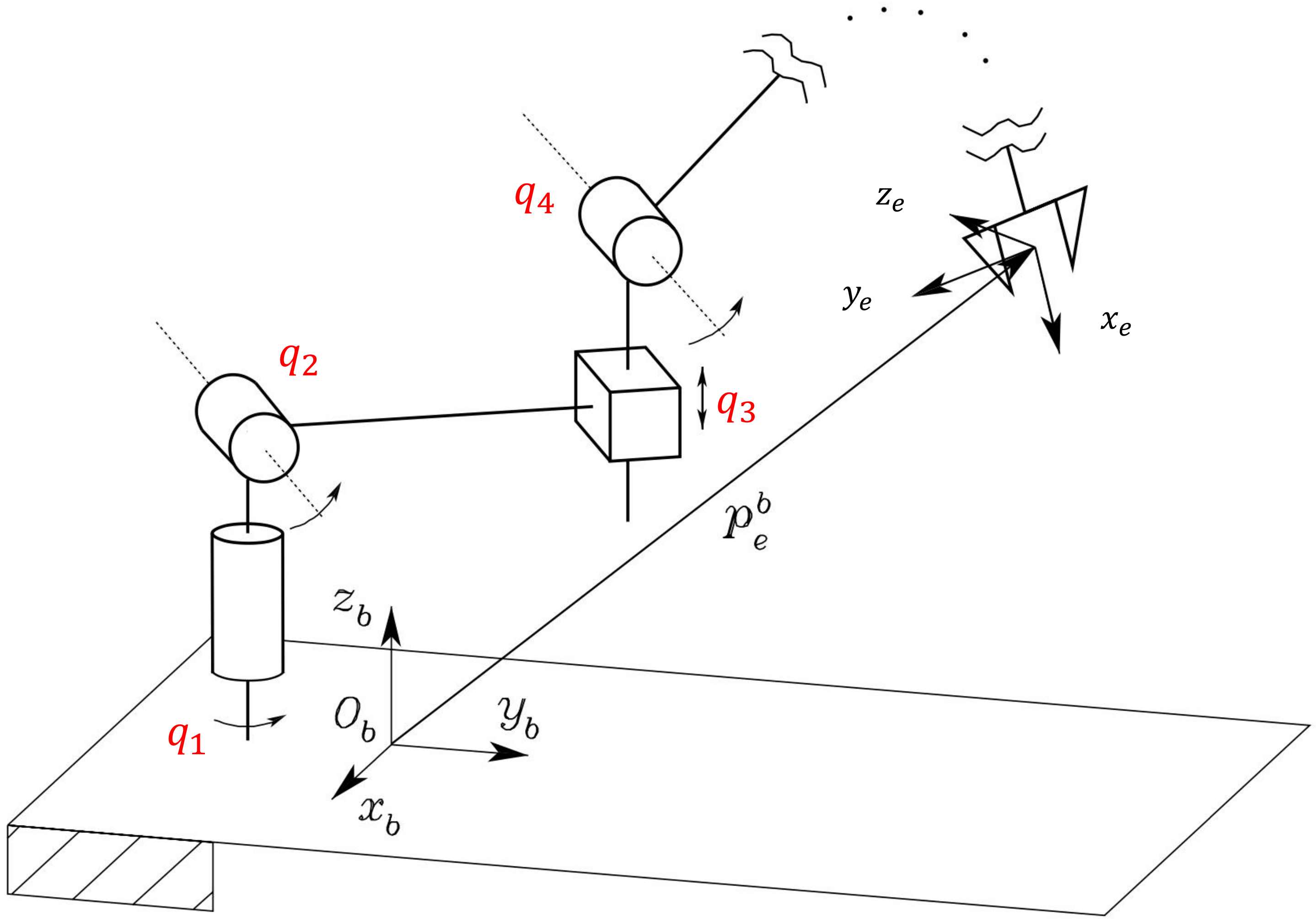
Robot Perception and Learning

Task planning and Motion planning

Tsung-Wei Ke

Fall 2025

Recap: Robotic Kinematics

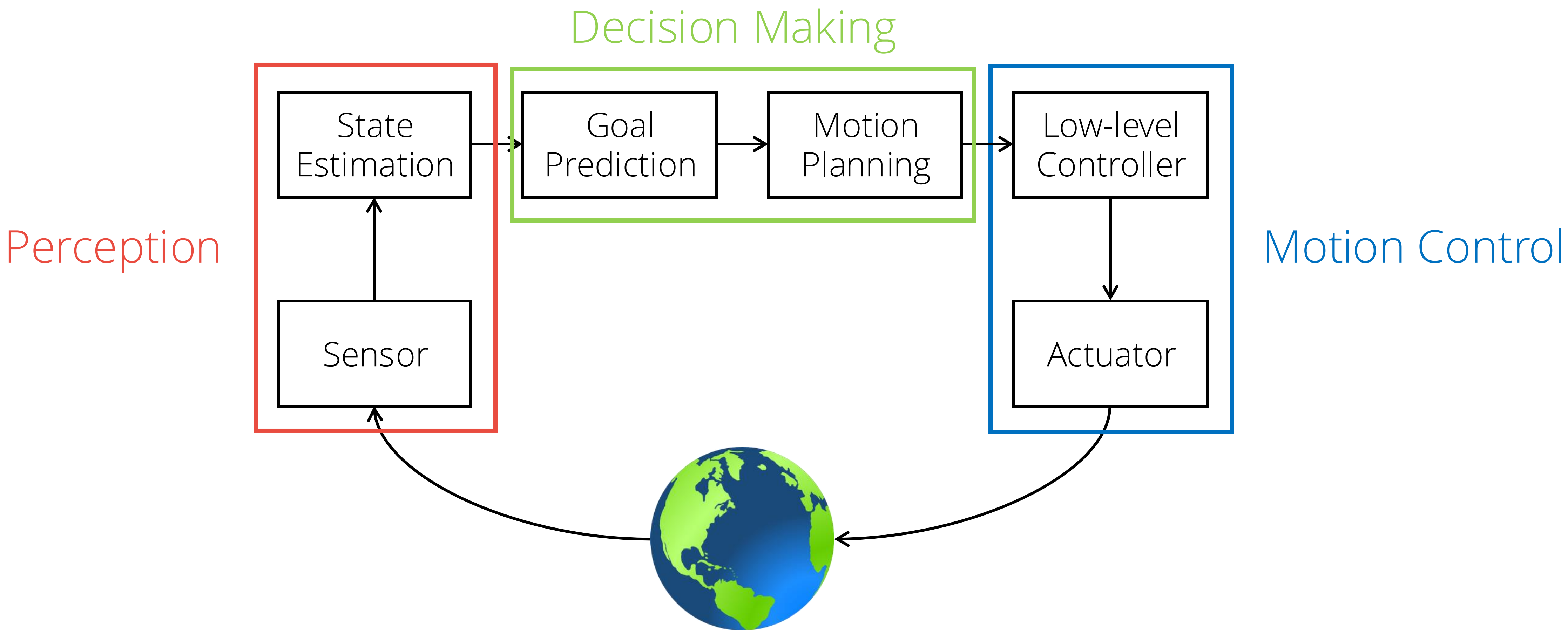


Robotics: Modeling, Planning and Control. Siciliano et al.

But We want a Robot that can See and Act

Video credit Pieter Abbeel

Action Requires Deciding a Goal, Planning to Achieve the Goal, and Controlling Motion to Follow the Plan



How to Define Goals?

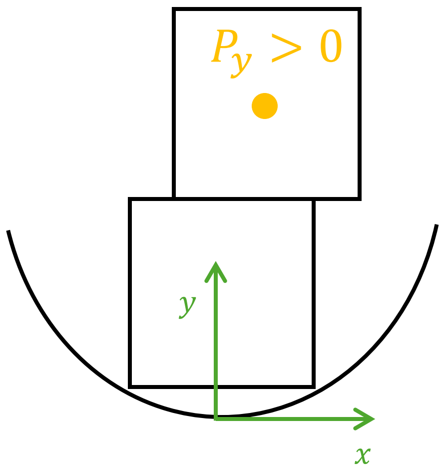
Goal → Motion Planning → Low-level Controller

Language instruction

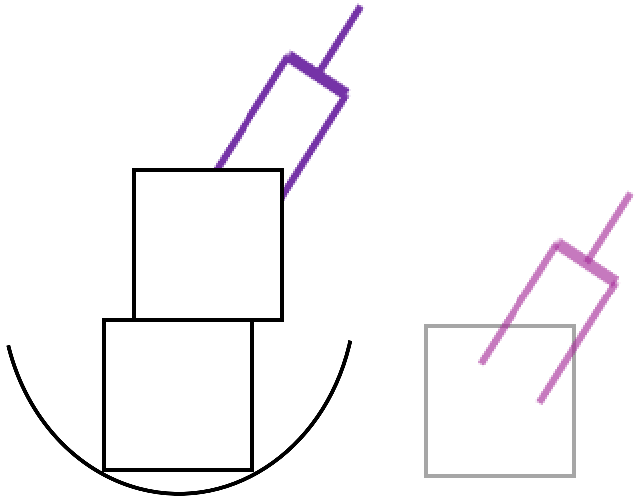
User

Stack the blocks on the empty bowl.

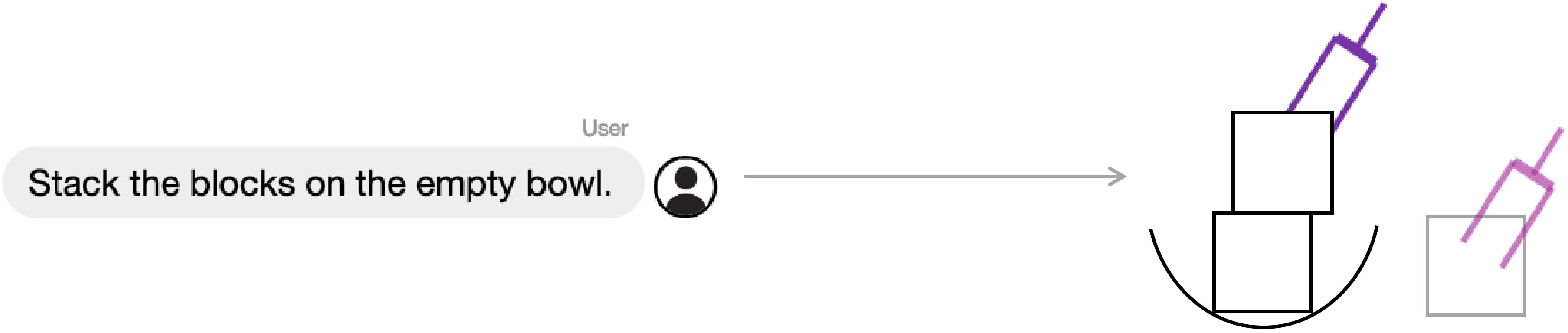
Object configuration



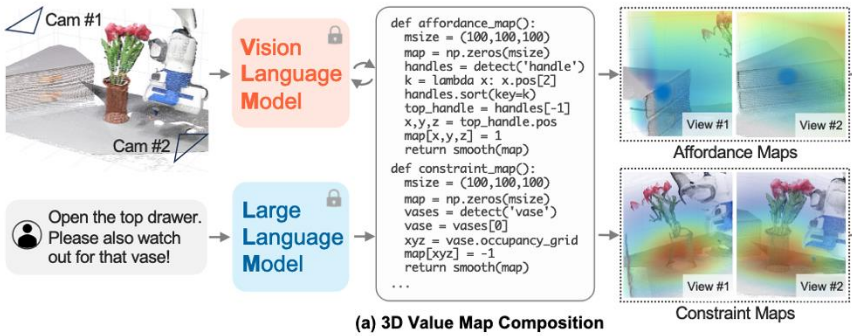
Robot pose



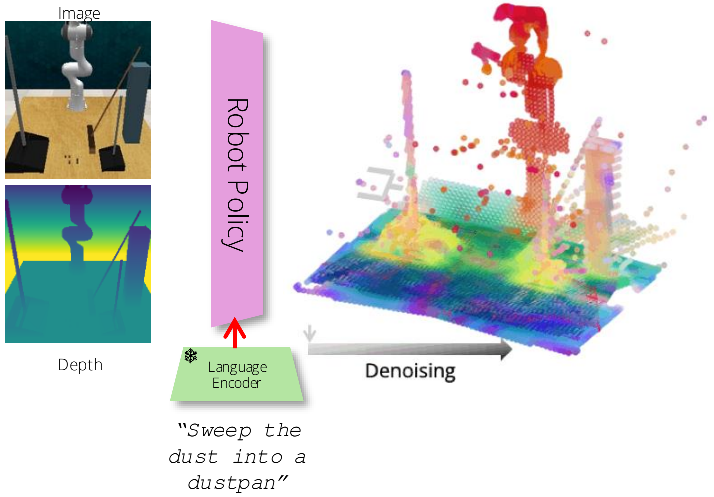
A Robot's End-Effector Poses are Common Representations of Goals



Neural Descriptor Fields: SE(3)-Equivariant Object Representations for Manipulation. Simeonov et al.



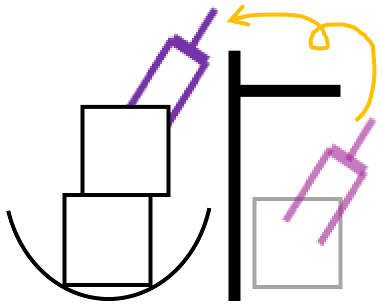
VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. Huang et al



3D Diffuser Actor: Policy Diffusion with 3D Scene Representations. Ke et al.

Motion Planning

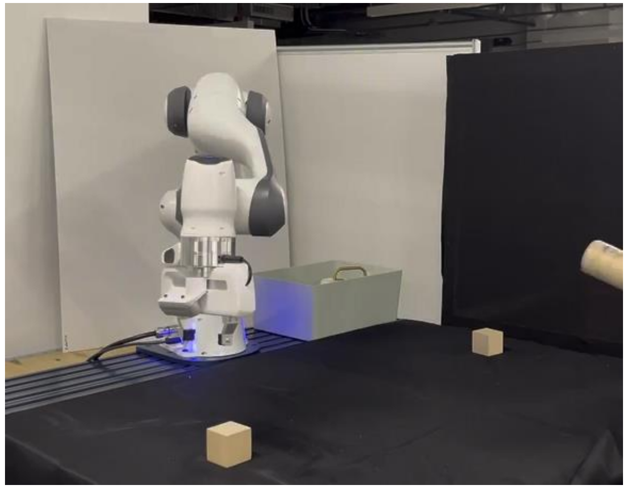
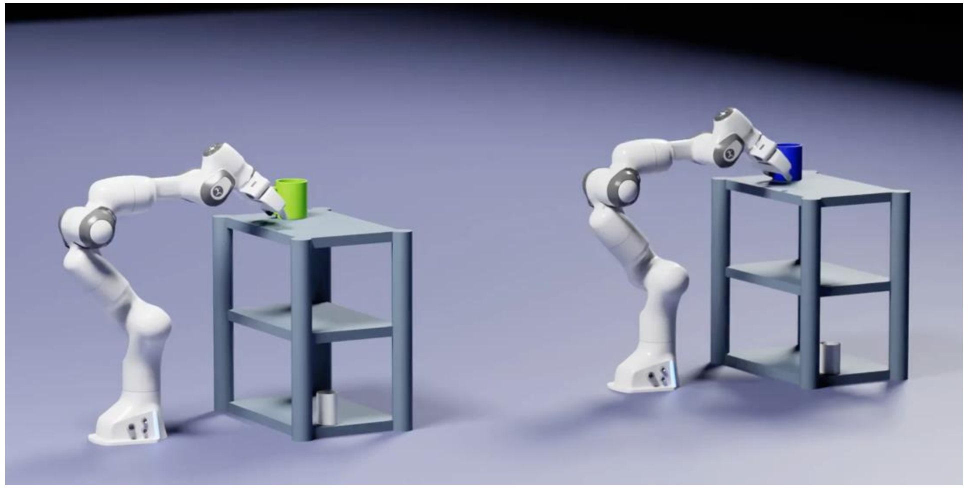
Goal → Motion Planning → Low-level Controller



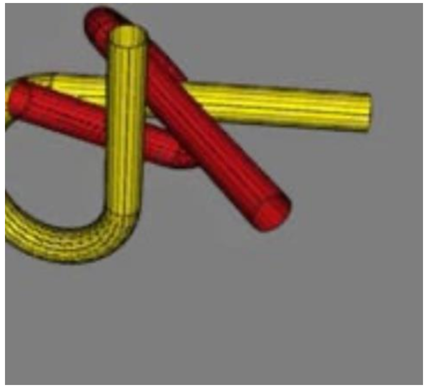
- Task: Find a **feasible** (and **optimal**) path/motion from the current configuration/pose of the robot to the goal configuration/pose
- Feasibility: The proposed plan should follow the given constraints
  - ➢ Environmental constraints (obstacles)
  - ➢ Efficiency constraints (dynamics/kinematics)
- Completeness: Report whether or not a feasible path exist in finite time
- Optimality: Return the best solution in finite time that minimizes the cost: time, energy, risk …

