}$

$$q \in C_{obs} \rightarrow \phi(q) = TRUE, else\ FALSE$$

- **Boolean functions** typically used **in sampling-based planning** for accepting or rejecting a q sampled from T^n
- Alternatively, distance function $d: C \rightarrow [0, \infty)$
- **Distance function** used for **optimization-based planning**, d used to assign a cost value to q

Two-Phase Collision Detection

- For *n*-joint robots like arms collision detection is a two-phase process
- **Broad Phase:**
  - Avoid expensive computation for links far away from each other
  - Place simple bounding boxes around each links
  - Perform simple overlap test to determine whether costly checking is needed

Two-Phase Collision Detection

- **Narrow Phase:**
  - Further process individual pairs of bodies that overlap in broad-phase check
  - Perform more expensive checking for collision

Sampling-Based Motion Planning

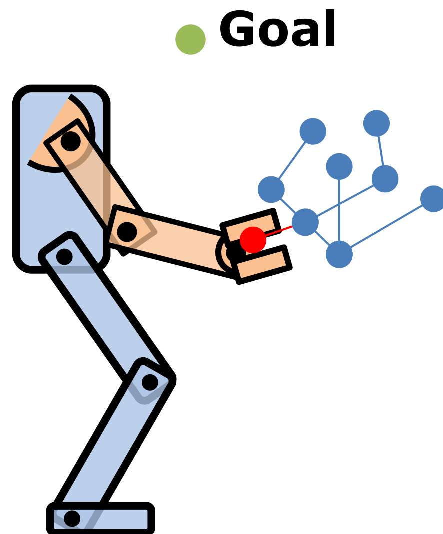
Different types of sampling-based planners

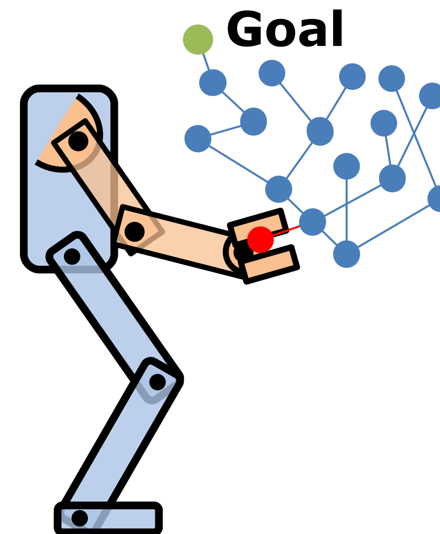
- **Multi-query** (e.g., probabilistic roadmap approach)
  - Constructs a **"roadmap" once** to map the C_{free}
  - Multiple queries in same environment using the roadmap

- **Single-query** (e.g., RRTs)
  - Build **tree data structures** on the fly for a given query
  - Explore part of C-space to solve specific query as fast as possible

Rapidly Exploring Random Trees (RRTs)

- Explore the configuration space by expanding incrementally from an initial configuration
- Explored space corresponds to a **tree rooted at the initial configuration**
- Basic principle: **Sample configuration** and compute **local connection to nearest neighbor**



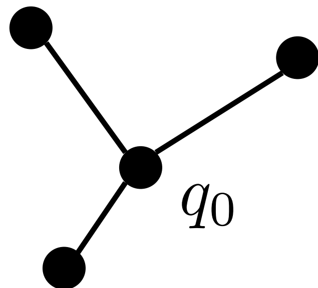


RRTs: General Algorithm

Given: Configuration space C and initial configuration q_0

$G.\text{init}(q_0)$
repeat
  $q_{rand} \rightarrow \text{RANDOM_CONFIG}(\mathcal{C})$
  $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
  $G.\text{add_edge}(q_{near}, q_{rand})$
until *condition*

sample **random configuration**



tree constructed so far

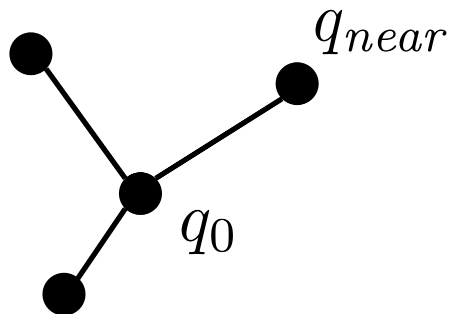
RRTs: General Algorithm

Given: Configuration space *C* and initial configuration q_0

$G.\text{init}(q_0)$
repeat
  $q_{rand} \rightarrow \text{RANDOM_CONFIG}(\mathcal{C})$
  $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
  $G.\text{add_edge}(q_{near}, q_{rand})$
until *condition*

