Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MIN Faculty
Department of Informatics

# Introduction to Robotics

## Lecture 10


**Michael Görner, Jianwei Zhang**
[goerner, zhang]@informatik.uni-hamburg.de

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
**Technical Aspects of Multimodal Systems**

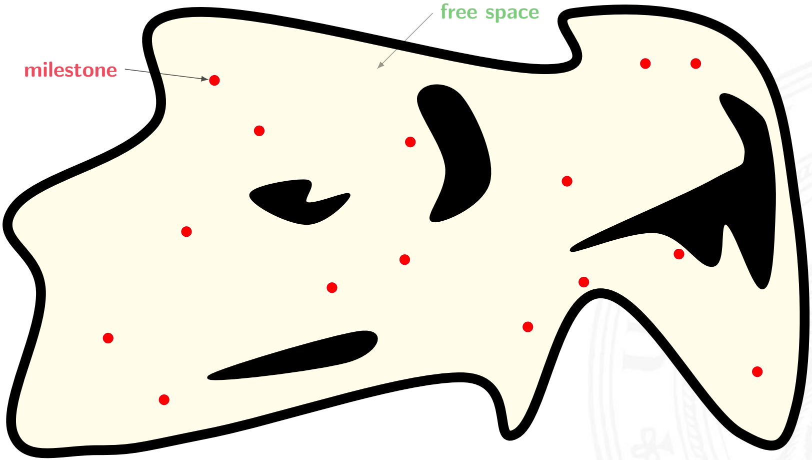

June 26, 2020

# Path Planning

# Idea

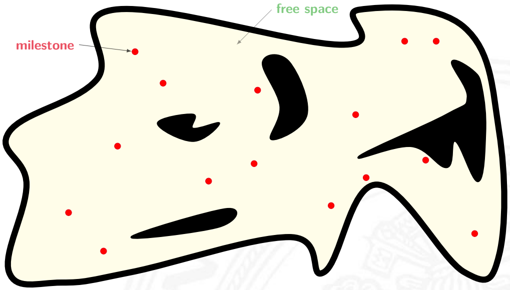
- Planning on graphs of reasonable size is simple
- Operating on grids ignores continuous spaces in $\mathcal{X}_{free}$
- Instead rely on **Probabilistic Sampling** to represent the space

- Planning on graphs of reasonable size is simple
- Operating on grids ignores continuous spaces in $\mathcal{X}_{free}$
- Instead rely on **Probabilistic Sampling** to represent the space



Key questions:

- How to generate the samples?
- How can the samples be connected to form a planning graph?
- How many samples do you need to describe the space?



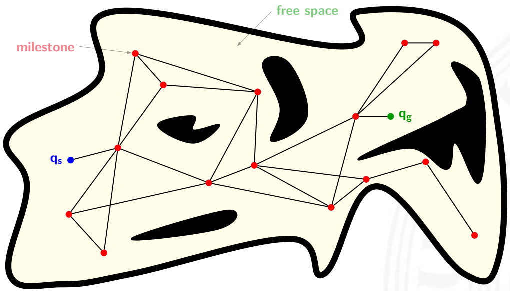

Abstract C-space with sampled valid states

# Probabilistic Road Maps

Proposed by Lydia E. Kavraki et.al. 1996 [1]

Two Step algorithm:

1. Construction Phase - Build Roadmap
2. Query Phase - Connect start and goal to graph and solve graph search




Abstract C-space with sampled valid states

# Probabilistic Road Maps - Algorithm



**Algorithm:** sPRM

$V \leftarrow \{x_{\mathrm{init}}\} \cup \{\mathtt{SampleFree}_i\}_{i=1,\ldots,n};\ E \leftarrow \emptyset;$

**foreach** $v \in V$ **do**

  $U \leftarrow \mathtt{Near}(G = (V, E), v, r) \setminus \{v\};$

  **foreach** $u \in U$ **do**

    **if** $\mathtt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$

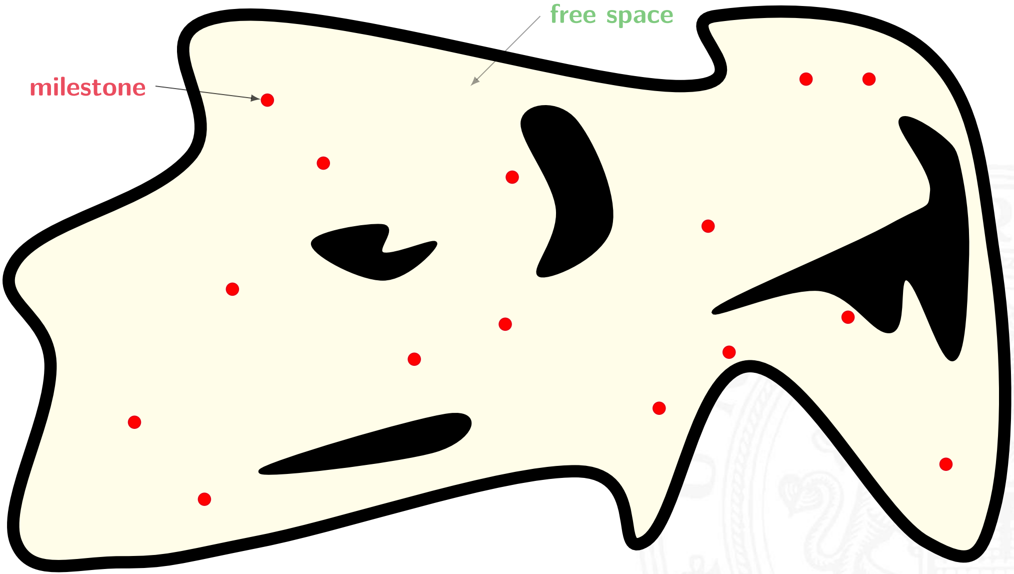
**return** $G = (V, E);$

Adapted from [2]