Motion Planning: Piano Mover's Problem

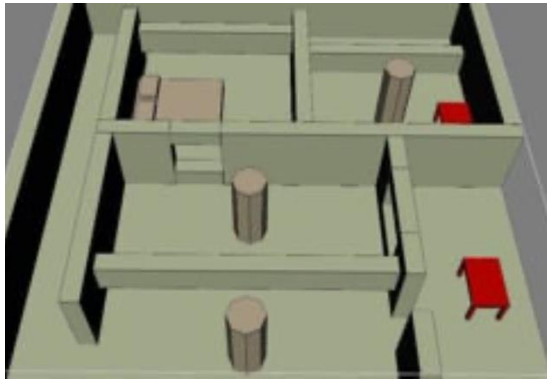
https://vimeo.com/58686590

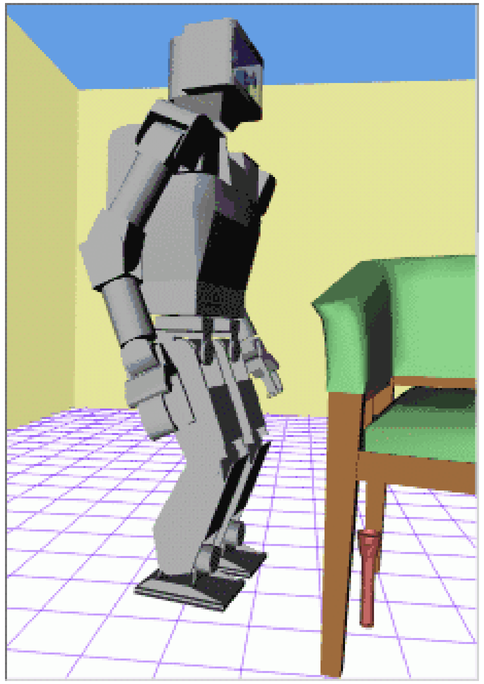
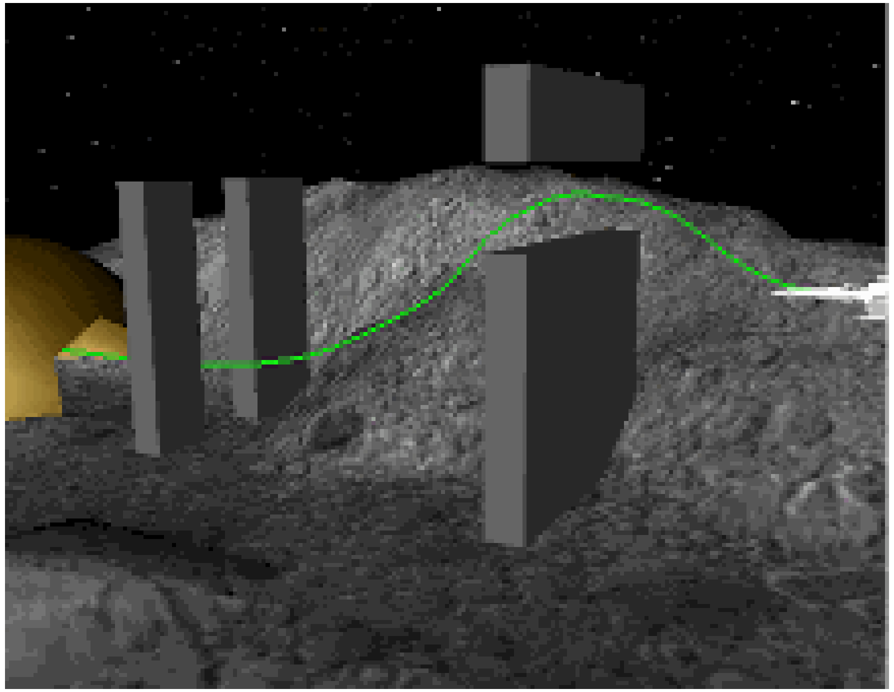
Video credit ccuRobo from Nvidia



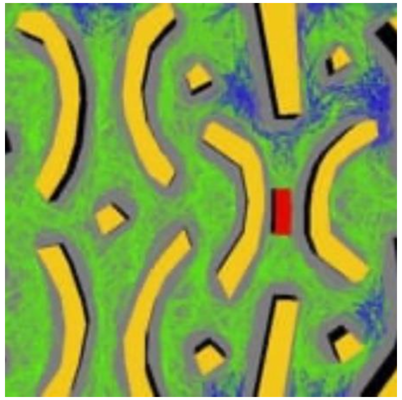
https://vimeo.com/58709484



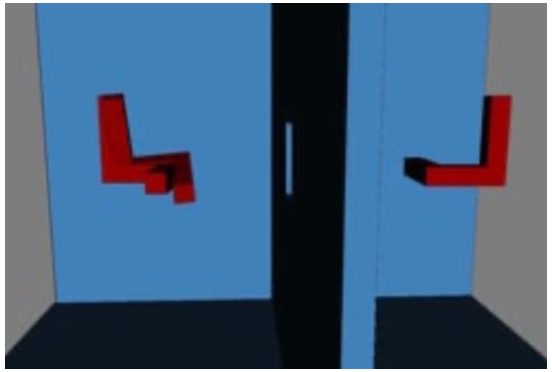
https://vimeo.com/58686593

Video credit J.J. Kuffner



https://vimeo.com/58686594



https://vimeo.com/58709589

Toy Example: 2D Path Planning with Point Robot

The robot is a point, that can only translate in 2D without rotation.
How can it reach for the red location from the green location?

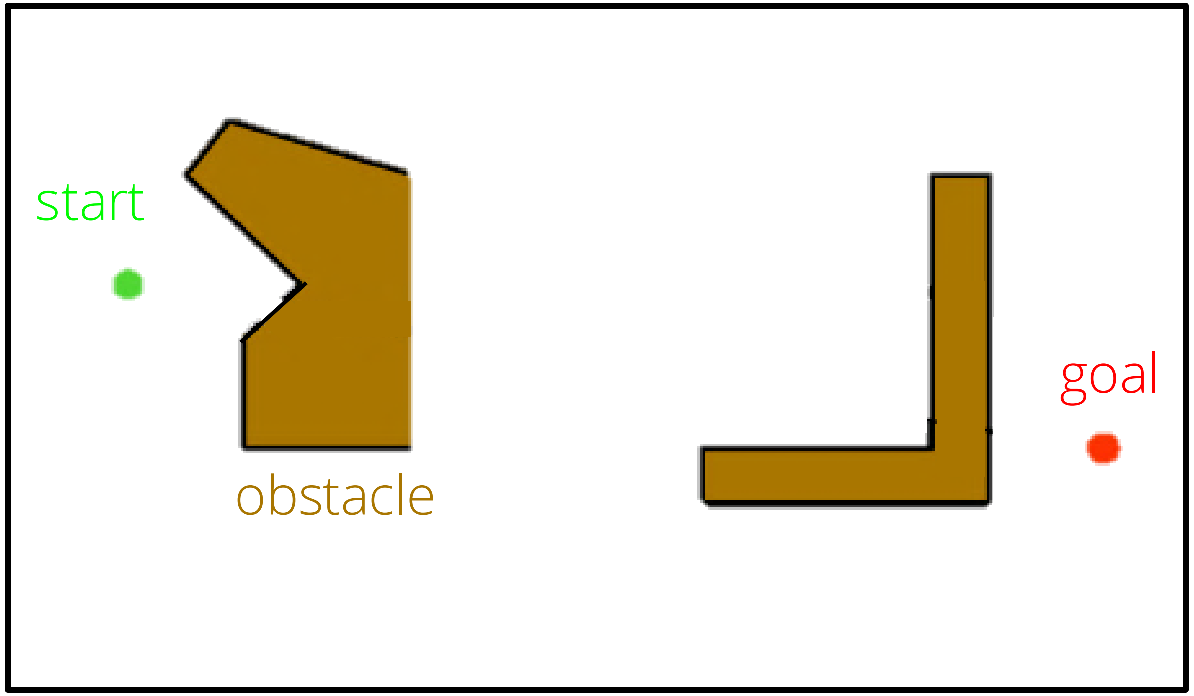


Image adapted from K. Hauser

Solution 1: Roadmap Planner Algorithm

- Discretize the 2D space into grids.
- Robot can only travel to neighboring grids
  - cost=1 to horizontal/vertical neighbors
  - cost=$\sqrt{2}$ to diagonal neighbors
  - cost=∞ to neighbors including obstacles

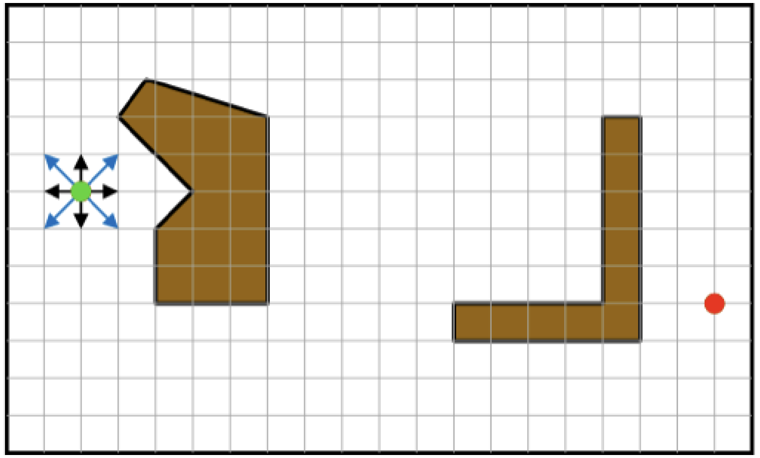
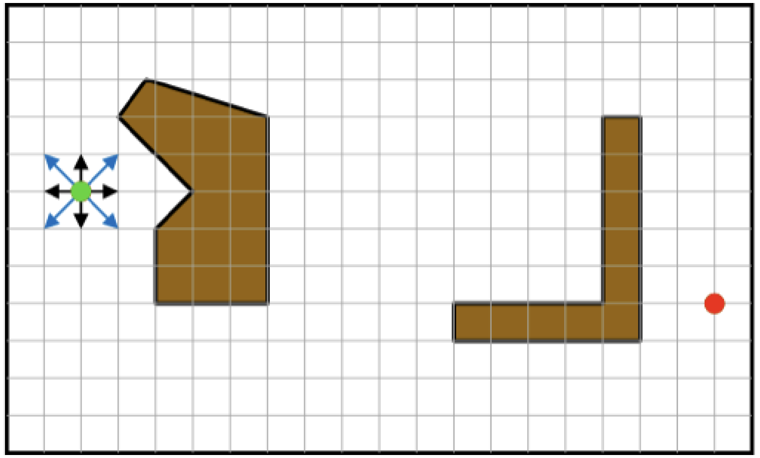


Image credit K. Hauser

Solution 1: Roadmap Planner Algorithm

- Discretize the 2D space into grids.
- Robot can only travel to neighboring grids
  - ➢ cost=1 to horizontal/vertical neighbors
  - ➢ cost=$\sqrt{2}$ to diagonal neighbors
  - ➢ cost=∞ to neighbors including obstacles
- Search the best path (e.g. Dijkstra's algorithm)



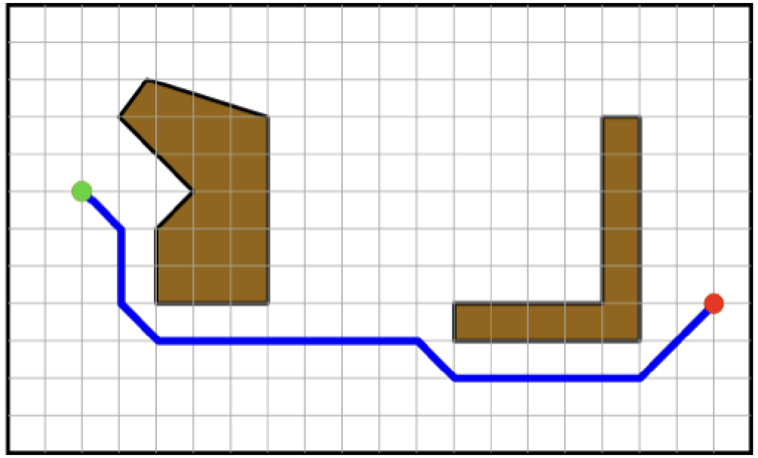


Image credit K. Hauser

Roadmap Planner Algorithm Has Resolution Problems

Coarse-resolution grids:

- Computationally efficient
- May not be optimal

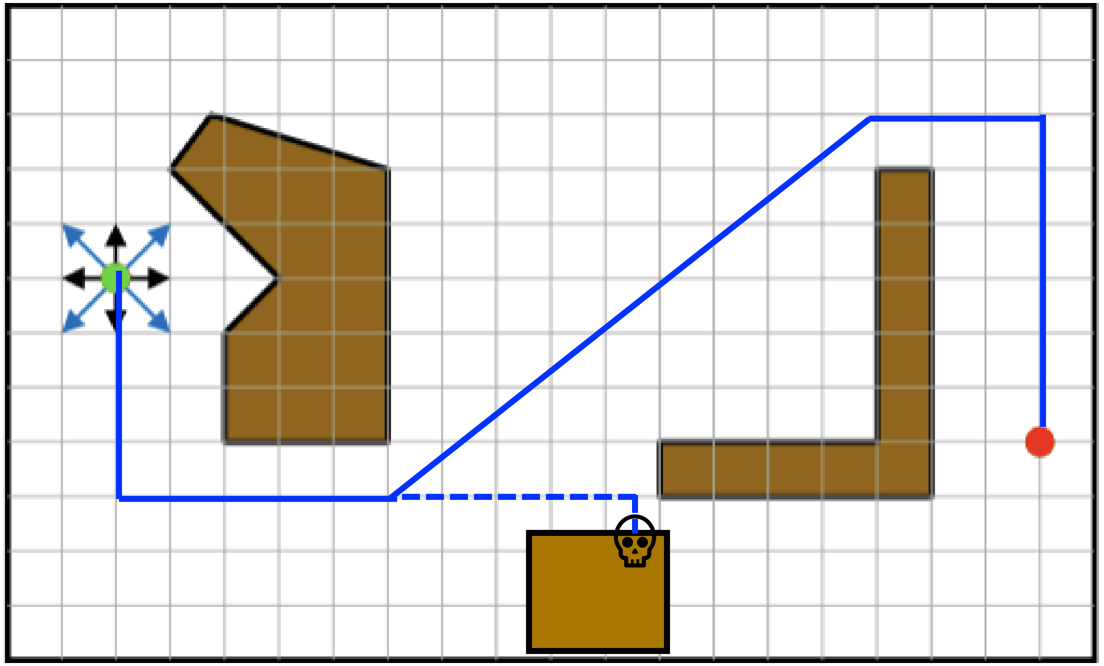


Image credit K. Hauser

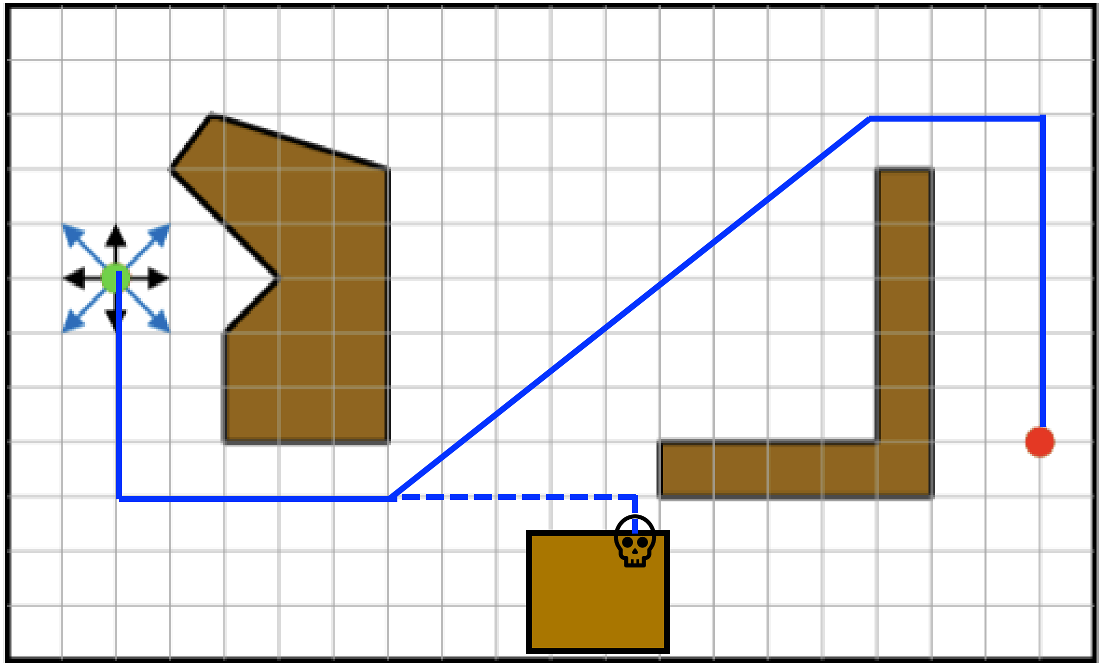
Roadmap Planner Algorithm Has Resolution Problems

Coarse-resolution grids:

- Computationally efficient
- May not be optimal

Fine-resolution grids:

- Computationally inefficient
- More likely to be optimal



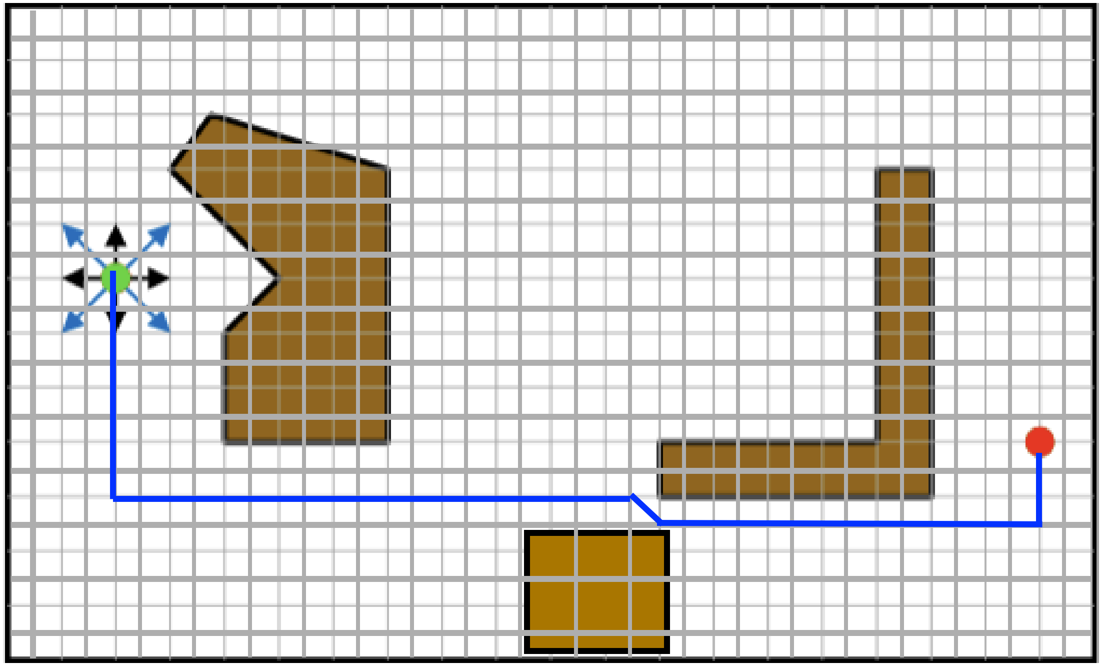
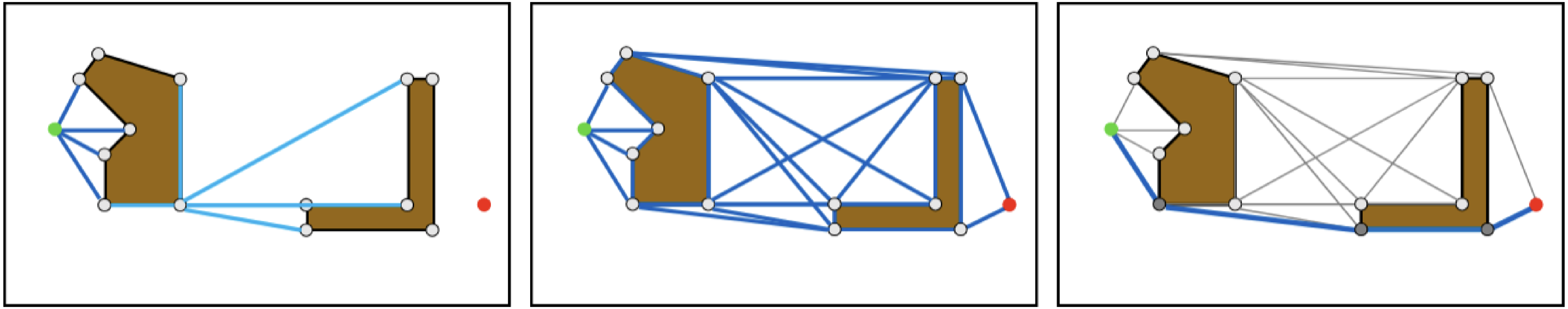


Image credit K. Hauser

Solution 2: Visibility Graph Algorithm

We don't need grids, but vertices of the start location, goal location and obstacle corners

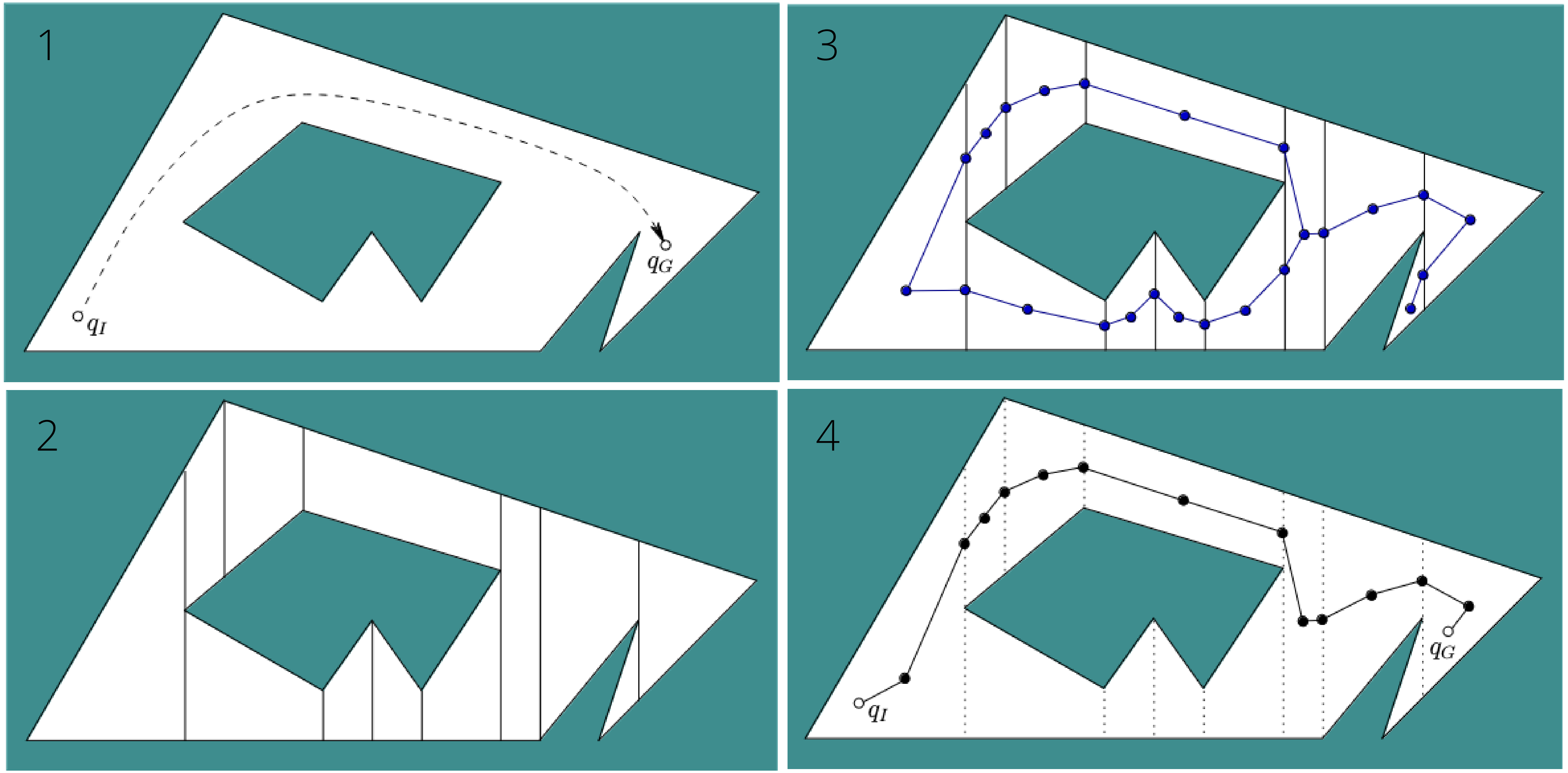


1. Let V be the union of the start, the goal and all obstacle vertices.
2. $E \leftarrow \{\}$
3. **for** all pairs of distinct vertices $u, v \in V$
4. **if** $\overline{uv}$ is an obstacle edge
5. Add (u, v) to E
6. **else if** $\overline{uv}$ is collision free
7. Add (u, v) to E
8. Search $G=(V,E)$, with Cartesian distance as the edge cost, to connect the start and goal
9. **return** the path if one is found.

Image credit K. Hauser

Solution 3: Cell Decomposition Algorithm

Connect neighboring free-space cells

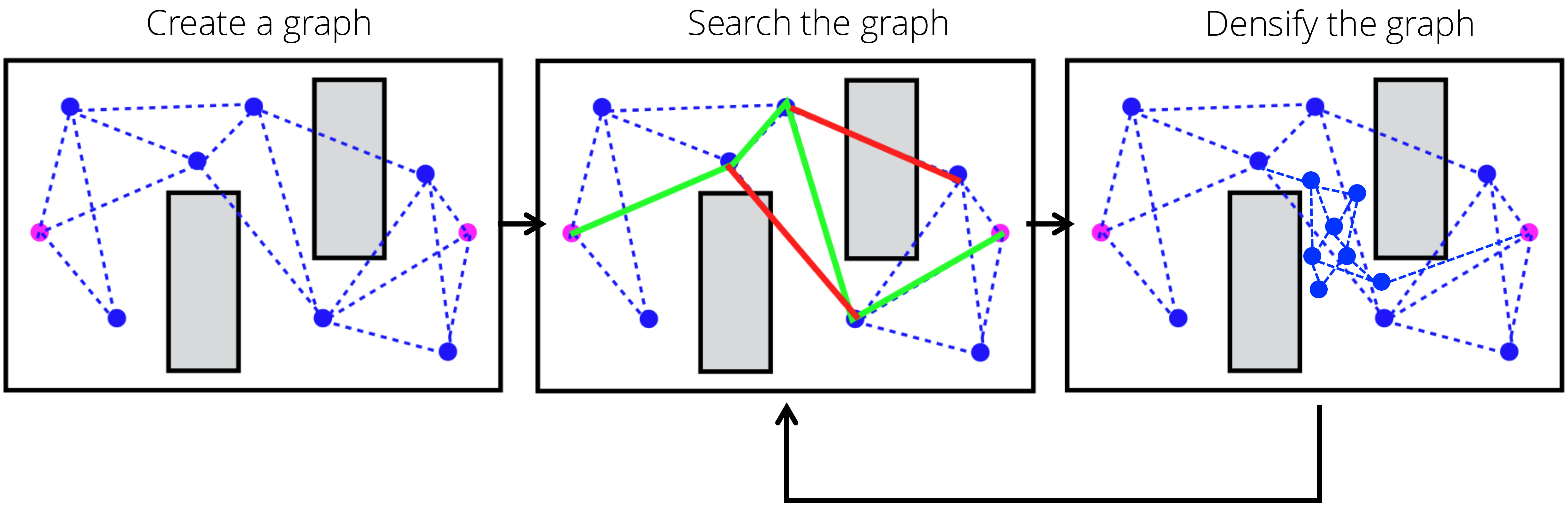


Segment the free space into cells by vertices

Search a path that traverses from the start cell to the goal cell

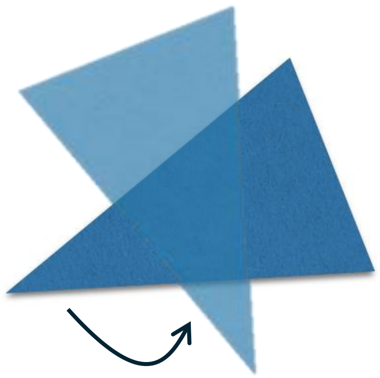
Image Credit S. M. LaValle

A General Framework for Motion/Path Planning



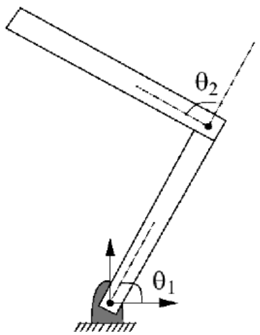
But Planning is Hard…

- Previously, we only consider 2D point robots. What if the robot shape is triangle that can both translate and rotate in 2D?
- What if the robot is a car which cannot make pure left/right movement?
- What if the robot is a 2-joint planar arm?



Triangle

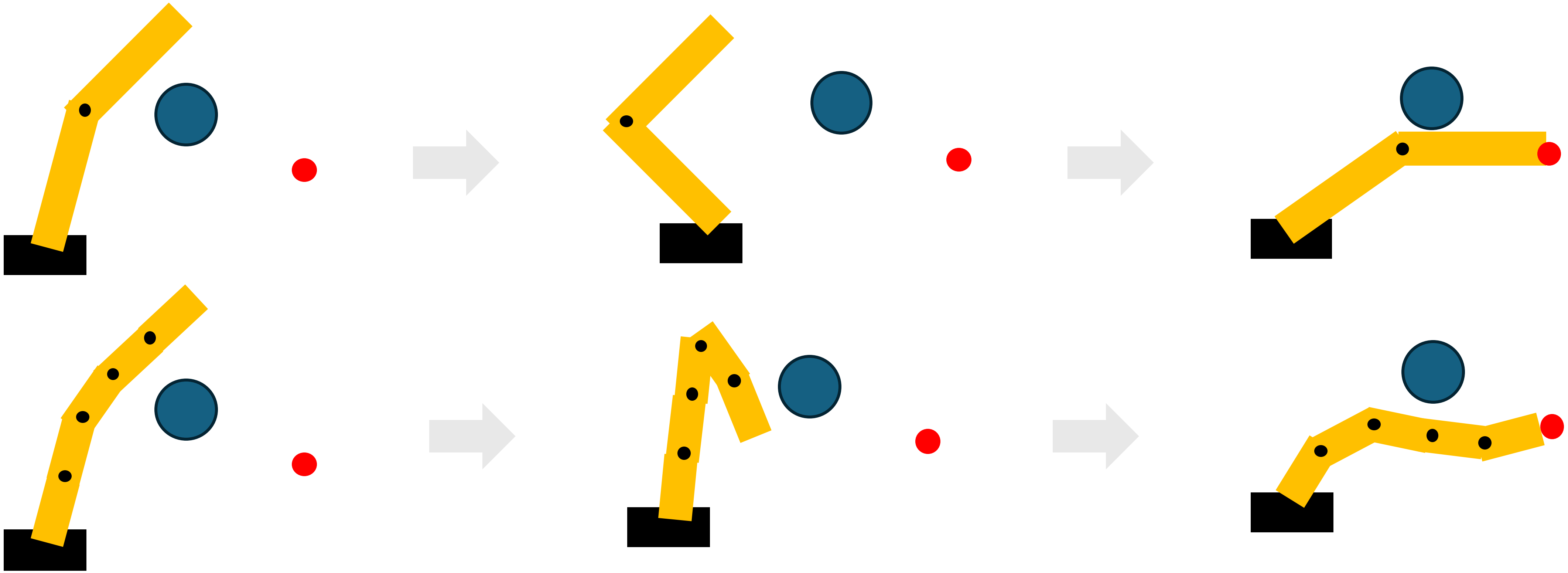
Racecar



2-joint planar arm

But Planning is Hard…

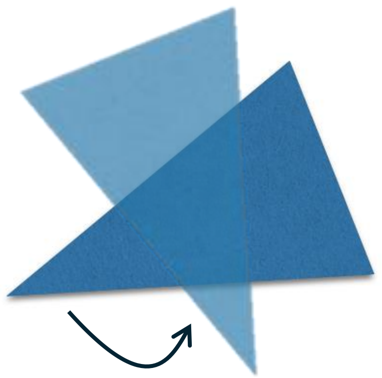
- Previously, we only consider 2D point robots. What if the robot shape is triangle that can both translate and rotate in 2D?
- What if the robot is a car which cannot make pure left/right movement?
- What if the robot is a 2-joint planar arm?



But Planning is Hard…

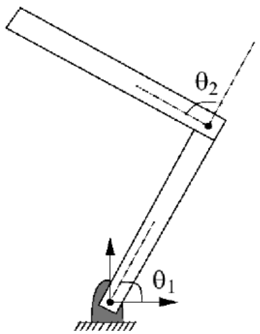
- Previously, we only consider 2D point robots. What if the robot shape is triangle that can both translate and rotate in 2D?
- What if the robot is a car which cannot make pure left/right movement?
- What if the robot is a 2-joint planar arm?
- What if the robot is a 7-joint planar arm?
- What if the robot works in 3D?

What is the unified planning formulation for robots with different mechanisms?