Find closest vertex in *G* using a **distance function**

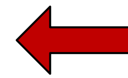
$$\rho : \mathcal{C} \times \mathcal{C} \rightarrow [0, \infty)$$



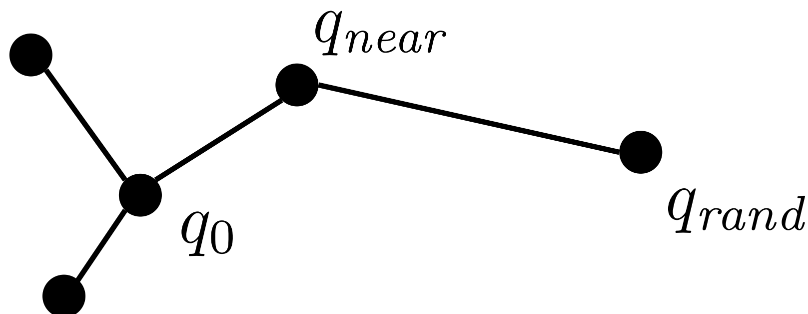
RRTs: General Algorithm

Given: Configuration space *C* and initial configuration q_0

$$
\begin{aligned}
&G.\text{init}(q_0) \\
&\textbf{repeat} \\
&\quad q_{rand} \rightarrow \text{RANDOM_CONFIG}(\mathcal{C}) \\
&\quad q_{near} \leftarrow \text{NEAREST}(G, q_{rand}) \\
&\quad G.\text{add_edge}(q_{near}, q_{rand}) \\
&\textbf{until } \textit{condition}
\end{aligned}
$$

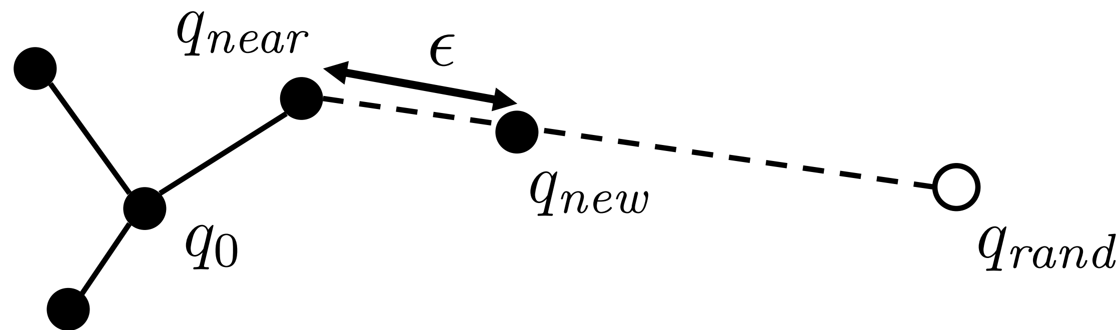


Connect q_{near} with q_{rand} using a **local planner**



Extension of the Tree: Constraints

- Need to consider obstacles: Check local connection for collisions and add edge **only if path collision-free**
- Use **fixed incremental step size** so that the likelihood of a collision-free path is increased
- Terminate when q_{new} is close to the desired q_{goal}



Bias Towards the Goal

- During tree expansion, pick the goal instead of a random node with some probability (5-10%)
- Why not picking the goal at each iteration?
- Avoiding running into local minima (due to obstacles or other constraints) instead of exploring the space

Bidirectional RRTs

- High-dimensional, complex motion planning problems require more effective methods: **bidirectional search**
- Grow **two RRTs**, one from q_0 and one from q_G
- In every other step, try to extend each tree towards q_{new} of the other tree

RRT-Connect: Basic Concept

- **Grow two trees**: from start and end node (start and goal configurations of the robot)
- Pick a **random** configuration: q_{rand}
- Find the **nearest** node in one tree: q_{near}
- Extend the tree from the nearest node by **taking a step** towards the random node to get q_{new}
- **Extend** the **other tree** towards that q_{new} from nearest node in the tree
- **Return the solution** path when the distance between q_{new} and the nearest node in the second tree is close enough