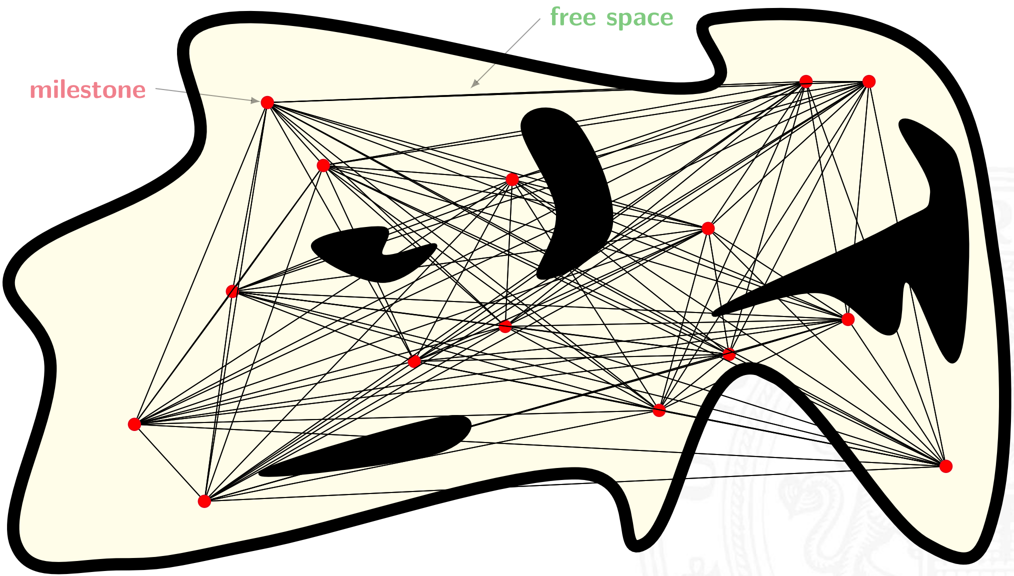
# Milestones and Roadmap - Construction



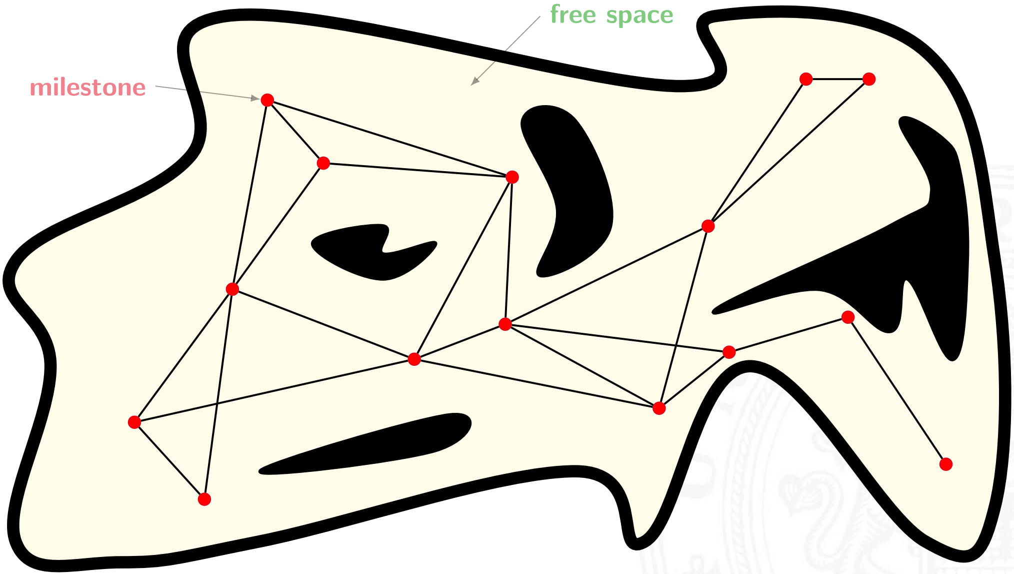
# Milestones and Roadmap - Construction



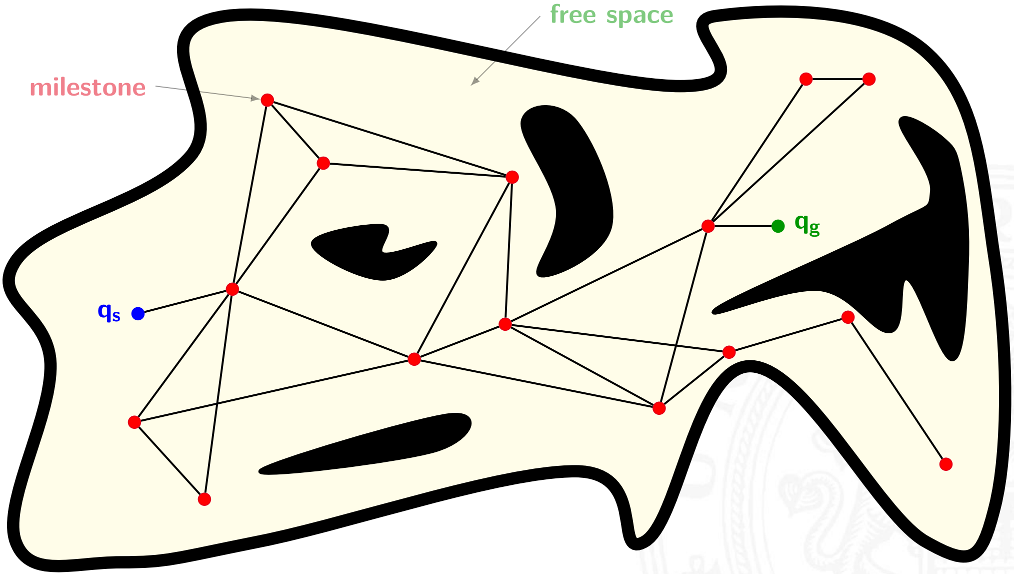
# Milestones and Roadmap - Construction



# Milestones and Roadmap - Construction

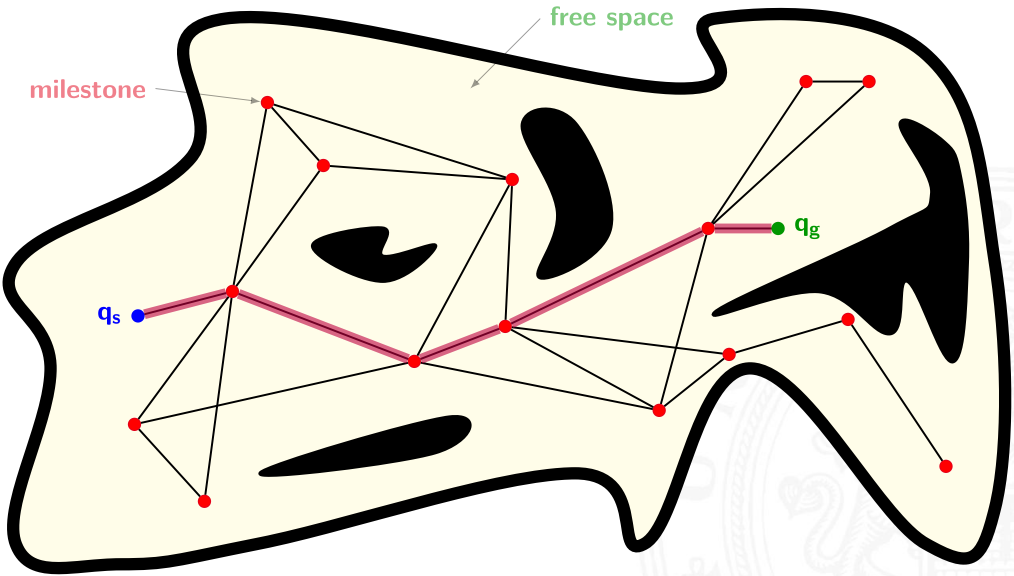


# Milestones and Roadmap - Query



# Milestones and Roadmap - Query





# Probabilistic Road Maps - Aspects



**Algorithm:** sPRM

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\dots,n}$; $E \leftarrow \emptyset$;
**foreach** $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\}$;
  **foreach** $u \in U$ **do**
    **if** $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
**return** $G = (V, E)$;