Triangle

Racecar



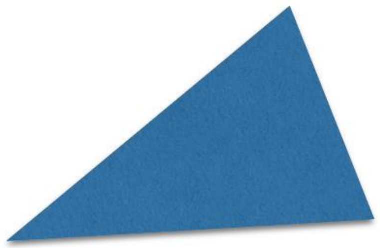
2-joint planar arm

Manipulator

The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

$\mathbb{R}^2$



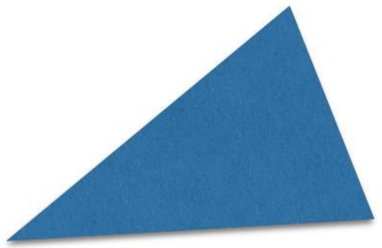
Translating Triangle

The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

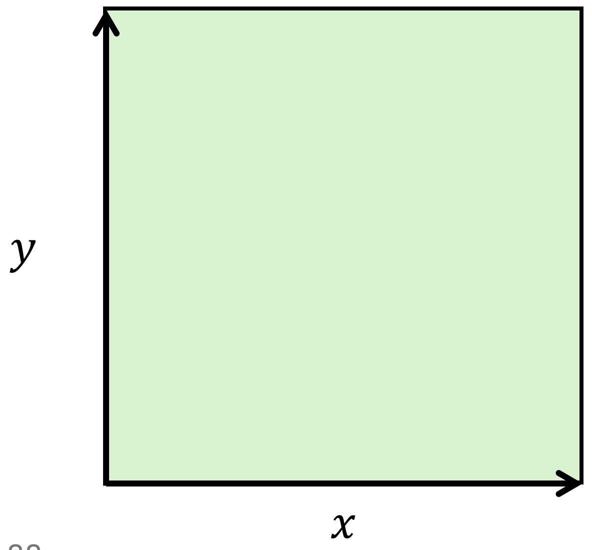
We can represent it with coordinates x, y

$\mathbb{R}^2$



Translating Triangle

C-space



The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

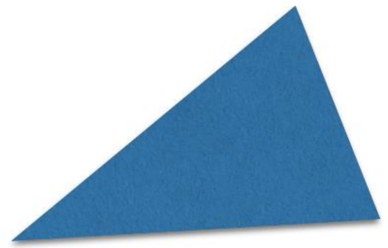
The configuration that the robot can reach

We can represent it with coordinates x, y

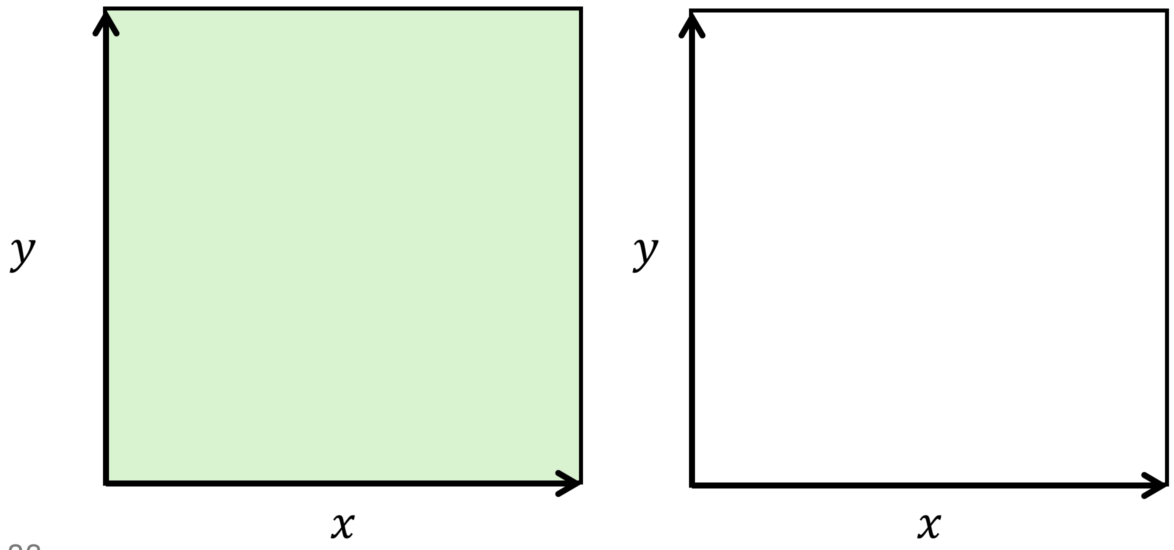
$\mathbb{R}^2$

C-space

workspace



Translating Triangle

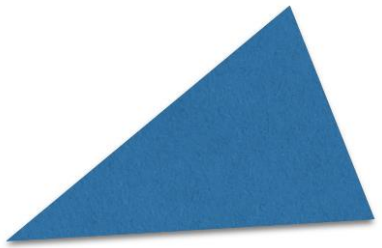


The Configuration Space

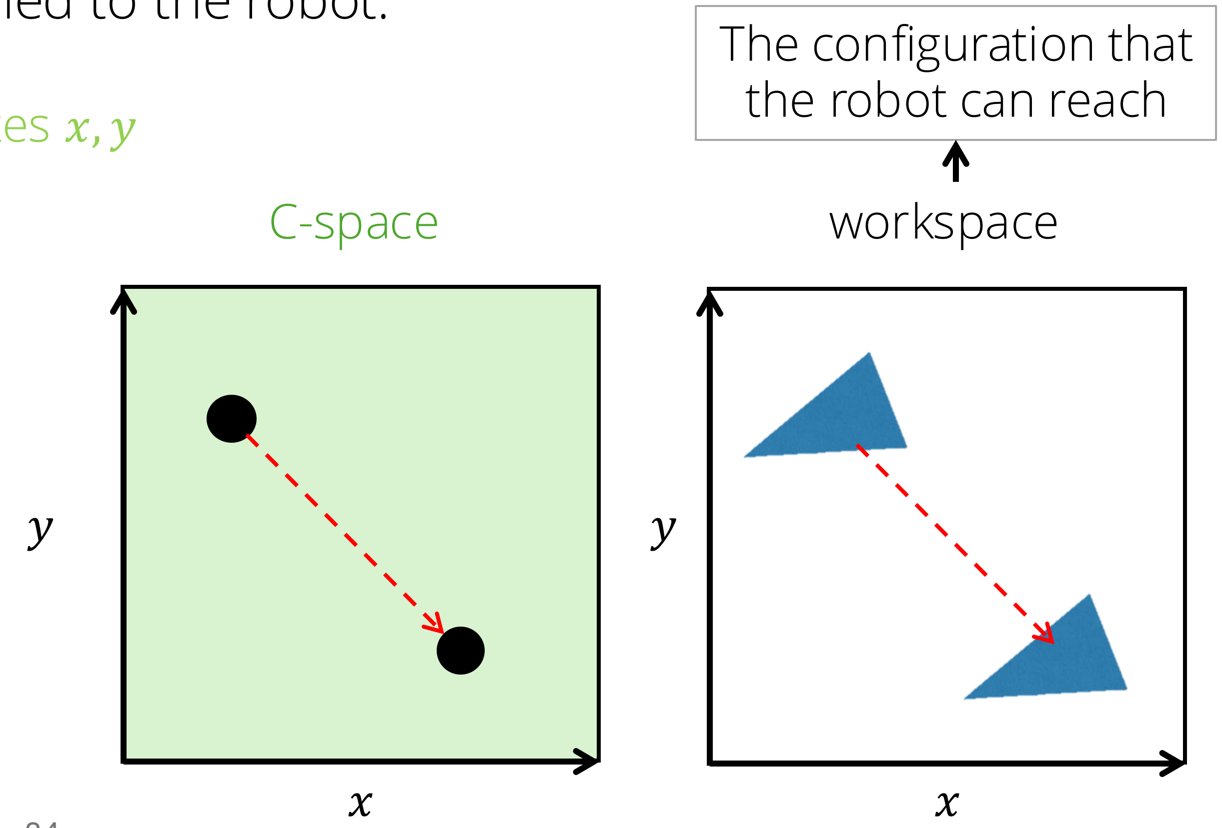
- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

We can represent it with coordinates x, y

$\mathbb{R}^2$



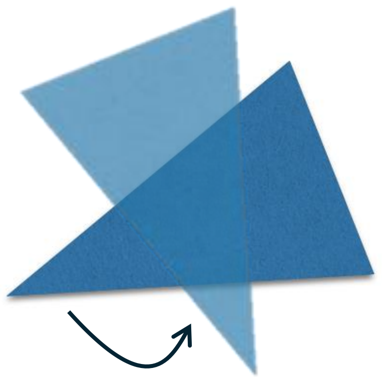
Translating Triangle



The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

$\mathbb{R}^2 \times \mathbb{S}^1$



Triangle

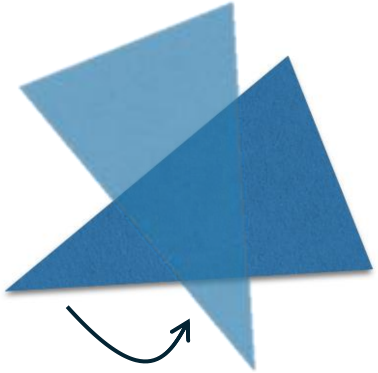
The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

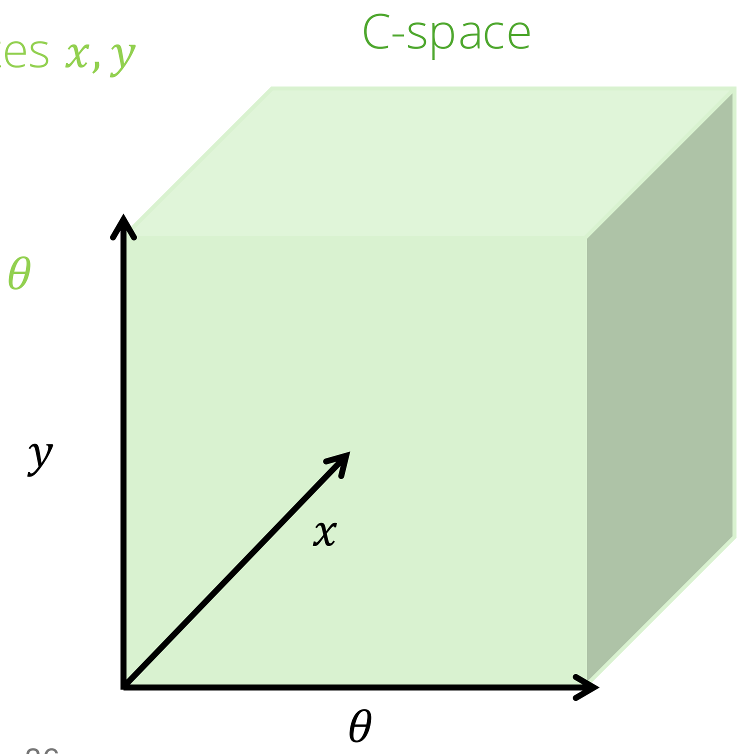
We can represent it with coordinates x, y

$$\mathbb{R}^2 \times \mathbb{S}^1$$

can represent it with angle θ



Triangle



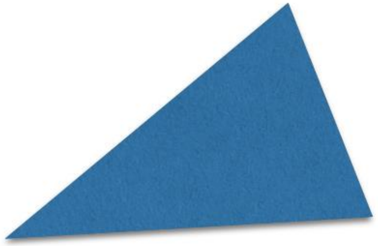
The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

We can represent it with coordinates x, y

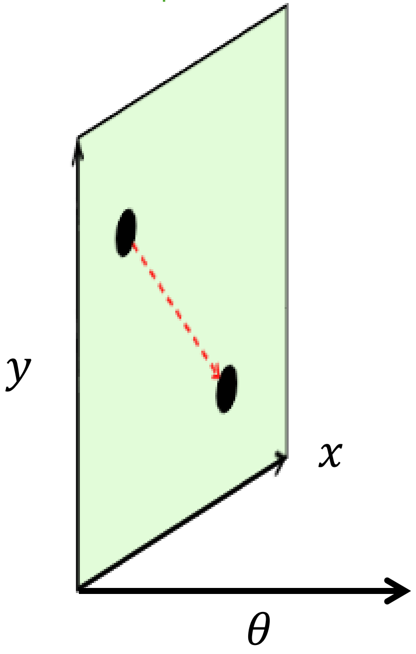
$$\mathbb{R}^2 \times \mathbb{S}^1$$

We can represent it with angle θ

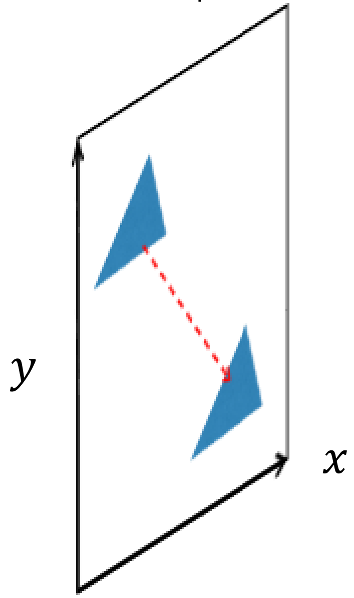


Triangle

C-space



workspace



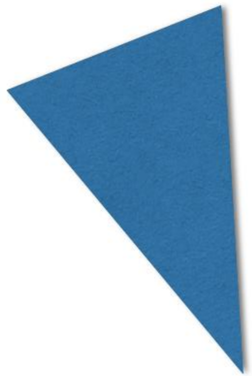
The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

We can represent it with coordinates x, y

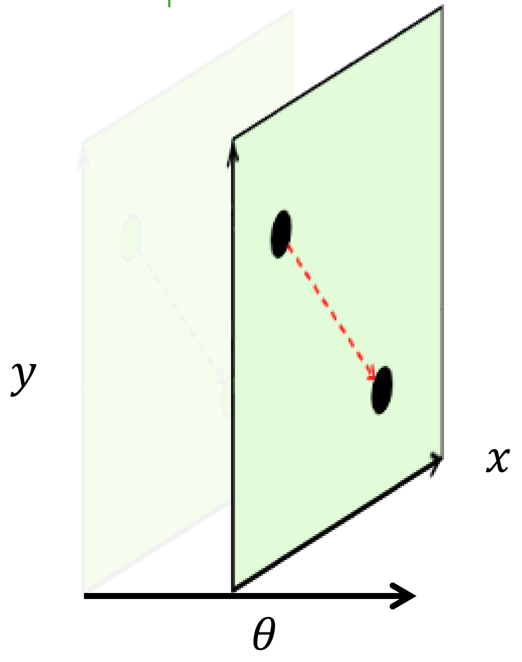
$$\mathbb{R}^2 \times \mathbb{S}^1$$

We can represent it with angle θ

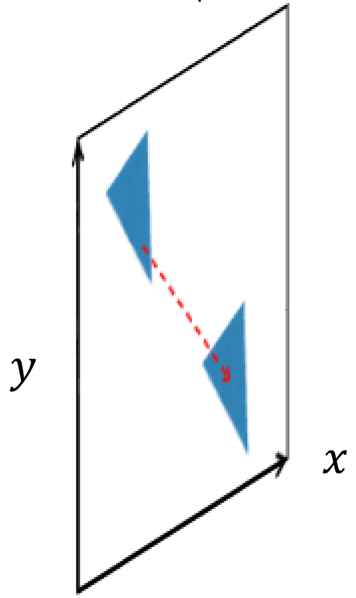


Triangle

C-space



workspace



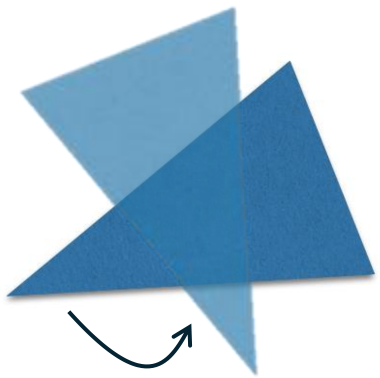
The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

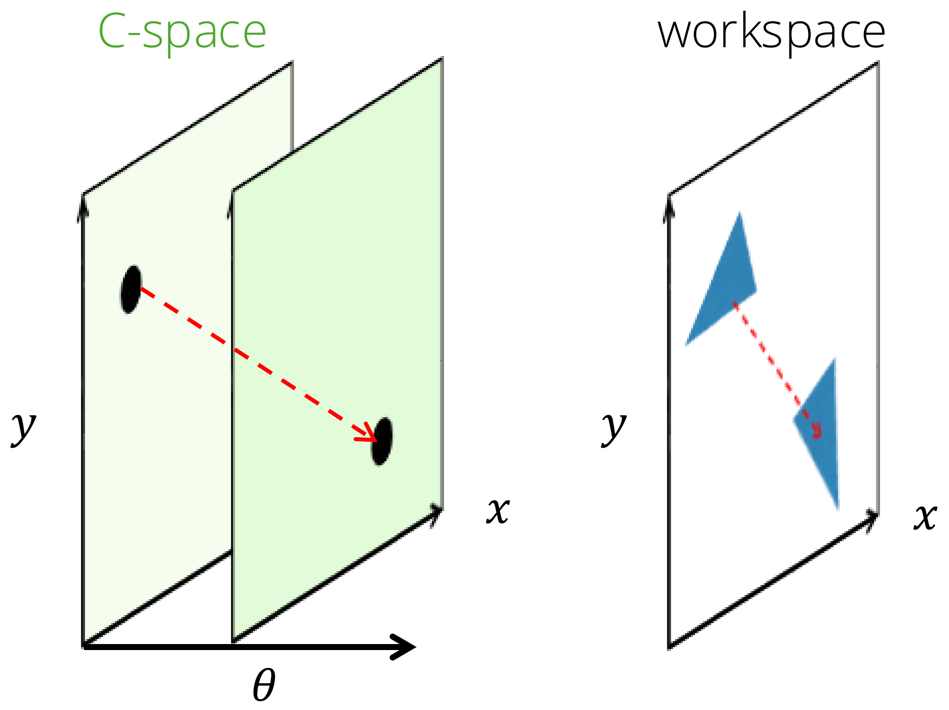
$\mathbb{R}^2 \times \mathbb{S}^1$

We can represent it with coordinates x, y

can represent it with angle θ



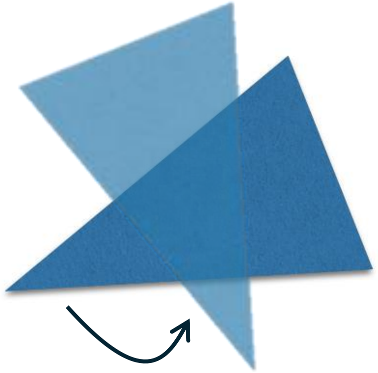
Triangle



The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

$\mathbb{R}^2 \times \mathbb{S}^1$



Triangle

We can represent it with coordinates x, y

$\mathbb{R}^2 \times \mathbb{S}^1$

We can represent it with angle θ

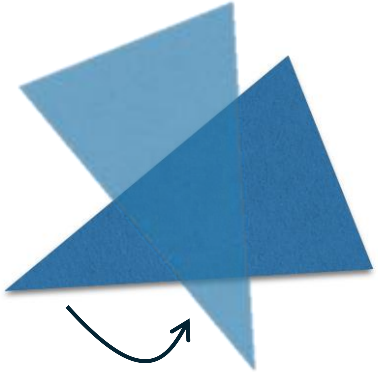
Racecar

The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

We can represent it with angle θ_1, θ_2

$\mathbb{R}^2 \times \mathbb{S}^1$

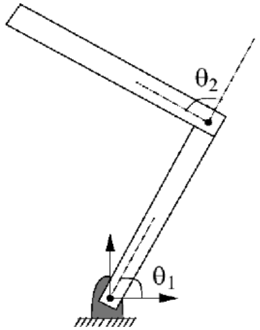


Triangle

$\mathbb{R}^2 \times \mathbb{S}^1$

Racecar

$\mathbb{S}^1 \times \mathbb{S}^1$



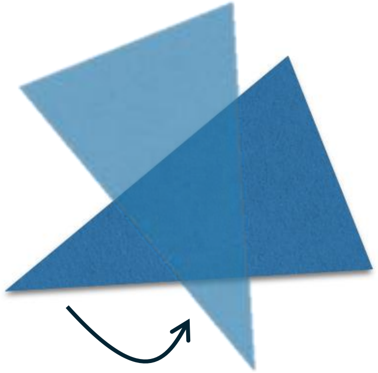
2-joint planar arm

The Configuration Space

- The configuration of of a robot is a complete specification of the position of every point of the robot
- Configuration space (C-space) is the n-dimensional space containing all possible configurations of the robot. In other words, C-space includes the set of all rigid-body transformations that can be applied to the robot.