Extend Function

Returns

- **Trapped**: Not possible to extend the tree due to collisions or constraints
- **Extended**: Performed a step from q_{near} towards q_{rand}, generated q_{new}
- **Reached**: Trees connected, path found

RRT-Connect

RRT_CONNECT ($q_{init,}$ q_{goal}) {

$T_a.init(q_{init})$; $T_b.init(q_{goal})$;

for k = 1 to K do

K=max number of iterations

q_{rand} = RANDOM_CONFIG();

if not (EXTEND(T_a, q_{rand}) = Trapped) then

First tree has been extended, try to extend second tree

if (EXTEND(T_b, q_{new}) = Reached) then

Return PATH(T_a, T_b);

Success: trees connected

SWAP(T_a, T_b);

Return Failure;

Max number of iterations reached

}

[Kuffner&Lavalle, ICRA 2000]

RRTs – Properties (1)

- Good **balance** between **greedy search and exploration**
- **Effective** for **high-dimensional** configuration spaces
- Produce **non-optimal paths**: solutions are typically jagged and may be overly long
- **Post-processing** such as smoothing is necessary
- Generated paths are **not repeatable** and **unpredictable**
- Rely on a distance metric (e.g., Euclidean)

RRTs – Properties (2)

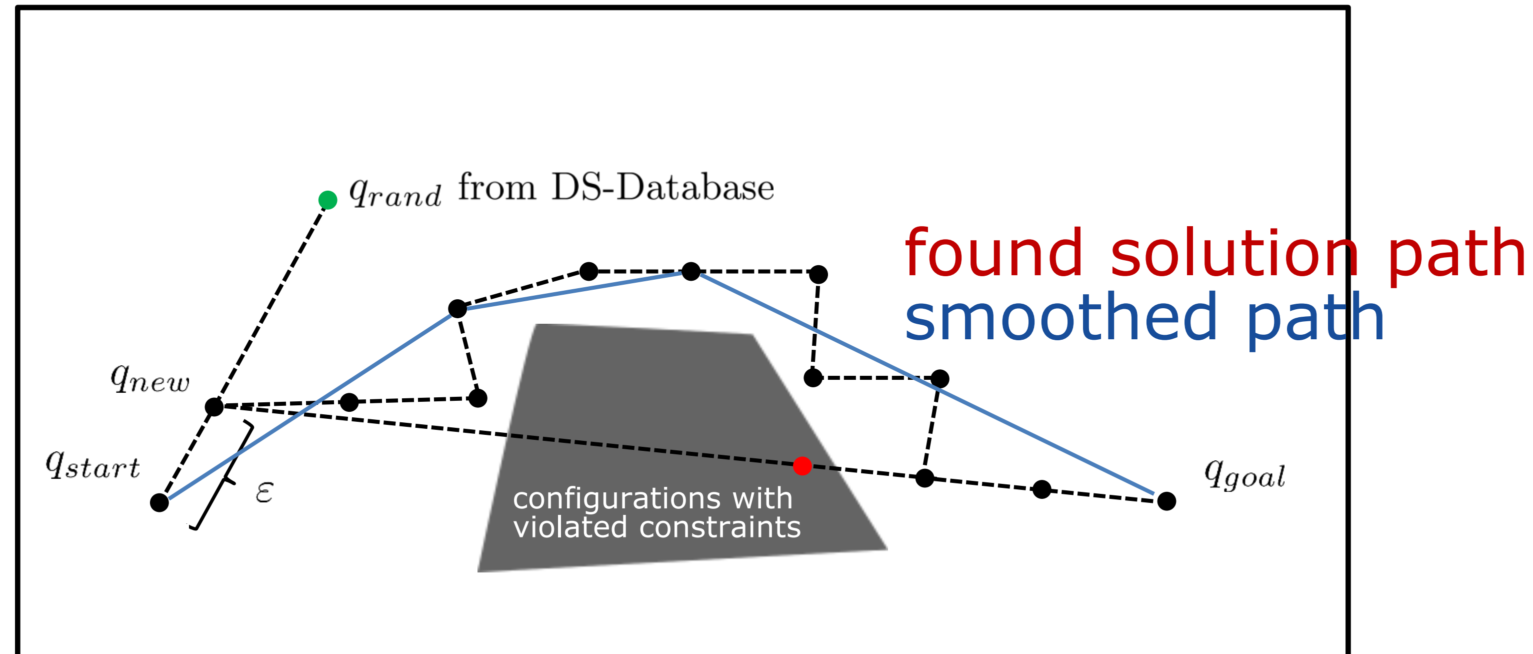
- **Probabilistic completeness**: Probability of finding a solution if one exists approaches 1
- **Unknown** rate of **convergence**
- When there is no solution (path is blocked due to obstacles or other constraints), the **planner may run forever**
- To avoid endless runtime, the search is **stopped** after a certain number of iterations