# Probabilistic Road Maps - Aspects



**Algorithm:** sPRM

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\dots,n};\ E \leftarrow \emptyset;$
**foreach** $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\};$
  **foreach** $u \in U$ **do**
    **if** $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
**return** $G = (V, E);$

- `SampleFree` - Sample states from $\mathcal{X}_{free}$
- `Near` - Choose Distance metric and threshold
- `CollisionFree(v,u)` - Check motion **between** states for collisions

# Probabilistic Road Maps - Aspects

**Algorithm:** sPRM

```
V ← {x_init} ∪ {SampleFree_i}_{i=1,...,n}; E ← ∅;
foreach v ∈ V do
    U ← Near(G = (V, E), v, r) \ {v};
    foreach u ∈ U do
        if CollisionFree(v, u) then E ← E ∪ {(v, u), (u, v)}
return G = (V, E);
```

SampleFree – sample states from $\mathcal{X}_{free}$

- Traditionally: Rejection Sampling
  Take samples uniformally, add sample if $x \in \mathcal{X}_{free}$
- Alternatives:
  - Projective Sampling: Replace samples $x \in \mathcal{X}_{obs}$ by closest state $x' \in \mathcal{X}_{free}$
  - Generative Sampling: For a sufficient **parameterized** space $\mathcal{X}'_{free} \subseteq \mathcal{X}_{free}$:
    Sample from $\mathcal{X}'_{free}$ via parameters

# Probabilistic Road Maps - Aspects

**Algorithm:** sPRM

```
V ← {x_init} ∪ {SampleFree_i}_{i=1,...,n}; E ← ∅;
foreach v ∈ V do
    U ← Near(G = (V, E), v, r) \ {v};
    foreach u ∈ U do
        if CollisionFree(v, u) then E ← E ∪ {(v, u), (u, v)}
return G = (V, E);
```

`Near` - choose distance metric and threshold

- Traditional C-space metric: $L_1$ distance
- Obvious alternatives: weighted $L_1$ distance, $L_2$ distance

- Higher threshold: more negative collision checks
- Lower threshold: slower graph building

# Probabilistic Road Maps - Aspects



**Algorithm:** sPRM

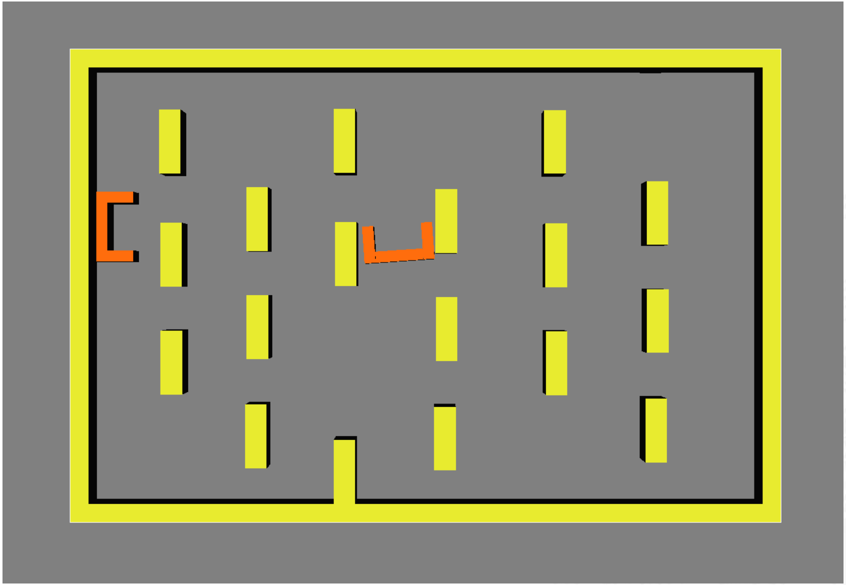
$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\dots,n};\ E \leftarrow \emptyset;$
**foreach** $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\};$
  **foreach** $u \in U$ **do**
    **if** $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
**return** $G = (V, E);$

`CollisionFree(v, u)` - Local Planning

- Traditionally collision-checking tests **one** state
- Interpolate states between $\langle v, u \rangle$ and check those
  - Fixed step size in C-space can imply huge motions in workspace!
- Continuous collision checking (CCD):
  - Current systems rely on primitive motions
  - Robot links move in complex splines

# Example

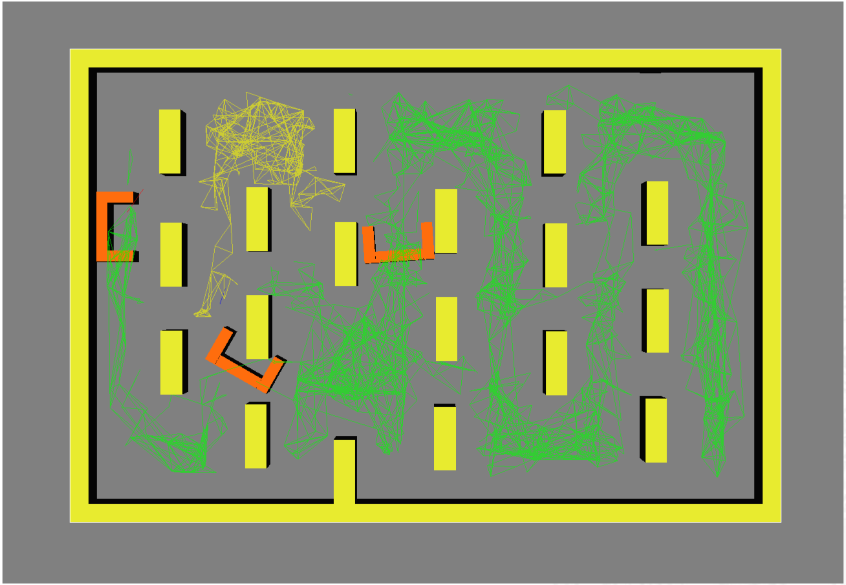




3dof planning problem

# Example





3dof planning problem

# Single and Multi-Query Planning

**Definition**

If only a single path is requested in a potentially changing scene, this is called **single-query** planning. If datastructures remain valid between motion requests, this is called **multi-query** planning.

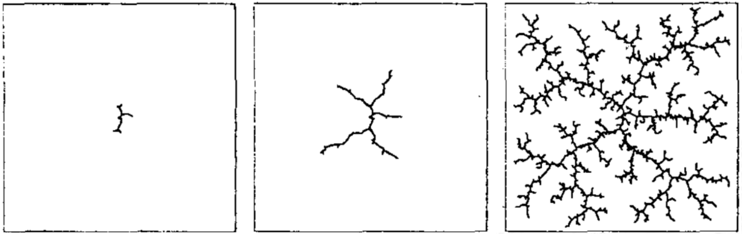
PRM solves a multi-query problem by building an undirected graph.