We can represent it with angle $\theta_1, \theta_2, \ldots \theta_n$

$\mathbb{R}^2 \times \mathbb{S}^1$

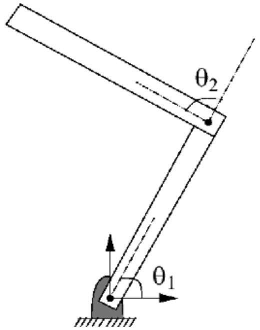


Triangle

$\mathbb{R}^2 \times \mathbb{S}^1$

Racecar

$\mathbb{S}^1 \times \mathbb{S}^1$



2-joint planar arm

$\mathbb{S}^1 \times \cdots \times \mathbb{S}^1$

Manipulator

How to Specify Obstacles in the Configuration Space?

Robot operates in a 2D / 3D workspace $\mathcal{W} = \mathbb{R}^2 \text{ or } \mathbb{R}^3$

Subset of this space is obstacles $\mathcal{O} \subset \mathcal{W}$

semi-algebraic models (polygons, polyhedra)

Geometric shape of the robot
(set of points occupied by robot at a config) $\mathcal{A}(q) \subset \mathcal{W}$

C-space obstacle region

$$\mathcal{C}_{obs} = \{\boldsymbol{q} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O} \neq \emptyset\}$$

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$$

Example: Rotational Motion

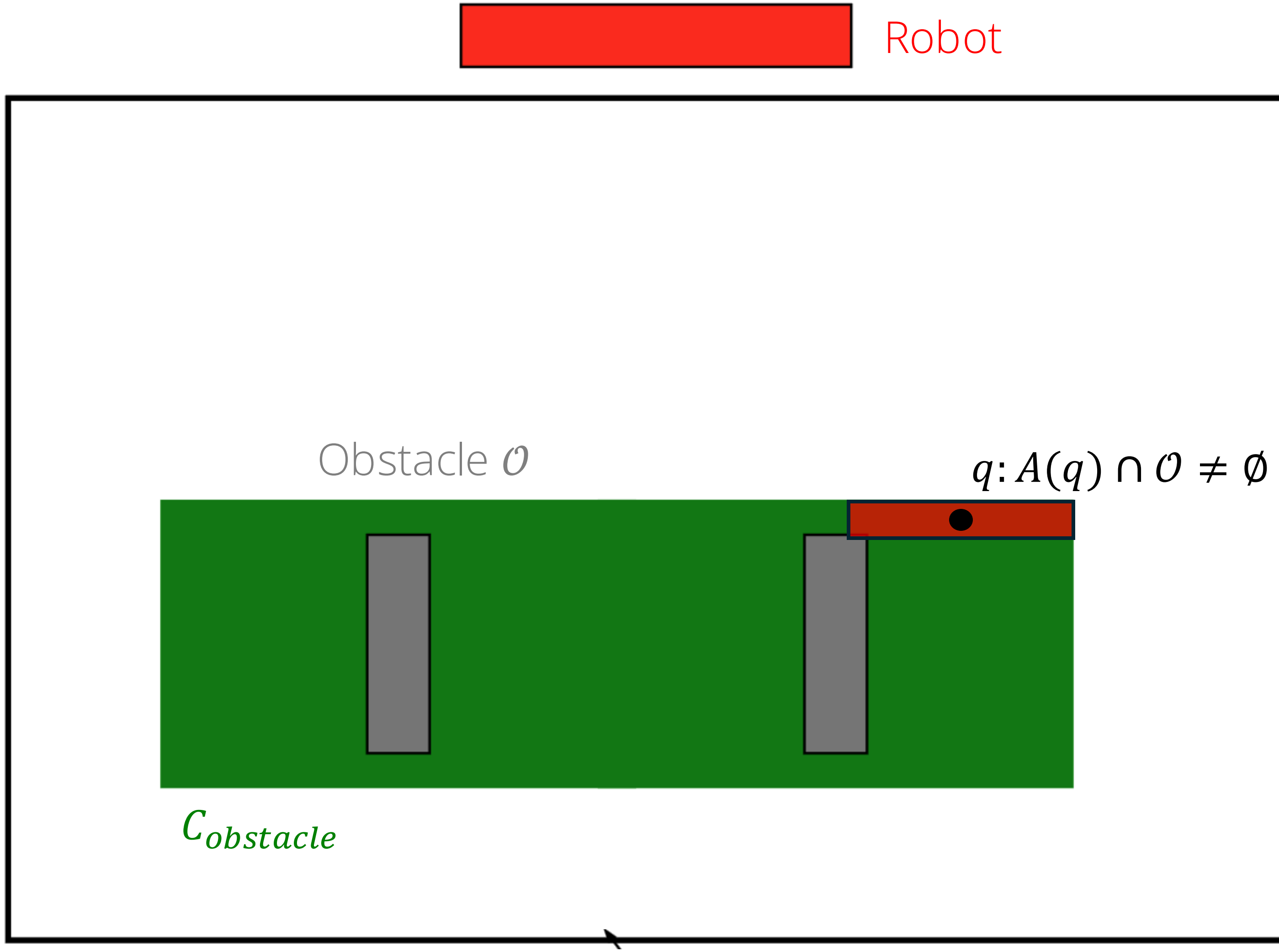


Image credit H. Choset

Example: Rotational Motion

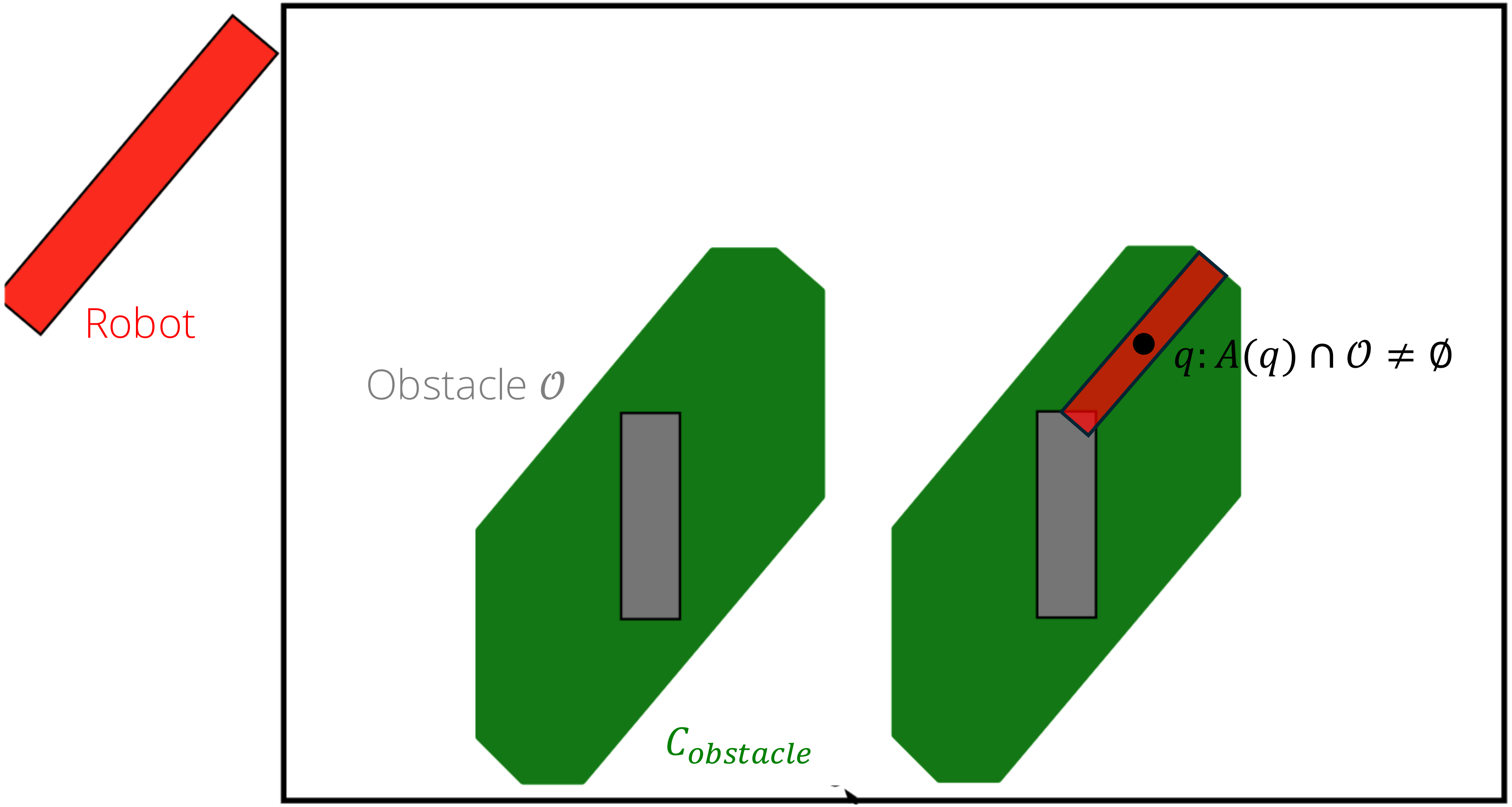
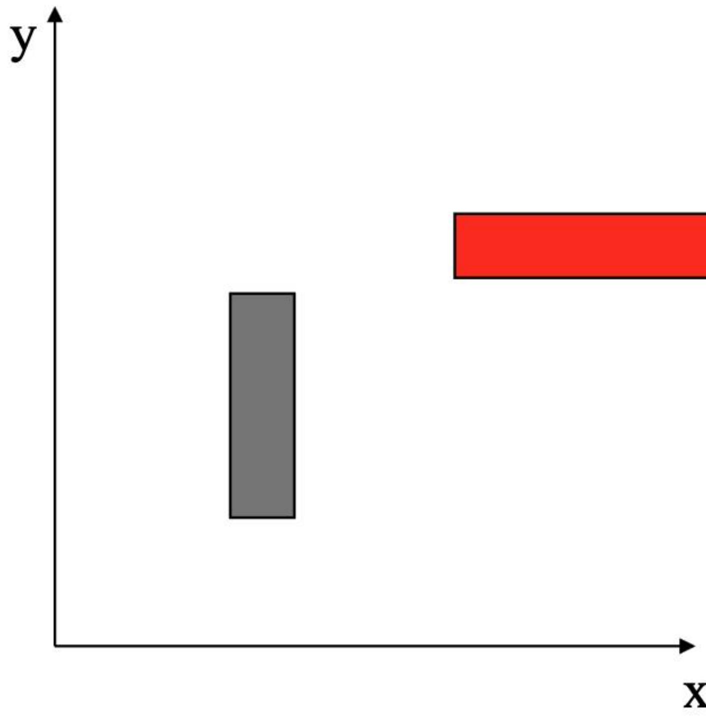


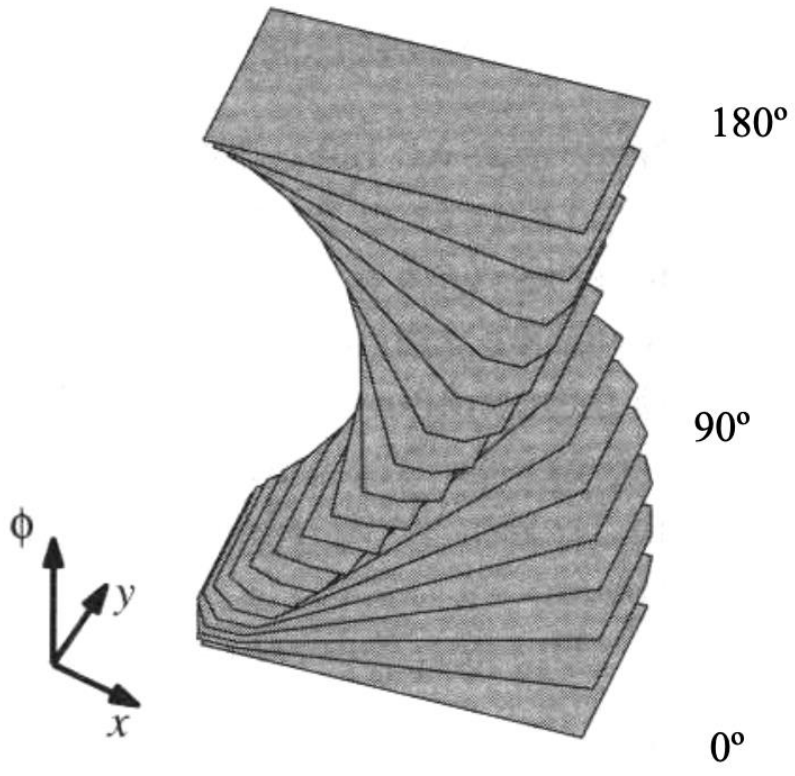
Image credit H. Choset

What would the configuration space of a rectangular robot (red) in this world look like?

(The obstacle is blue.)

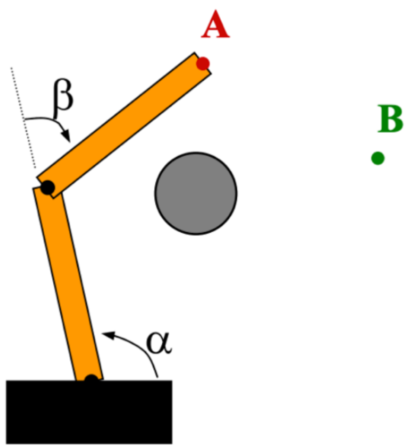


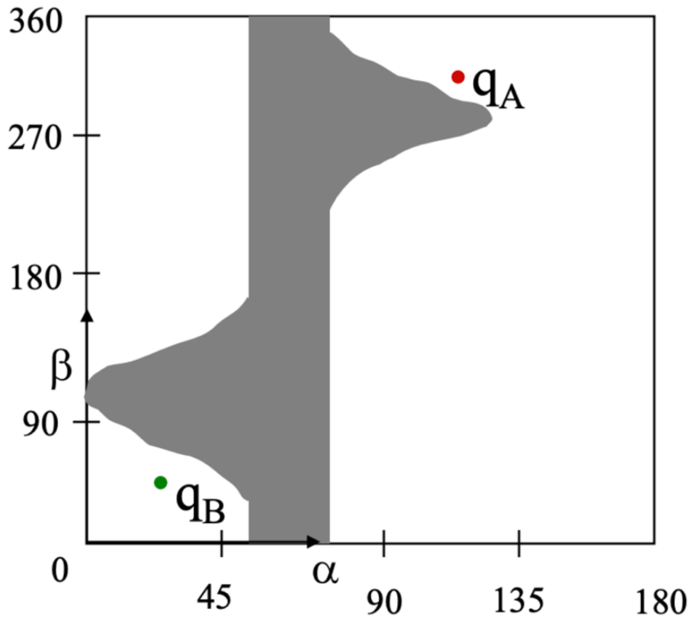
configuration space

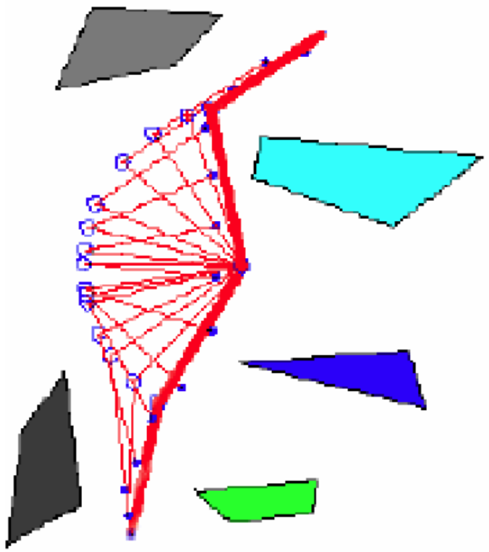


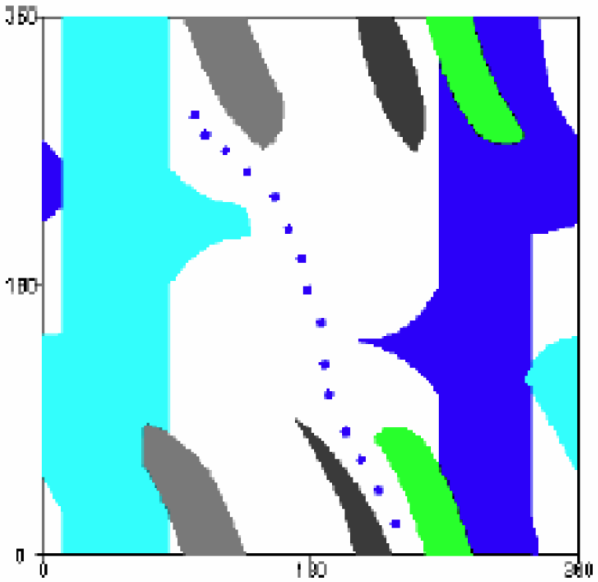
this is twisted...