Considering Constraints for Humanoid Motion Planning

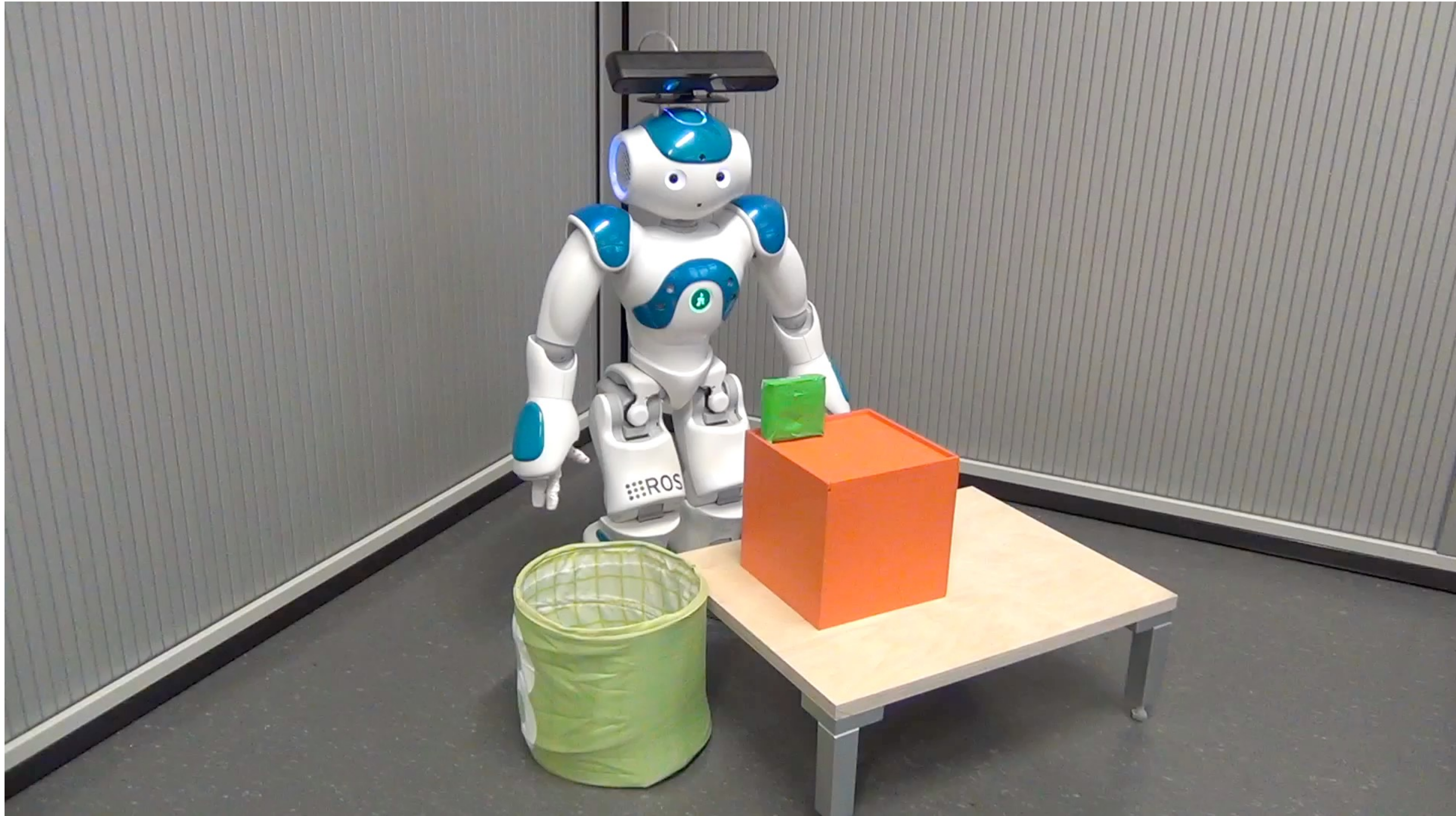
- When randomly sampling configurations, most of them will not be valid since they cause the robot to lose its balance
- Use a set of **predetermined statically stable** double support configurations from which to sample q_{rand}
- In the extend function: Check q_{new} for **joint limits, self-collision, collision with obstacles, and whether it is statically stable**