For single-shot planning, the graph search can be avoided altogether.

Proposed by Kuffner and LaValle 2000 [3]

Instead of building a graph, grow a tree from the start state.

If for any leaf state $x \in \mathcal{X}_{goal}$, a solution is found.



RRT at multiple stages of extension

# Rapidly-exploring Random Trees (RRT) - Algorithm



**Algorithm 3:** RRT

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
**for** $i = 1, \ldots, n$ **do**
    $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
    $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
    $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$ ;
    **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
        $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$ ;
**return** $G = (V, E)$;

Adapted from [2]

# Rapidly-exploring Random Trees (RRT) - Algorithm

**Algorithm 3:** RRT

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
**for** $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$ ;
  **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$ ;
**return** $G = (V, E)$;

`Steer(x,y)` - Compute new state $x'$

- Move from $x$ towards $y$: $\|y - x'\| < \|y - x\|$
- $\|x - x'\| < \eta$ to limit step size
- Alternatively compute closest $x' \in \mathcal{X}_{free}$ reachable via straight motion

Adapted from [2]

# Rapidly-exploring Random Trees (RRT) - Algorithm



**Algorithm 3:** RRT

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
**for** $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$ ;
  **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$ ;
**return** $G = (V, E)$;

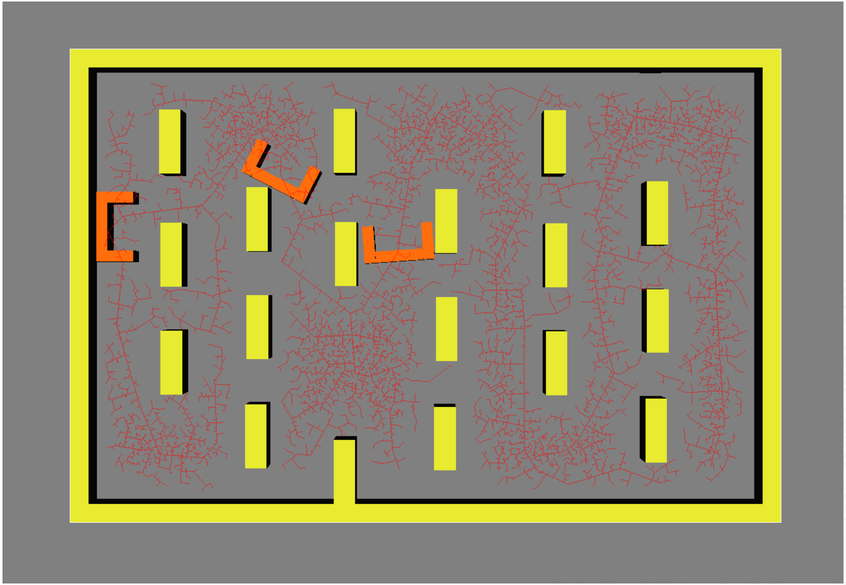
`SampleFree` – sample states from $\mathcal{X}_{free}$

- Traditionally: uniform sampling
- To improve heuristically, a **Goal Bias** can be added
  - Low fraction of samples are sampled from $\mathcal{X}_{goal}$
  - Required if $\mathcal{X}_{goal}$ is small in $\mathcal{X}$

Adapted from [2]

# Example





RRT graph of an example

# Bi-Directional Search

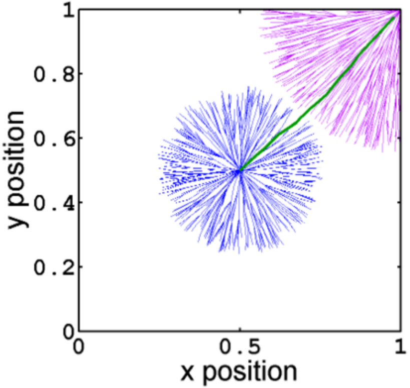
In robotics, start and goal are often in constraint areas of $\mathcal{X}_{free}$, e.g., close to obstacles.

The **transition phase** between these states is often quite flexible.

Instead of growing a single tree towards the goal

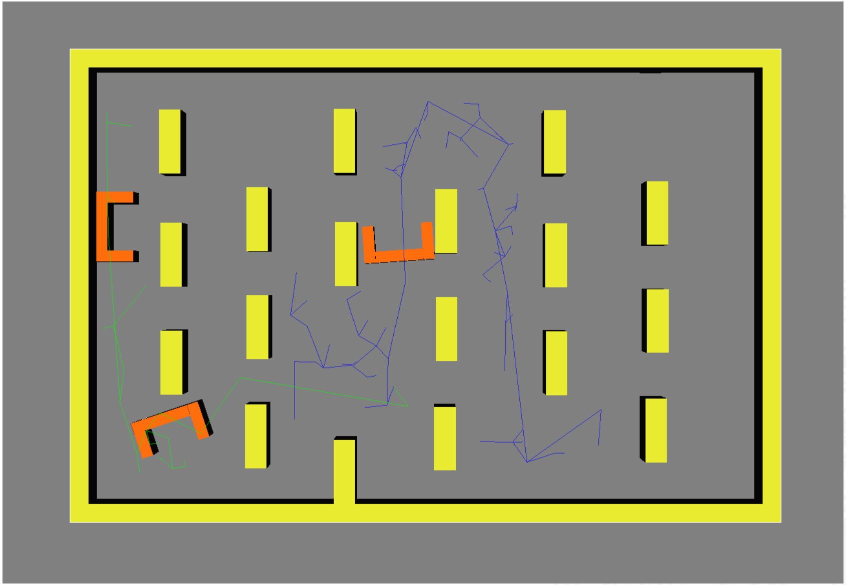
- Grow two trees from start and goal each.
- Attempt to connect them at each step.

In practice, this speeds up planning to the first solution significantly.




Bi-directional search trees [4]

# RRT-Connect - Example



RRT-Connect for an example

# Expansive Space Trees

PRM and RRT sample random configurations from $\mathcal{X}_{free}$.
Thus they also sample in areas which are already well-represented by milestones.

**Definition**

The *density* around a state $x$ can be represented by the cardinality of its neighborhood within a distance $d$: $|N_d(x)|$.

**Ideas**

- Sample next expansion step weighted by inverse density $w(x) = \frac{1}{|N_d(x)|}$
- Stochastically reject samples in high-density areas

---

Hsu et.al. 1997 [5]

PRM and RRT sample random configurations from $\mathcal{X}_{free}$.
Thus they also sample in areas which are already well-represented by milestones.

**Definition**

The *density* around a state $x$ can be represented by
the cardinality of its neighborhood within a distance $d$: $|N_d(x)|$.

**Ideas**

- Sample next expansion step weighted by inverse density $w(x) = \frac{1}{|N_d(x)|}$
- Stochastically reject samples in high-density areas

Hsu et.al. 1997 [5]

# Expansive Space Trees



PRM and RRT sample random configurations from $\mathcal{X}_{free}$.
Thus they also sample in areas which are already well-represented by milestones.