Example: 2-Link Planar Arm

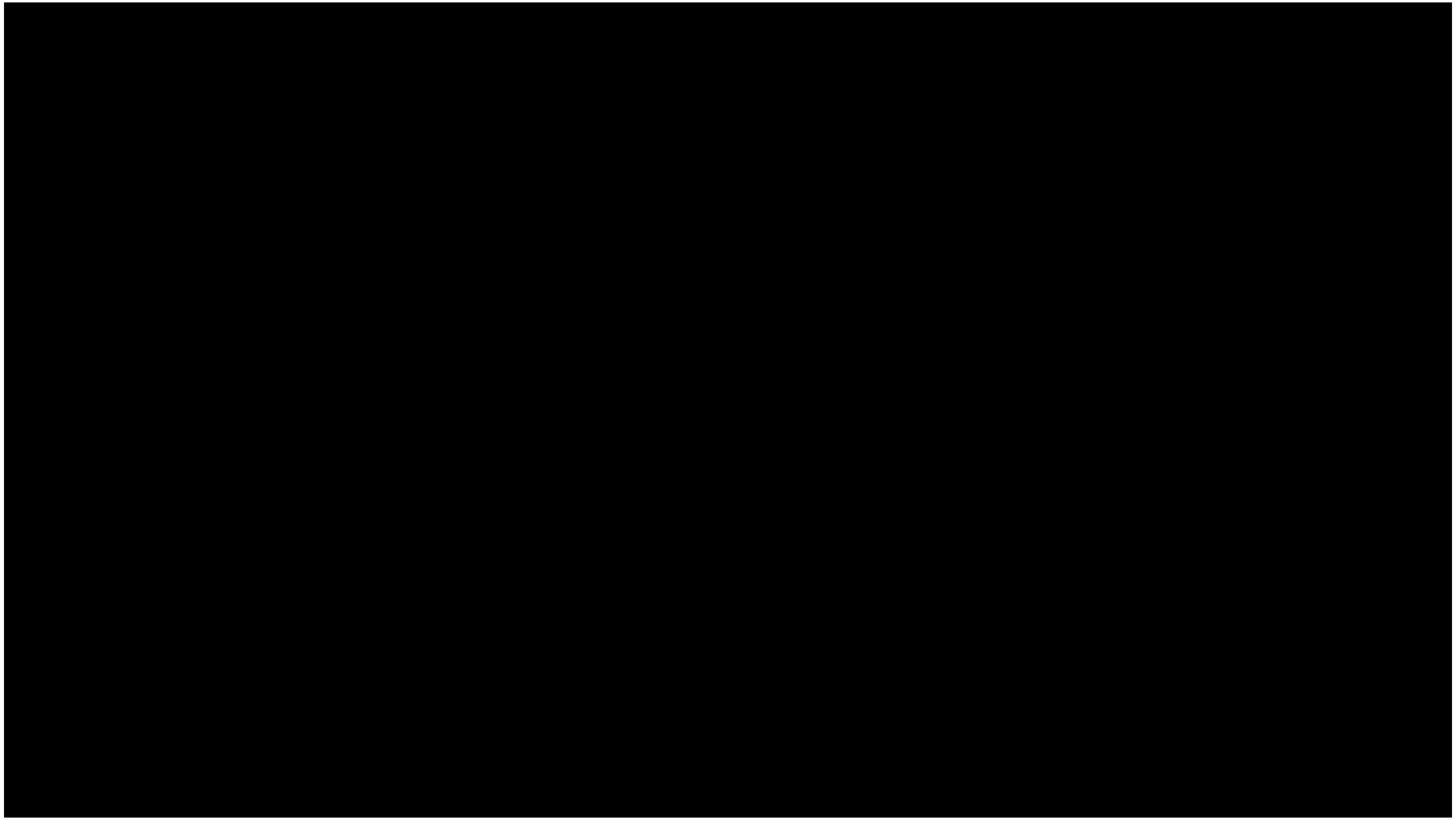






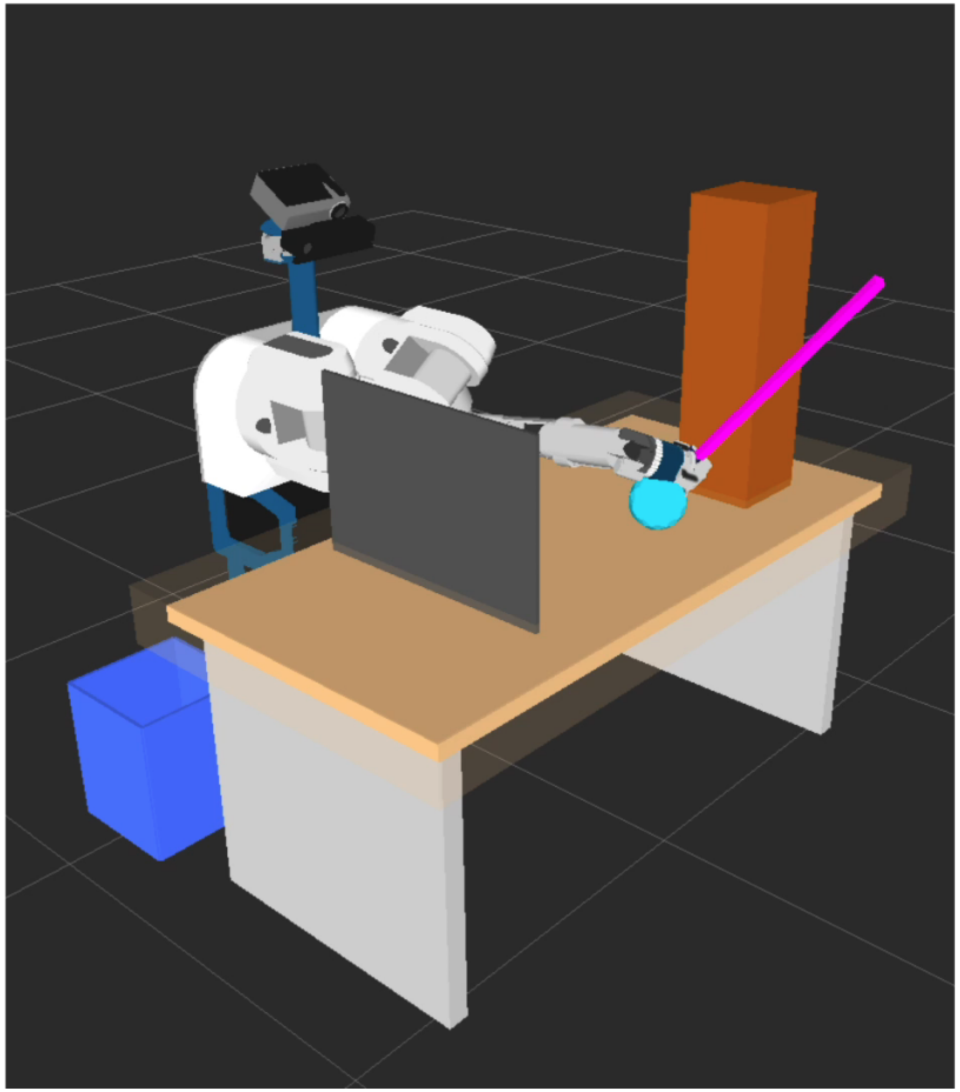


Motion Planning is Hard….



https://www.youtube.com/watch?v=SBFwgR4K1Gk

Geometric Path Planning Problem



Also known as

Piano Mover's Problem (Reif 79)

Given:

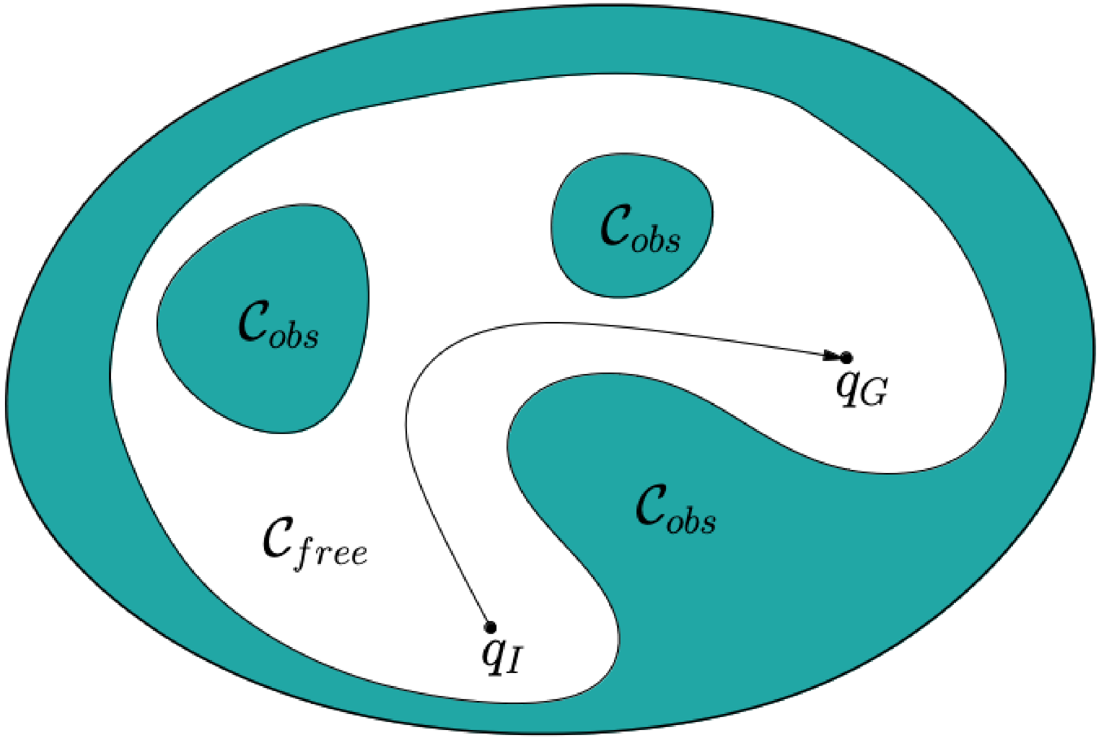
1. A *workspace* $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An *obstacle region* $\mathcal{O} \subset \mathcal{W}$.
3. A *robot* defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of m links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The *configuration space* $\mathcal{C}$ (C_{obs} and C_{free} are then defined).
5. An *initial configuration* $\boldsymbol{q_I} \in \mathcal{C}_{free}$.
6. A *goal configuration* $\boldsymbol{q_G} \in \mathcal{C}_{free}$. The initial and goal configuration are often called a *query* $(\boldsymbol{q_I}, \boldsymbol{q_G})$.

Compute a (continuous) path, $\tau : [0,1] \to \mathcal{C}_{free}$, *such that* $\tau(0) = \boldsymbol{q_I}$ *and* $\tau(1) = \boldsymbol{q_G}$.

Also may want to minimize cost $c(\tau)$

Geometric Path Planning Problem

We can apply previous techniques (e.g. grid-search, visibility graph, cell decomposition) here!



Also known as

Piano Mover's Problem (Reif 79)

Given:

1. A *workspace* $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An *obstacle region* $\mathcal{O} \subset \mathcal{W}$.
3. A *robot* defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of m links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The *configuration space* $\mathcal{C}$ (C_{obs} and C_{free} are then defined).
5. An *initial configuration* $\boldsymbol{q_I} \in \mathcal{C}_{free}$.
6. A *goal configuration* $\boldsymbol{q_G} \in \mathcal{C}_{free}$. The initial and goal configuration are often called a *query* $(\boldsymbol{q_I}, \boldsymbol{q_G})$.

Compute a (continuous) path, $\tau : [0,1] \rightarrow \mathcal{C}_{free}$, *such that* $\tau(0) = \boldsymbol{q_I}$ *and* $\tau(1) = \boldsymbol{q_G}$.

Also may want to minimize cost $c(\tau)$

Image credit S. LaValle

Motion Planning is Hard….

<u>10 vertices per dim</u>

Dimension d	# vertices
2	100
3	1,000
6	1,000,000
8	100,000,000
10	10,000,000,000
15	1,000,000,000,000,000
20	100,000,000,000,000,000,000

- n-joint robot has n-dim C-space
- Assume we have M vertices per dim. The n-dim C-space has M^n vertices…
- In high dimensions:
  - ➢ Computing the C-space obstacle is hard
  - ➢ Planning is hard

Motion Planning is Hard….

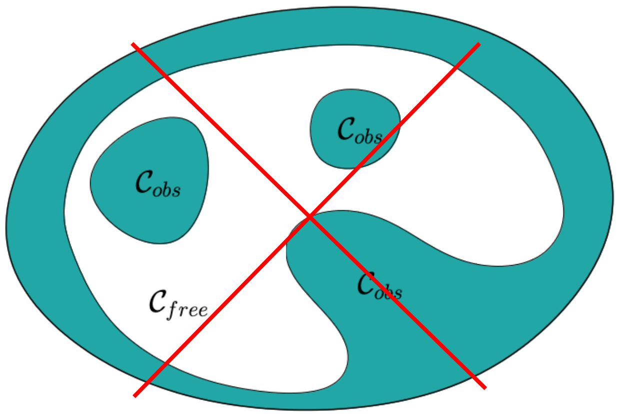
10 vertices per dim

Dimension d	# vertices
2	100
3	1,000
6	1,000,000
8	100,000,000
10	10,000,000,000
15	1,000,000,000,000,000
20	100,000,000,000,000,000,000

- n-joint robot has n-dim C-space
- Assume we have M vertices per dim. The n-dim C-space has M^n vertices…
- In high dimensions:
  - ➢ Computing the C-space obstacle is hard
    Solution: Don't compute C_{obs} explicitly, instead we query a collision detector
  - ➢ Planning is hard
    Solution: Don't search a path from all vertices, but sampled vertices

- Idea: Don't compute until being queried!

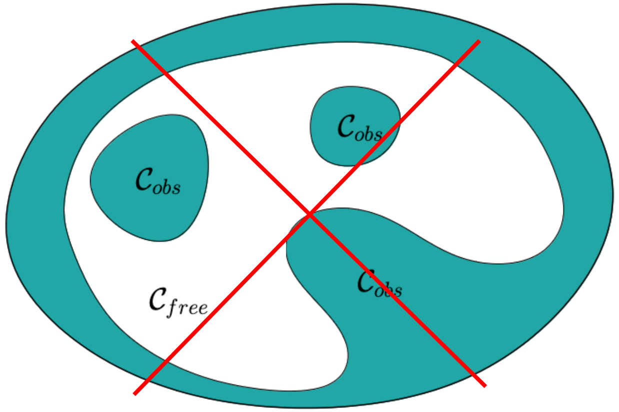
Implicit C-Obstacle Representations



- Feasibility query:

$$Feasible(q) = \begin{cases} 1, & if\ q\ is\ in\ the\ free\ space \\ 0, & if\ q\ is\ in\ the\ obstacle\ space \end{cases}$$

Implicit C-Obstacle Representations

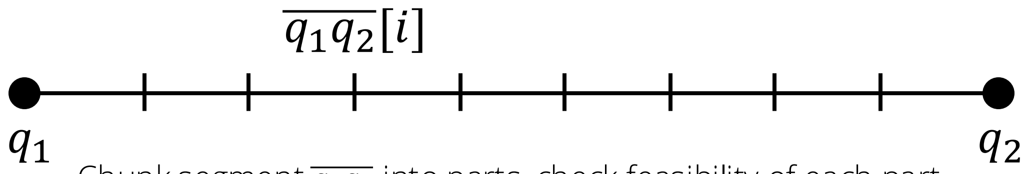


- Feasibility query:

$$Feasible(q) = \begin{cases} 1, & if\ q\ is\ in\ the\ free\ space \\ 0, & if\ q\ is\ in\ the\ obstacle\ space \end{cases}$$

- Visibility query:

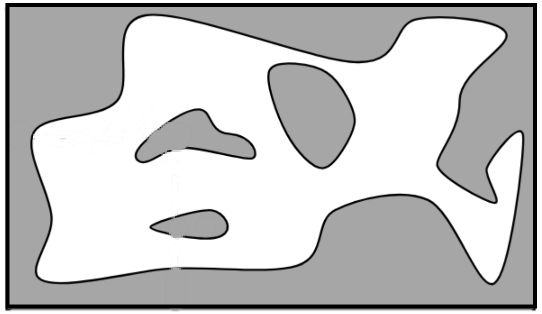
$$Visible(q_1, q_2) = \begin{cases} 1, & if\ \overline{q_1 q_2}\ is\ completly\ in\ the\ free\ space \\ 0, & if\ \overline{q_1 q_2}\ intersects\ the\ obstacle\ space \end{cases}$$



Chunk segment $\overline{q_1 q_2}$ into parts, check feasibility of each part

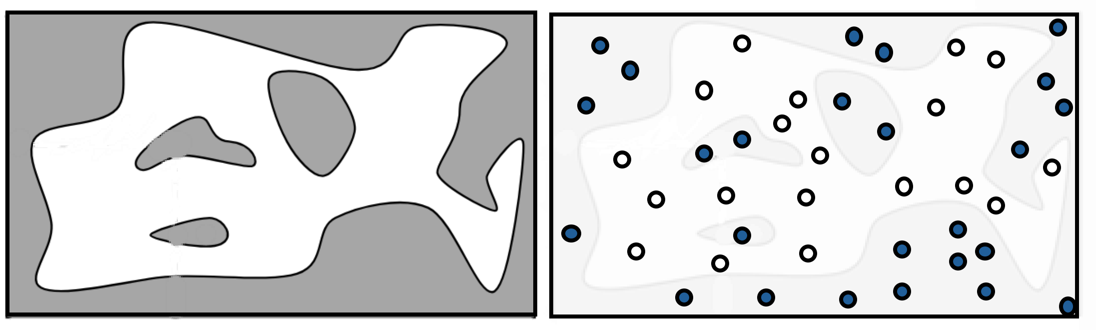
Probabilistic Roadmaps (PRM)

We don't' have an explicit feasibility map. Just for reference...