RRT-Connect: Considering Constraints

- Check for constraint violation in configuration space
- Smooth path after a solution is found



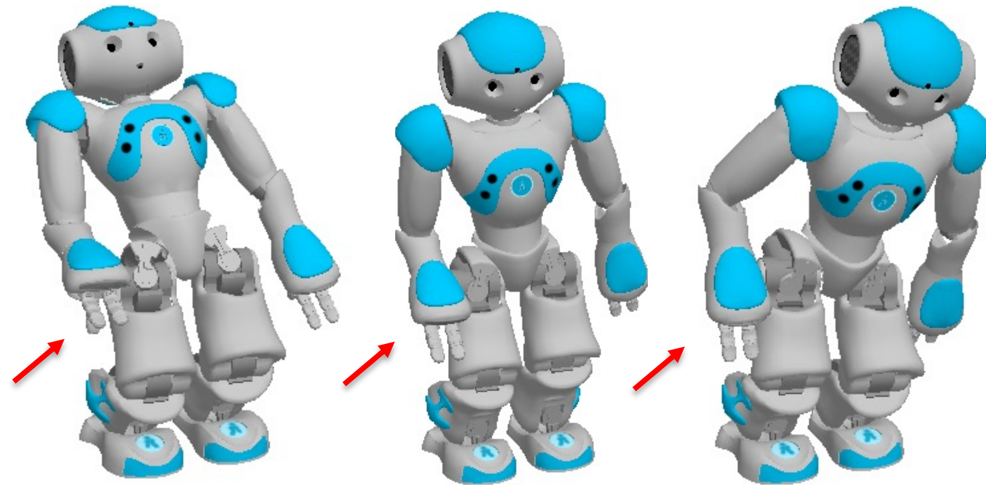
Path Execution: Pick and Place



Past Execution: Grabbing Into a Cabinet

Goal Configuration

- How to actually determine the robot’s goal configuration for a given manipulation task?
- Use inverse reachability maps (see previous chapter)



all valid goal configurations
for the same desired end effector pose

Literature Motion Planning

- *Principles of Robot Motion: Theory, Algorithms, and Implementations,* Choset, Lynch, Hutchinson, Kantor, and Burgard, MIT press, 2005
- *Planning Algorithms,* LaValle, Cambridge University Press, 2006
- *Motion planning.* In *Springer Handbook of Robotics ,* Kavraki and LaValle (pp. 139-162), Springer International Publishing, 2016
- *Curobo: Parallelized Collision-Free Robot Motion Generation,* Sundaralingam, Hari, Fishman, Garrett, Van Wyk, Blukis, Millane, Oleynikova, Handa, Ramos, and Ratliff, IEEE/RAS Int. Conf. on Robotics and Automation (ICRA), 2023
- *HPP: A New Software for Constrained Motion Planning,* Mirabel, Tonneau, Fernbach, Seppälä, Campana, Mansard, and Lamiraux, IEEE/ RSJ Int. Conf. on Int. Robots and Systems (IROS), 2016
- *RRT-Connect: An Efficient Approach to Single-Query Path Planning* Kuffner and LaValle , IEEE International Conference on Robotics & Automation (ICRA), 2000
- *Whole-Body Motion Planning for Manipulation of Articulated Objects* Burget, Hornung, and Bennewitz, M. IEEE International Conference on Robotics & Automation (ICRA), 2013

Trajectory Generation