**Definition**
The *density* around a state $x$ can be represented by
the cardinality of its neighborhood within a distance $d$: $|N_d(x)|$.

**Ideas**

- Sample next expansion step weighted by inverse density $w(x) = \frac{1}{|N_d(x)|}$
- Stochastically reject samples in high-density areas

Hsu et.al. 1997 [5]

**Algorithm** *expansion*

1. Pick a node $x$ from $V$ with probability $1/w(x)$.
2. Sample $K$ points from $N_d(x) = \{q \in \mathcal{C} \mid dist_c(q, x) < d\}$, where $dist_c$ is some distance metric of $\mathcal{C}$. ($K$ and $d$ are parameters.)
3. **for** each configuration $y$ that has been picked **do**
4. calculate $w(y)$ and retain $y$ with probability $1/w(y)$.
5. **if** $y$ is retained, $clearance(y) > 0$ and $link(x, y)$ returns YES
6. **then** put $y$ in $V$ and place an edge between $x$ and $y$.

- Expand from an existing node instead of global samples from $\mathcal{X}$
- Samples rejected in 4. are never collision checked!
- Original formulation is bidirectional

Hsu et.al. 1997 [5]

**Algorithm** *expansion*

1. Pick a node $x$ from $V$ with probability $1/w(x)$.
2. Sample $K$ points from $N_d(x) = \{q \in \mathcal{C} \mid dist_c(q,x) < d\}$, where $dist_c$ is some distance metric of $\mathcal{C}$. ($K$ and $d$ are parameters.)
3. **for** each configuration $y$ that has been picked **do**
4. calculate $w(y)$ and retain $y$ with probability $1/w(y)$.
5. **if** $y$ is retained, $clearance(y) > 0$ and $link(x,y)$ returns YES
6. **then** put $y$ in $V$ and place an edge between $x$ and $y$.

- Expand from an existing node instead of global samples from $\mathcal{X}$
- Samples rejected in 4. are never collision checked!
- Original formulation is bidirectional

Hsu et.al. 1997 [5]

# Expansive Space Trees - Algorithm

**Algorithm** *expansion*

1. Pick a node $x$ from $V$ with probability $1/w(x)$.
2. Sample $K$ points from $N_d(x) = \{q \in \mathcal{C} \mid dist_c(q, x) < d\}$, where $dist_c$ is some distance metric of $\mathcal{C}$. ($K$ and $d$ are parameters.)
3. **for** each configuration $y$ that has been picked **do**
4. calculate $w(y)$ and retain $y$ with probability $1/w(y)$.
5. **if** $y$ is retained, $clearance(y) > 0$ and $link(x, y)$ returns YES
6. **then** put $y$ in $V$ and place an edge between $x$ and $y$.

- Expand from an existing node instead of global samples from $\mathcal{X}$
- Samples rejected in 4. are never collision checked!
- Original formulation is bidirectional

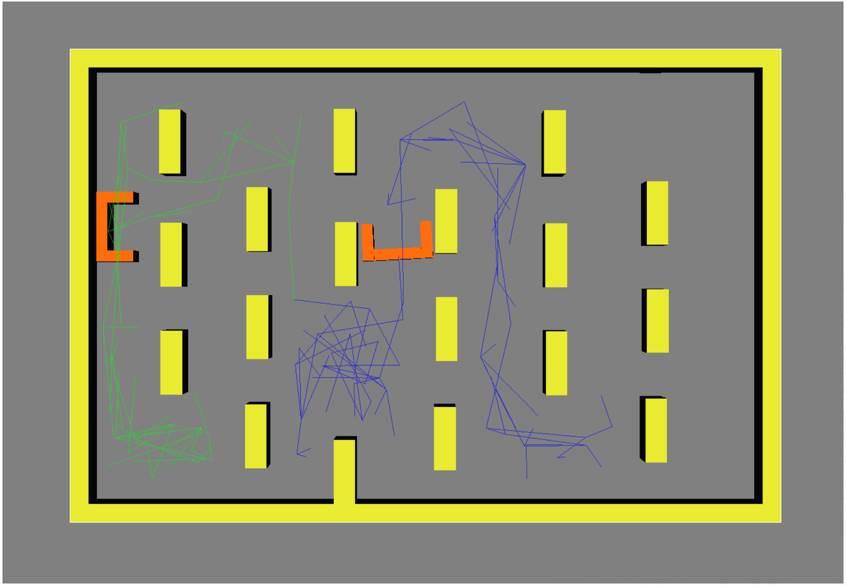
Hsu et.al. 1997 [5]

**Algorithm** *expansion*

1. Pick a node $x$ from $V$ with probability $1/w(x)$.
2. Sample $K$ points from $N_d(x) = \{q \in \mathcal{C} \mid dist_c(q, x) < d\}$, where $dist_c$ is some distance metric of $\mathcal{C}$. ($K$ and $d$ are parameters.)
3. **for** each configuration $y$ that has been picked **do**
4. calculate $w(y)$ and retain $y$ with probability $1/w(y)$.
5. **if** $y$ is retained, $clearance(y) > 0$ and $link(x, y)$ returns YES
6. **then** put $y$ in $V$ and place an edge between $x$ and $y$.

- Expand from an existing node instead of global samples from $\mathcal{X}$
- Samples rejected in 4. are never collision checked!
- Original formulation is bidirectional

Hsu et.al. 1997 [5]



(Bi-directional) EST for an example

# Path Postprocessing

The resulting paths are not smooth and often contain unnecessary motions.

Traditional post-processing includes:

- Path Shortcutting
  - Repeatedly pick two non-consecutive waypoints and attempt to connect them
- Perturbation of individual waypoints
  - Optional
  - Can reduce solution costs
  - Computationally expensive
  - For differentiable costs: exploit gradient
- Fit smooth splines through waypoints

**All modifications need to be collision checked.**

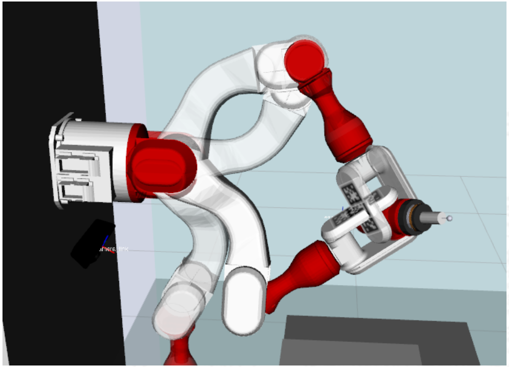
# Multiple IK Solutions



Redundant robots generate multiple joint solutions per pose.

Each Cartesian goal region adds a number of disjoint C-space goal regions.

Most tree-based planners naturally extend to **Multi-Goal Planning**, implicitly building multiple goal trees.