RPM Algorithm:

Probabilistic Roadmaps (PRM)



$Feasible(q)$

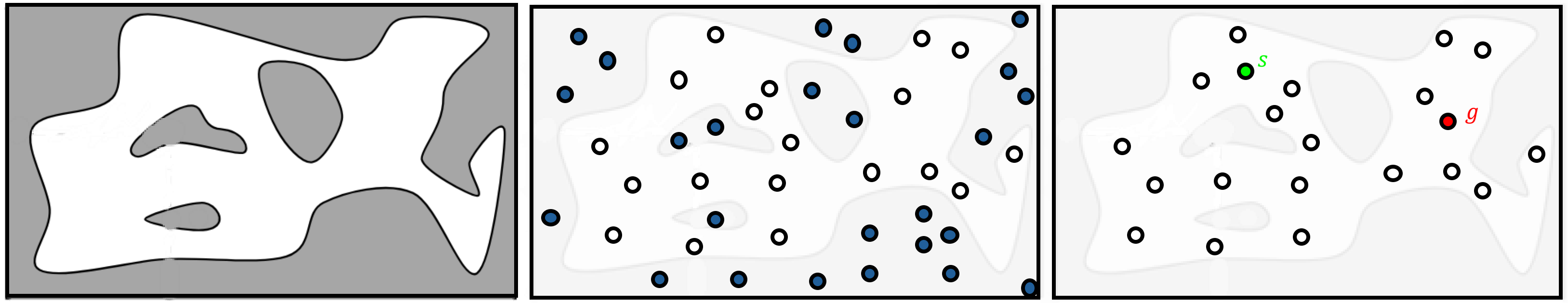
RPM Algorithm:

1. Sample random N configurations from the C-space and query their feasibilities

Slide adapted from K. Hauser

Probabilistic Roadmaps (PRM)

We don't't have an explicit feasibility map. Just for reference...

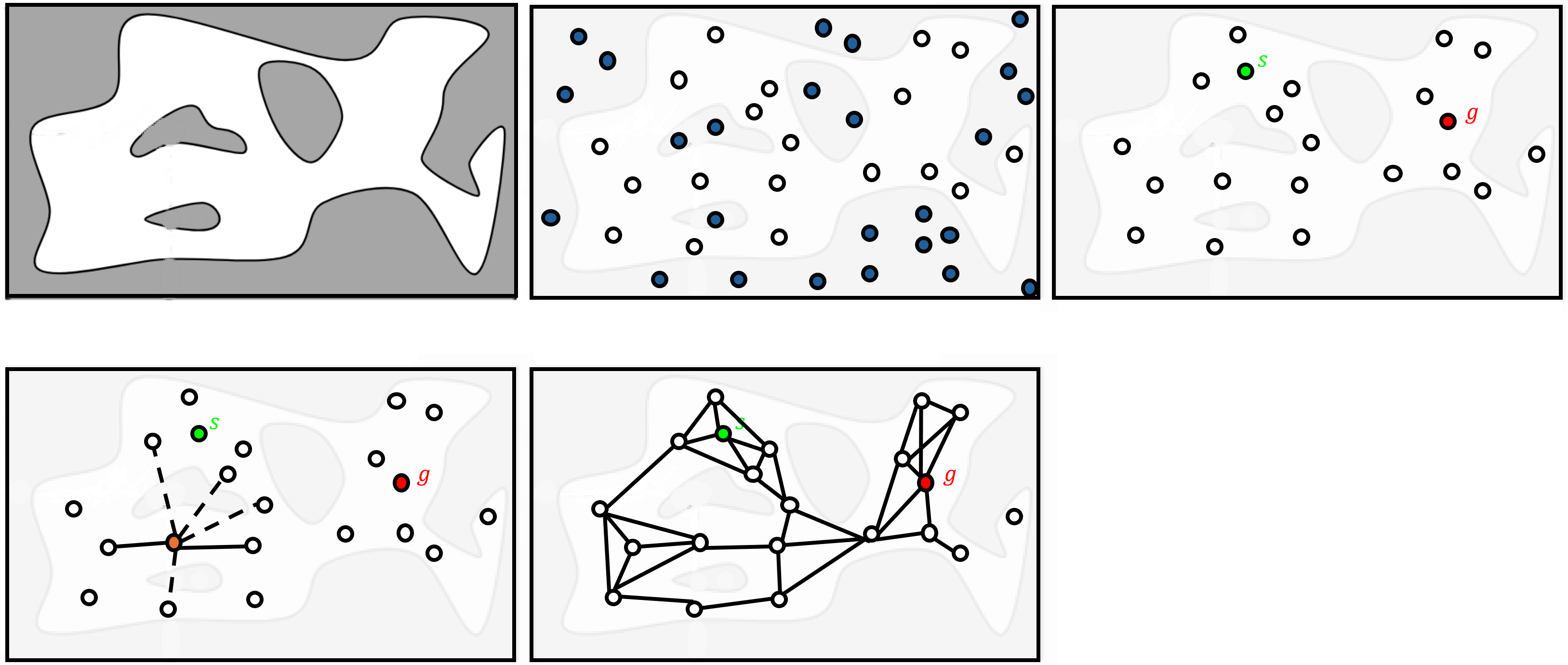


<u>RPM Algorithm:</u>

1. Sample random N configurations from the C-space and query their feasibilities
2. Add **milestones** (all feasible, the start and the goal configurations).

Probabilistic Roadmaps (PRM)

We don't' have an explicit feasibility map. Just for reference…



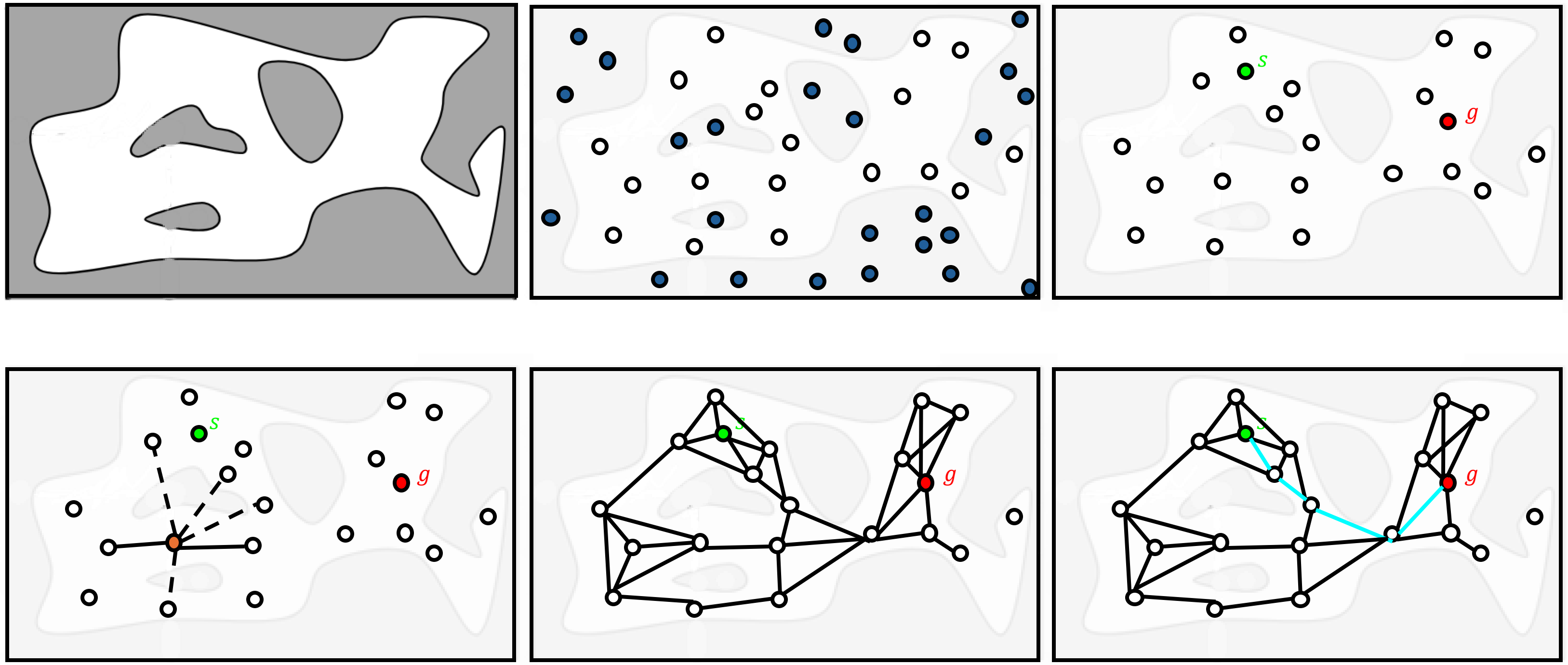
$Visible(q,p)$

RPM Algorithm:

1. Sample random N configurations from the C-space and query their feasibilities
2. Add **milestones** (all feasible, the start and the goal configurations).
3. Connect pairs of neighboring **milestones** if they are visible

Probabilistic Roadmaps (PRM)

We don't't have an explicit feasibility map. Just for reference…



RPM Algorithm:

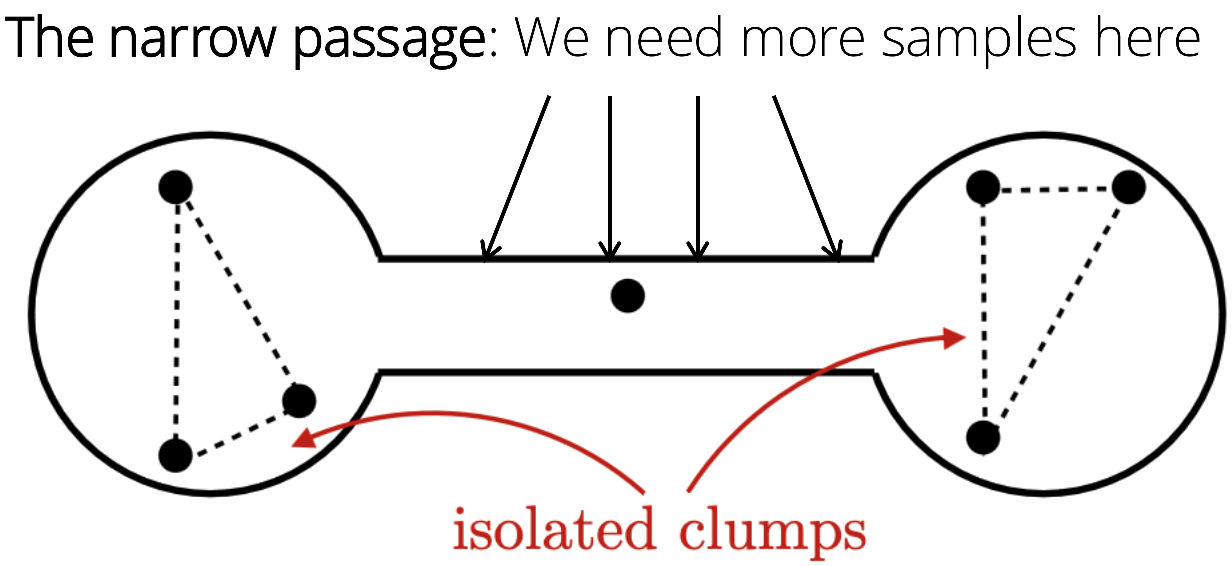
1. Sample random N configurations from the C-space and query their feasibilities
2. Add **milestones** (all feasible, the start and the goal configurations).
3. Connect pairs of neighboring **milestones** if they are visible
4. Search for a path from the start to the goal

Probabilistic Roadmaps (PRM)

Algorithm Basic-PRM(s,g,N)

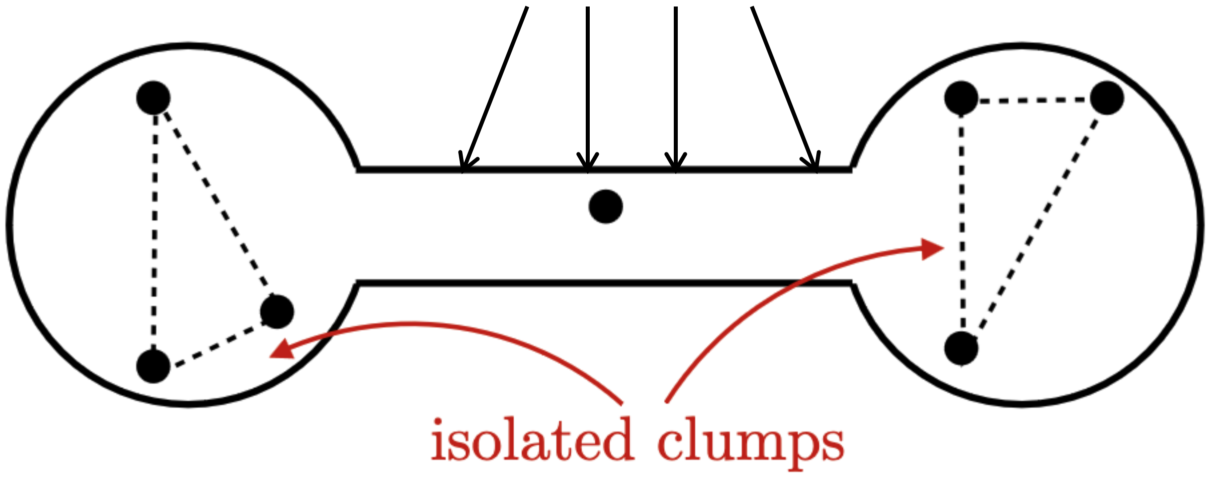
1. $V \leftarrow \{s, g\}$.
2. $E \leftarrow \{\}$.
3. **for** $i = 1, \ldots, N$ **do**
4. $\quad q \leftarrow Sample()$
5. $\quad$ **if** not Feasible(q) **then** return to Line 3.
6. $\quad$ Add q to V (add q as a new milestone)
7. $\quad$ **for** all $p \in near(q, V)$
8. $\quad\quad$ **if** Visible(p, q) **then**
9. $\quad\quad\quad$ Add (p, q) to E.
10. Search $G = (V, E)$, with Cartesian distance as the edge cost, to connect s and g.
11. **return** the path if one is found.

Is Uniformly Randomly Sampling Good Enough?



Is Uniformly Randomly Sampling Good Enough?