Multiple IK solutions for one target pose © Hendrich

# Optimal Planning

## Definition

An **Optimal Path Planning Problem** is defined by a path planning problem $\mathcal{P} = \langle \mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal} \rangle$ and a cost function $c(\tau) : R \geq 0$. It requires to find a feasible path $\tau^*$ such that $\tau^* = \text{argmin}_{\tau}\{c(\tau) \mid \tau \text{ is feasible for } \mathcal{P}\ \}$

In practice:

- Two-step process:
  - Find feasible path(s)
  - Optimize path(s)
- Planners are **asymptotically** optimal
  - Convergence might take long
  - Non-trivial to detect $\varepsilon$-optimal solution
- What cost function should be used?
  - C-space path length
  - Accumulated clearance (distance to obstacles)
  - Cartesian end-effector path length
  - Physical work

**Definition**

An **Optimal Path Planning Problem** is defined by a path planning problem $\mathcal{P} = \langle \mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal} \rangle$ and a cost function $c(\tau) : R \geq 0$. It requires to find a feasible path $\tau^*$ such that $\tau^* = \text{argmin}_\tau \{c(\tau) \mid \tau \text{ is feasible for } \mathcal{P}\ \}$

In practice:

- Two-step process:
  - Find feasible path(s)
  - Optimize path(s)
- Planners are **asymptotically** optimal
  - Convergence might take long
  - Non-trivial to detect $\varepsilon$-optimal solution
- What cost function should be used?
  - C-space path length
  - Accumulated clearance (distance to obstacles)
  - Cartesian end-effector path length
  - Physical work

**Definition**

An **Optimal Path Planning Problem** is defined by a path planning problem $\mathcal{P} = \langle \mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal} \rangle$ and a cost function $c(\tau) : R \geq 0$. It requires to find a feasible path $\tau^*$ such that $\tau^* = \text{argmin}_\tau\{c(\tau) \mid \tau \text{ is feasible for } \mathcal{P}\ \}$

In practice:

- Two-step process:
  - Find feasible path(s)
  - Optimize path(s)
- Planners are **asymptotically** optimal
  - Convergence might take long
  - Non-trivial to detect $\varepsilon$-optimal solution
- What cost function should be used?
  - C-space path length
  - Accumulated clearance (distance to obstacles)
  - Cartesian end-effector path length
  - Physical work

**Definition**

An **Optimal Path Planning Problem** is defined by a path planning problem $\mathcal{P} = \langle \mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal} \rangle$ and a cost function $c(\tau) : R \geq 0$. It requires to find a feasible path $\tau^*$ such that $\tau^* = \text{argmin}_\tau \{c(\tau) \mid \tau \text{ is feasible for } \mathcal{P}\}$

In practice:

- Two-step process:
  - Find feasible path(s)
  - Optimize path(s)
- Planners are **asymptotically** optimal
  - Convergence might take long
  - Non-trivial to detect $\varepsilon$-optimal solution
- What cost function should be used?
  - C-space path length
  - Accumulated clearance (distance to obstacles)
  - Cartesian end-effector path length
  - Physical work

# Planner*

**Method**
Instead of stopping at the first trajectory, continue sampling to improve solution.

Karaman and Frazzoli 2011 [2] introduced **PRM*** and **RRT***.
Both are efficient, asymptotically optimal versions of the basic algorithms.

# PRM*



**PRM is asymptotically optimal as-is.**

- Eventually all points on the optimal path will be added to the roadmap.

Ensure minimal required graph connectivity of $O(n \cdot \log(n))$.

- Reduce the neighborhood radius $r$ with sample size $n$:

$$r(n) = \gamma_{PRM} \cdot \left(\frac{log(n)}{n}\right)^{\frac{1}{d}}$$

where $\gamma_{PRM}$ depends on the planning space, $d$ is the dimensionality of $\mathcal{X}$

**Method**

Update tree whenever new samples yield cheaper paths to root.

- Instead of connecting the new states to *closest node*, connect to the *cheapest node* in neighborhood
- Change parent of neighboring states to new state if new path is cheaper

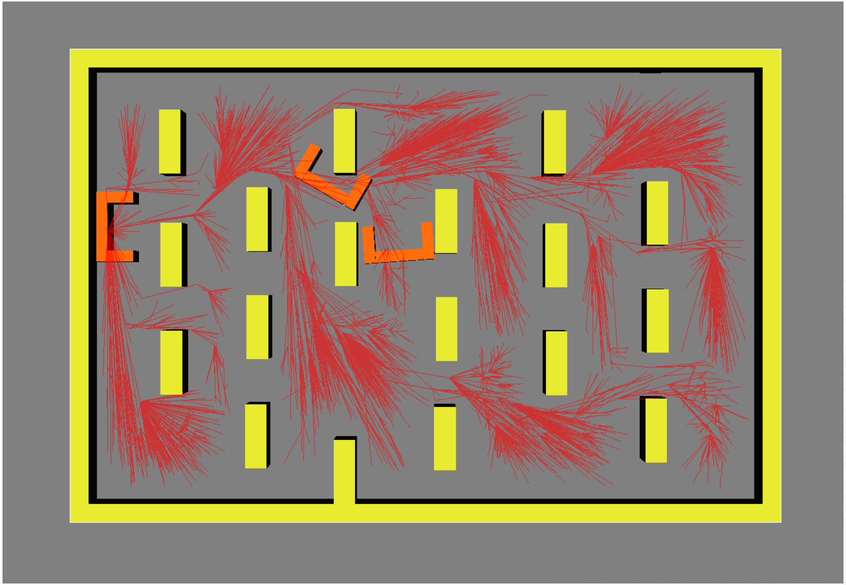
# RRT* - Algorithm



**Algorithm 6:** RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
**for** $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$ ;
  **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$ ;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      **if** $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      **if** $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      **then** $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
**return** $G = (V, E)$;

# RRT* - Example



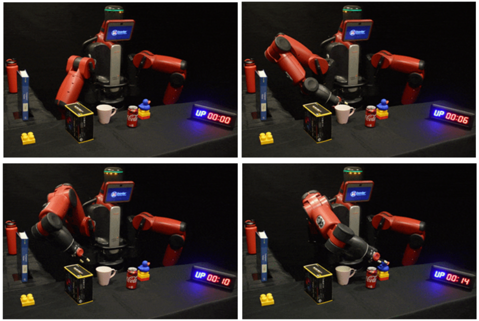
RRT* for an example

# Summary - Sampling Based Planning



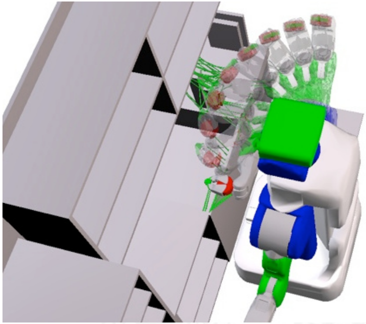
- Represent $\mathcal{X}_{free}$ probabilistically through samples
- Relies heavily on binary collision checking
- Post-processing solutions is essential
- Various (dozens) of algorithms with varying performance
- Straight-forward extensions for asymptotically optimal planning

## MPNet




Fast deep-learning system learning from planners [6]

## TrajOpt



Sequential convex optimizer solving trajectories [7]

# Outline

# Outline (cont.)

# Motivation

Robotic manipulation consists of more than waypoint-to-waypoint planning.

Even with a perfect path planner,

- **Where should you go?**
  - *Grasp Planning*
- **In what order should you go there?**
  - *Task Planning*
- Different planning steps usually operate in **different $\mathcal{X}$ or $\mathcal{X}_{free}$**
  - *Multi-Modal Planning*

The field is extremely spread out and only a few ideas are mentioned here.

# Motivation

Robotic manipulation consists of more than waypoint-to-waypoint planning.

Even with a perfect path planner,

- **Where should you go?**
  - *Grasp Planning*
- **In what order should you go there?**
  - *Task Planning*
- Different planning steps usually operate in **different** $\mathcal{X}$ **or** $\mathcal{X}_{free}$
  - *Multi-Modal Planning*

The field is extremely spread out and only a few ideas are mentioned here.

# Motivation

Robotic manipulation consists of more than waypoint-to-waypoint planning.

Even with a perfect path planner,

- **Where should you go?**
  - *Grasp Planning*
- **In what order should you go there?**
  - *Task Planning*
- Different planning steps usually operate in **different** $\mathcal{X}$ **or** $\mathcal{X}_{free}$
  - *Multi-Modal Planning*

The field is extremely spread out and only a few ideas are mentioned here.

# Motivation

Robotic manipulation consists of more than waypoint-to-waypoint planning.

Even with a perfect path planner,

- **Where should you go?**
  - *Grasp Planning*
- **In what order should you go there?**
  - *Task Planning*
- Different planning steps usually operate in **different** $\mathcal{X}$ **or** $\mathcal{X}_{free}$
  - *Multi-Modal Planning*

The field is extremely spread out and only a few ideas are mentioned here.

# Motivation

Robotic manipulation consists of more than waypoint-to-waypoint planning.

Even with a perfect path planner,

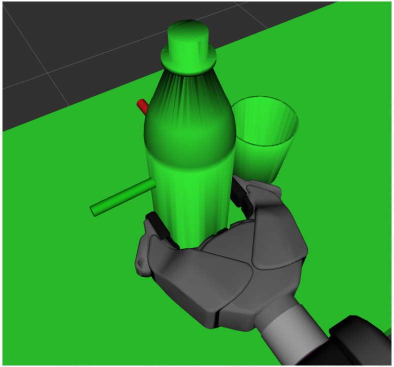
- **Where should you go?**
  - *Grasp Planning*
- **In what order should you go there?**
  - *Task Planning*
- Different planning steps usually operate in **different** $\mathcal{X}$ **or** $\mathcal{X}_{free}$
  - *Multi-Modal Planning*

The field is extremely spread out and only a few ideas are mentioned here.

# Identify Grasps

If you know where your object is, you can annotate fixed grasps.

To pick up the object, move to Cartesian pose relative to object.



Single Grasp for a bottle mesh

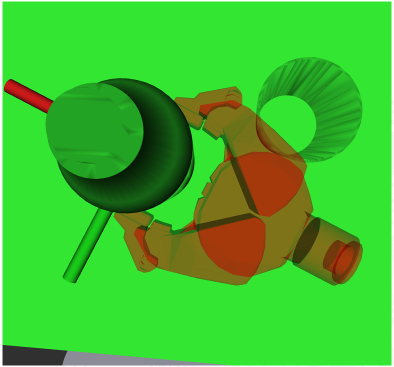
# Identify Grasps

If you know where your object is,
you can annotate fixed grasps.

To pick up the object, move to
Cartesian pose relative to object.

**Shortcoming**

Pose must be reachable and collision-free.



Candidate grasp in collision

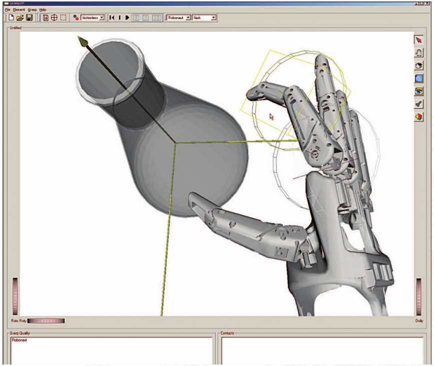
# Identify Grasps - By Simulation

For complex manipulators, the grasp has many parameters.

**Approach**
Simulate force interaction to generate reachable, stable grasps.

**Shortcomings**

- Computationally expensive
- Grasps without natural interpretation/use intention



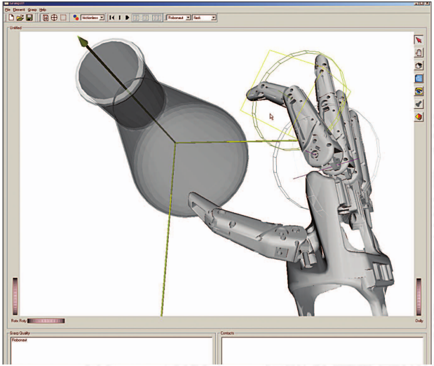
GraspIt: Grasp stability simulator [8]

For complex manipulators, the grasp has many parameters.

**Approach**
Simulate force interaction to generate reachable, stable grasps.

**Shortcomings**

- Computationally expensive
- Grasps without natural interpretation/use intention



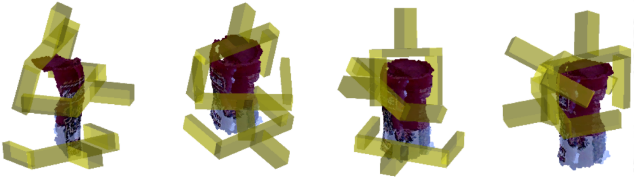
GraspIt: Grasp stability simulator [8]

For unknown or unmodelled objects, neither method is usable.

**Approach**

Learn to estimate good grasps from vision.

- Predict success rate for candidate grasps
- Or directly predict grasp parameters
- Often restricted to $< 6$ degrees of freedom (2 or 3)



Grasp candidates for a two-finger parallel gripper grasping a can

ten Pas et.al. 2017 [9]

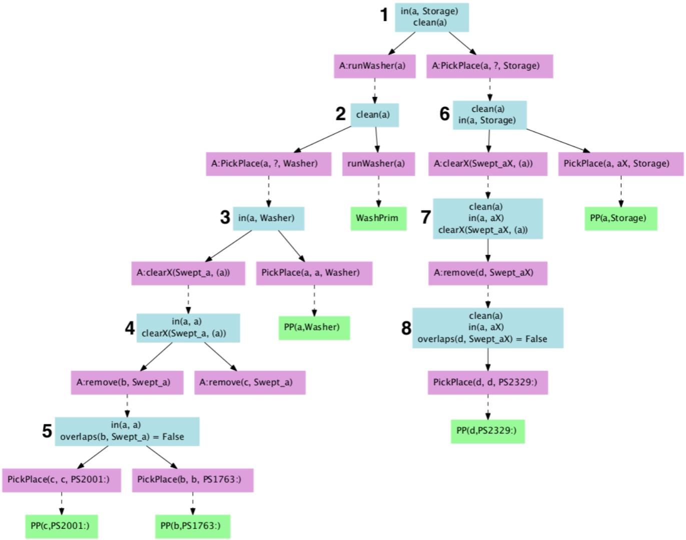
# Task Planning



**Definition**
Task Planning refers to the process of finding a feasible sequence of actions and their parameters to achieve a specified goal.

Requires well-defined action descriptions and goal specifications, e.g. `pickup(a)`.

# Hierarchical Task Network



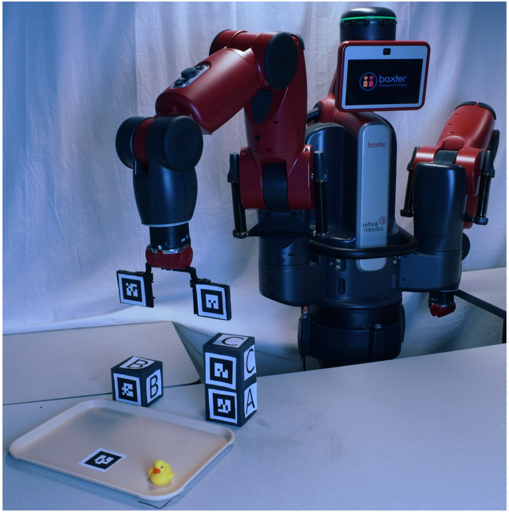

HTN plan for cleaning a through a washer and storing it away [10]

# Task and Motion Planning (TMP)

In robotics, task planning and motion planning are often entwined.

To pickup `A`, `C` has to be moved away.

Action preconditions include reachability constraints solved through Path Planning.




TMP framework implementing a traditional blocks-world task [11]

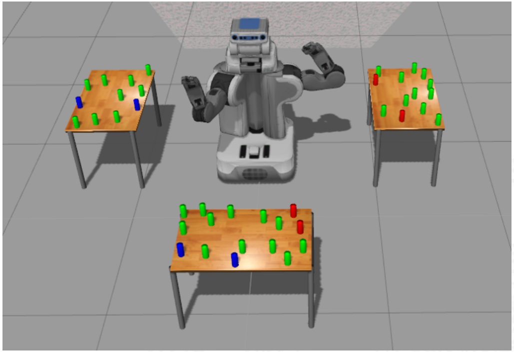
# Task and Motion Planning (TMP)

In robotics, task planning and motion planning are often entwined.

To pickup `A`, `C` has to be moved away.

Action preconditions include reachability constraints solved through Path Planning.

In practice these constraints are often implicit.



Sorting task: Move colored bottles to respective table [12]

# Motion Phases



Manipulation actions can be split up in motion phases with different concerns.

- Transit phase
  - Move towards object
- Approach phase
  - Move in contact with object
- Stabilization phase
  - Acquire sufficient grasp
- Lift phase
  - Retract grasped object from surface
- ...

# Motion Phases

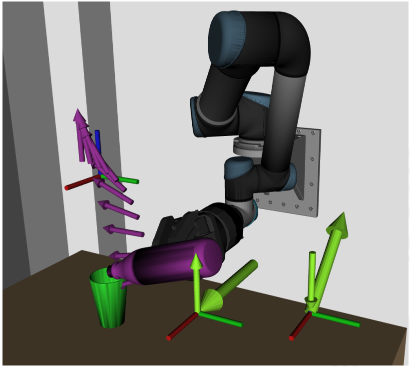


These different motions . . .

- Require different controllers
  - Position control, effort control, impedance control
- Have different motion characteristics
  - Restricted approach direction or variable free-space motion
- Have different validity concerns
  - Transit must not collide, approach will collide with object
- Actuate different joint sets
  - Gripper, arm, mobile base

## Approach

- Split up manipulation action along custom motion phases
- Allow custom path solvers for each phase
- Exchange interface states between the solvers



Combined manipulation plan to pick, pour from and place a bottle [13]

**Idea**

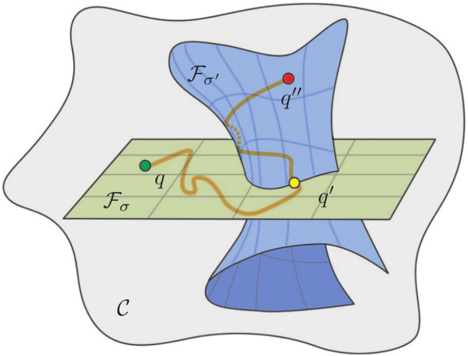
Manipulation plans can be interpreted as connected paths on multiple intersecting manifolds in $\mathcal{X}$.

Picking up an object might consist of

- Moving to a pose from which grasping is possible
- Moving grasped object to target location

**Approach**

Sample from each manifold *and each intersection* in turn.




Sketch of two intersecting planning manifolds [14]

# Bibliography

[1] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[2] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. DOI: 10.1177/0278364911406761.

[3] J. Kuffner and S. LaValle, “RRT-Connect: An Efficient Approach to Single-Query Path Planning.,”, vol. 2, Jan. 2000, pp. 995–1001. DOI: 10.1109/ROBOT.2000.844730.

[4] J. Starek, J. Gómez, E. Schmerling, L. Janson, L. Moreno, and M. Pavone, “An asymptotically-optimal sampling-based algorithm for bi-directional motion planning,” *Proceedings of the ... IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2015, Jul. 2015. DOI: 10.1109/IROS.2015.7353652.

[5] D. Hsu, J. .-. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *Proceedings of International Conference on Robotics and Automation*, vol. 3, 1997, 2719–2726 vol.3.

[6] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 2118–2124.

[7] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *in Proc. Robotics: Science and Systems*, 2013.

[8] A. T. Miller and P. K. Allen, "Graspit! a versatile simulator for robotic grasping," *IEEE Robotics Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004.

[9] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp pose detection in point clouds," *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1455–1473, 2017. DOI: 10.1177/0278364917735594.

# Bibliography (cont.)



[10] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1470–1477.

[11] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach.," in *Robotics: Science and Systems*, 2016, pp. 1–6.

[12] J. Ferrer-Mestres, G. Francès, and H. Geffner, "Combined task and motion planning as classical ai planning," *arXiv preprint arXiv:1706.06927*, 2017.

[13] M. Görner, R. Haschke, H. Ritter, and J. Zhang, "MoveIt! Task Constructor for Task-Level Motion Planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[14] K. Hauser and J.-C. Latombe, "Multi-modal motion planning in non-expansive spaces," *The International Journal of Robotics Research*, vol. 29, no. 7, pp. 897–915, 2010.