The narrow passage: We need more samples here

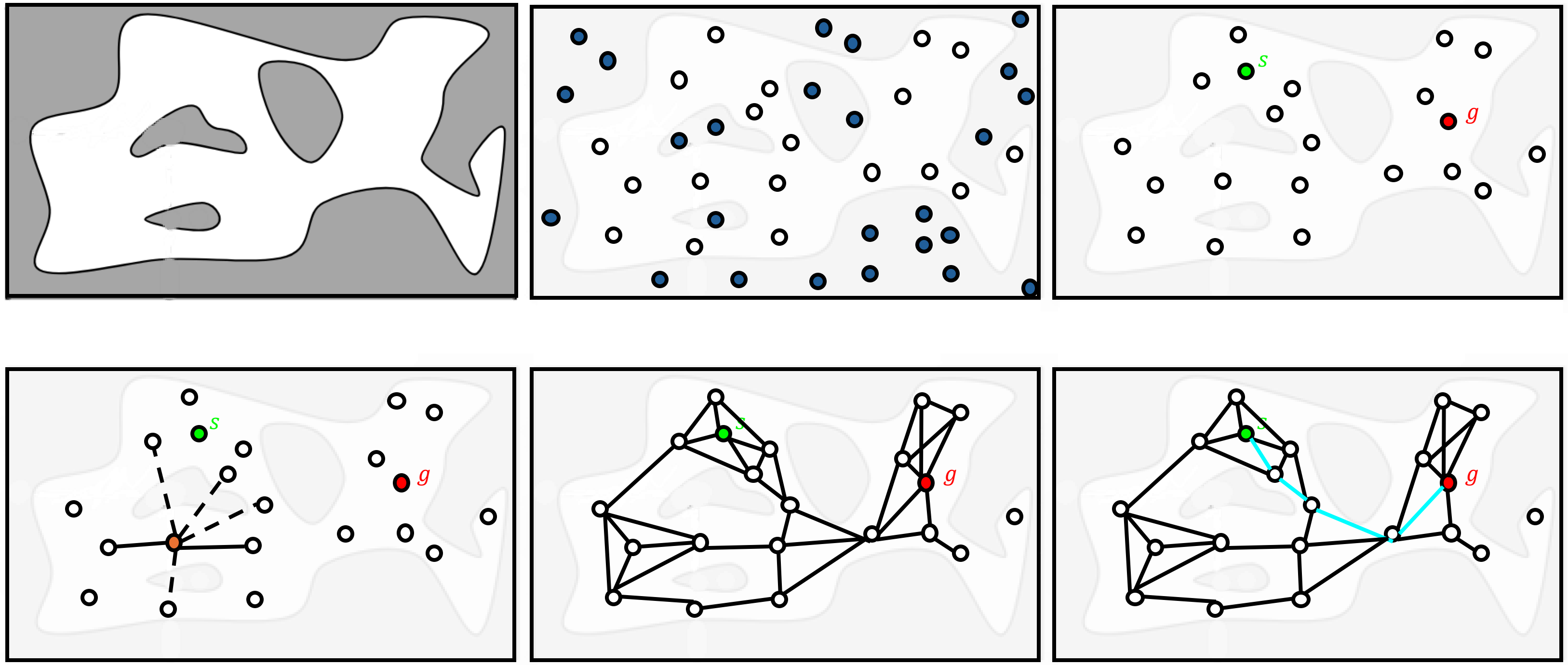


Solutions:

1. Sample near obstacle surface
2. Add samples that are in between two obstacles
3. Train a learner to detect the narrow passages

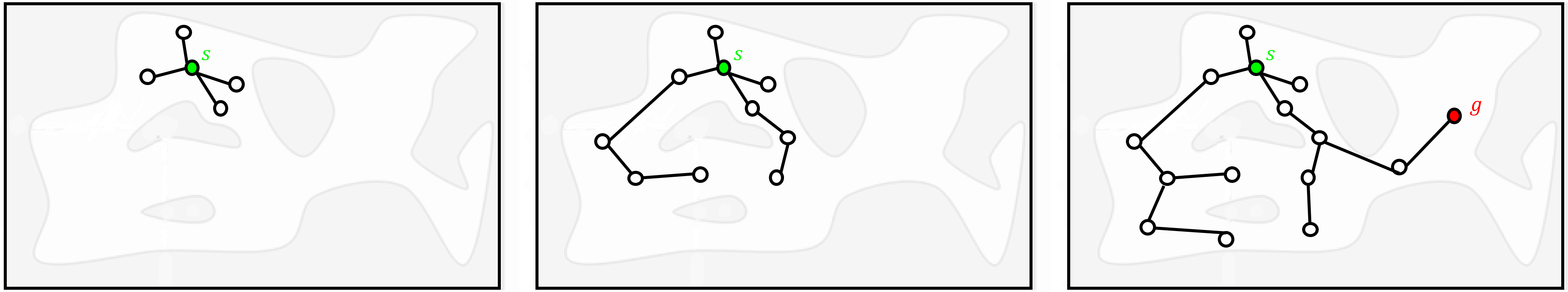
Quick summary of PRM

We don't' have an explicit feasibility map. Just for reference...



- PRM randomly samples configurations to build a roadmap, hoping to cover the free space
- PRM is great when we want to reuse the graph. For example, plan many times with different start/goal pairs
- What if we just need to plan once? For example, a robot only needs to *pick up* the object once

Rapidly Exploring Random Trees (RRTs)



Grow a tree of feasible paths from the start to the goal configuration, instead of building a graph.

Rapidly Exploring Random Trees (RRTs)

```
BUILD_RRT(q_init)
  T.init(q_init);
  for k = 1 to K do
      q_rand ← RANDOM_CONFIG();
      EXTEND(T, q_rand);
  Return T
```

Sample random configuration with probability p, and the goal with probability $1 - p$

RRT-Connect: An Efficient Approach to Single-Query Path Planning. J. Kuffner and S. LaValle.

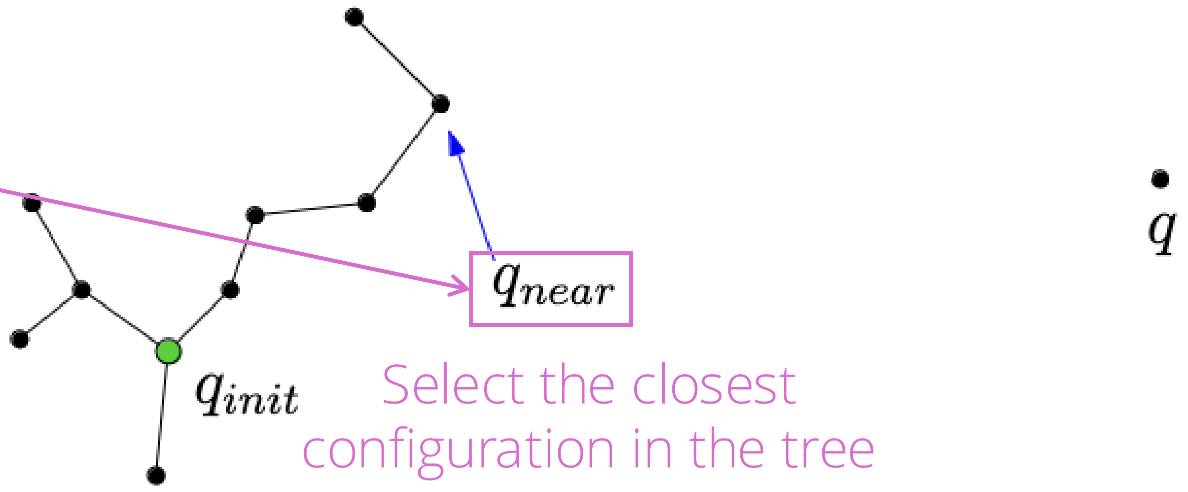
Slide adapted from M. Likhachev

Rapidly Exploring Random Trees (RRTs)

```
BUILD_RRT(q_init)
    T.init(q_init);
    for k = 1 to K do
        q_rand ← RANDOM_CONFIG();
        EXTEND(T, q_rand);
    Return T
```

```
EXTEND(T, q)
    q_near ← NEAREST_NEIGHBOR(q, T);
    if NEW_CONFIG(q, q_near, q_new) then
        T.add_vertex(q_new);
        T.add_edge(q_near, q_new);
        if q_new = q then
            Return Reached;
        else
            Return Advanced;
    Return Trapped;
```

How to expand the tree? Expand from the node that is the closest to the configured q



RRT-Connect: An Efficient Approach to Single-Query Path Planning. J. Kuffner and S. LaValle.

Slide adapted from M. Likhachev

Rapidly Exploring Random Trees (RRTs)

BUILD_RRT(q_{init})

$\mathcal{T}$.init(q_{init});

for $k = 1$ **to** K **do**

  $q_{rand} \leftarrow$ RANDOM_CONFIG();

  EXTEND($\mathcal{T}, q_{rand}$);

Return $\mathcal{T}$

EXTEND($\mathcal{T}, q$)

$q_{near} \leftarrow \boxed{\text{NEAREST\_NEIGHBOR}(q, \mathcal{T})}$;

if $\boxed{\text{NEW\_CONFIG}(q, q_{near}, q_{new})}$ **then**

  $\mathcal{T}$.add_vertex(q_{new});

  $\mathcal{T}$.add_edge(q_{near}, q_{new});

  if $q_{new} = q$ **then**

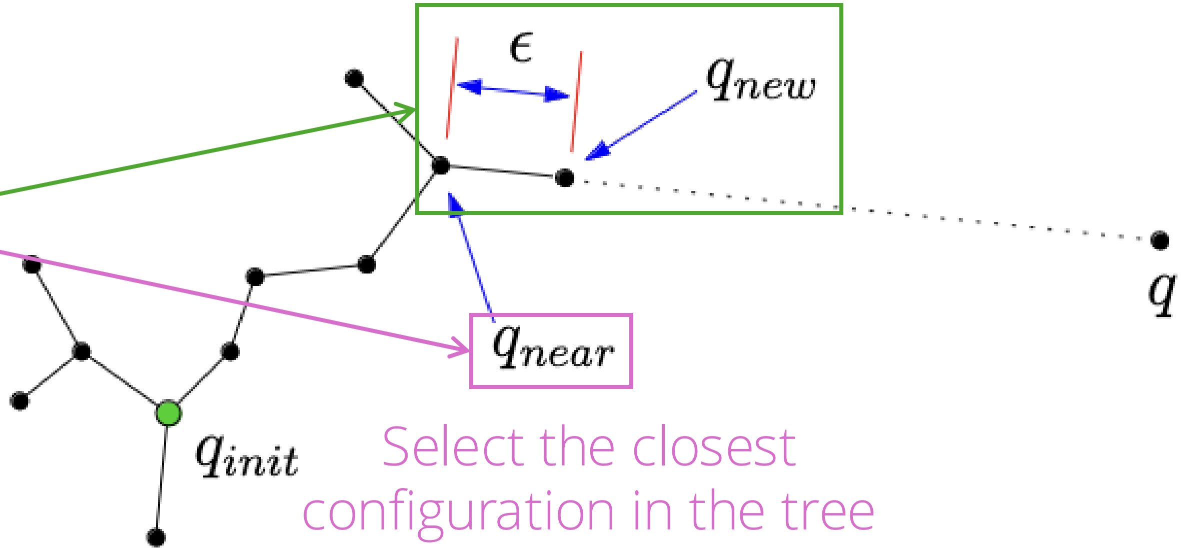
    Return *Reached*;

  else

    Return *Advanced*;

Return *Trapped*;

Move by at most ε from q_{near} to q



Select the closest configuration in the tree

RRT-Connect: An Efficient Approach to Single-Query Path Planning. J. Kuffner and S. LaValle.

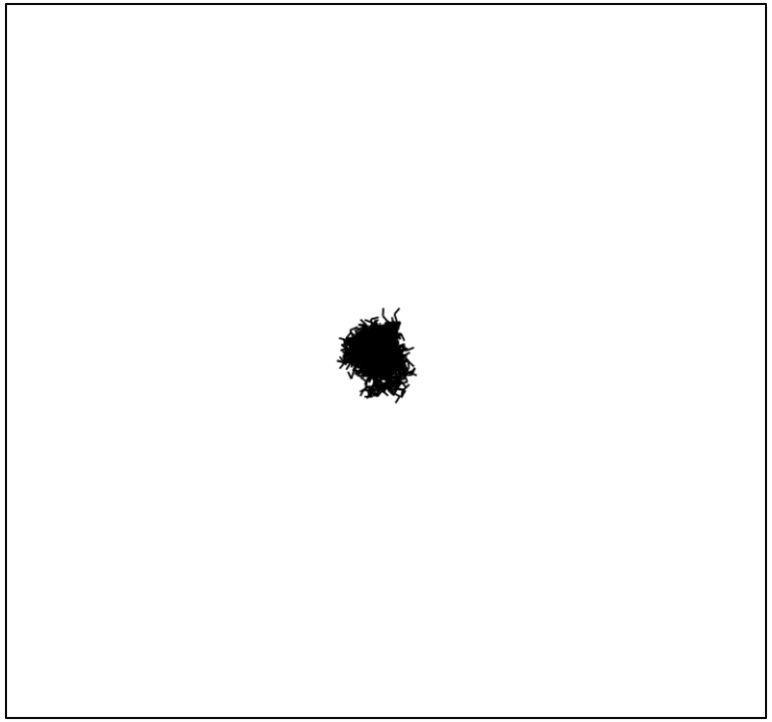
Slide adapted from M. Likhachev

Expansion Strategy

BUILD_RRT(q_{init})
$\mathcal{T}$.init(q_{init});
for $k = 1$ **to** K **do**
$q_{rand} \leftarrow$ $\boxed{\text{RANDOM\_CONFIG}();}$
EXTEND($\mathcal{T}, q_{rand}$);
Return $\mathcal{T}$

Sample random configuration with probability p, and the goal with probability $1 - p$

Randomly uniformly sampling



Biased sampling toward unexplored regions

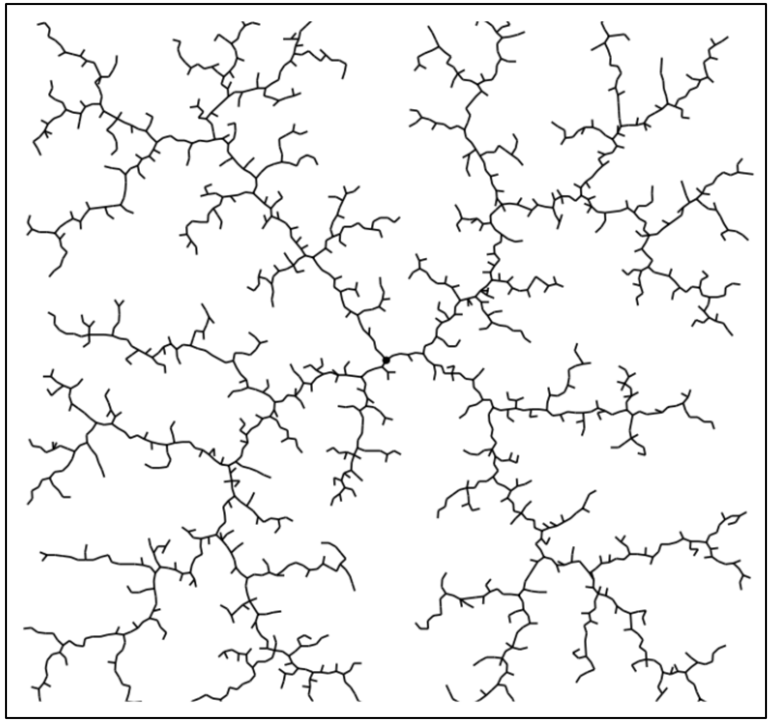
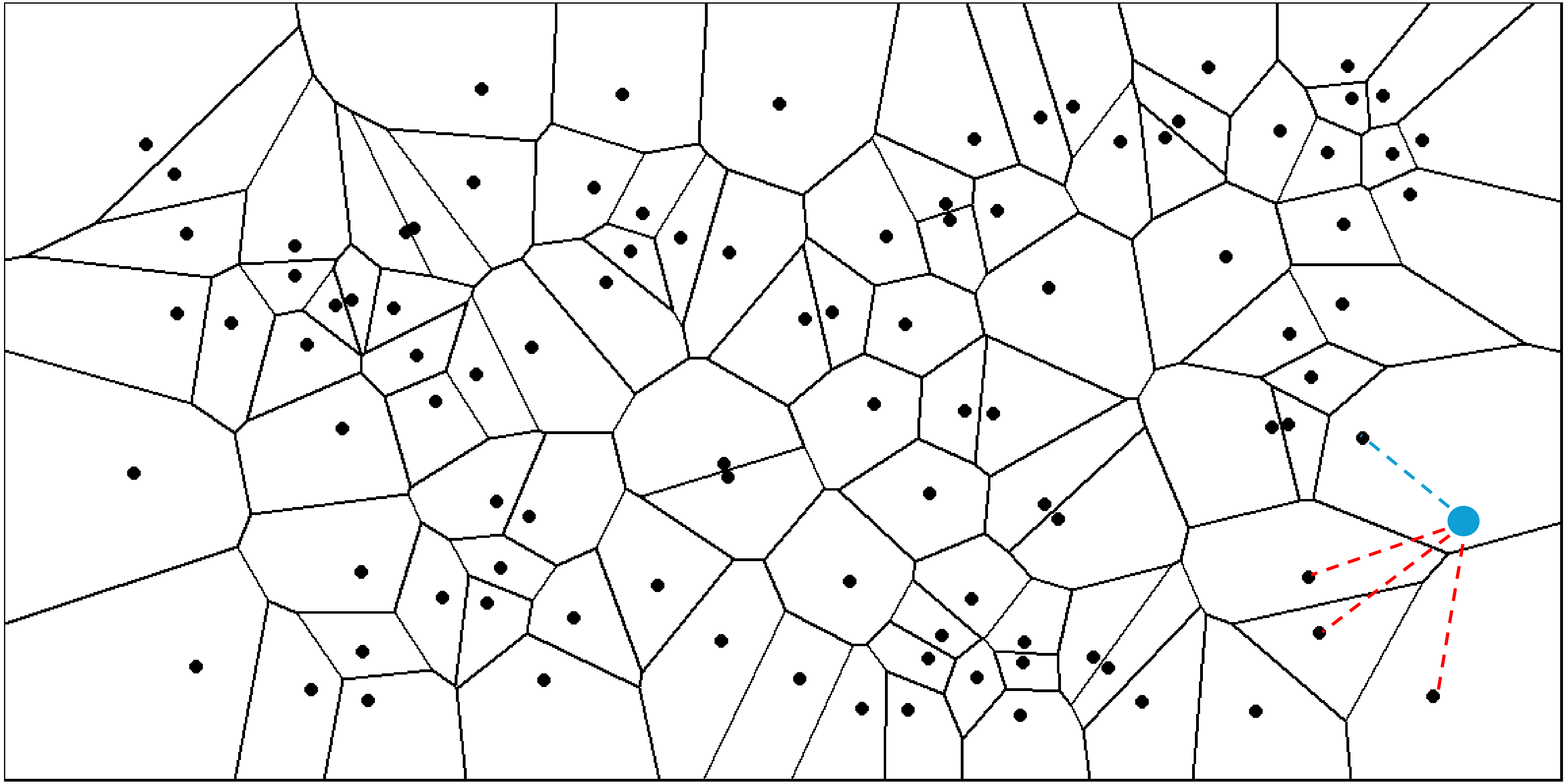


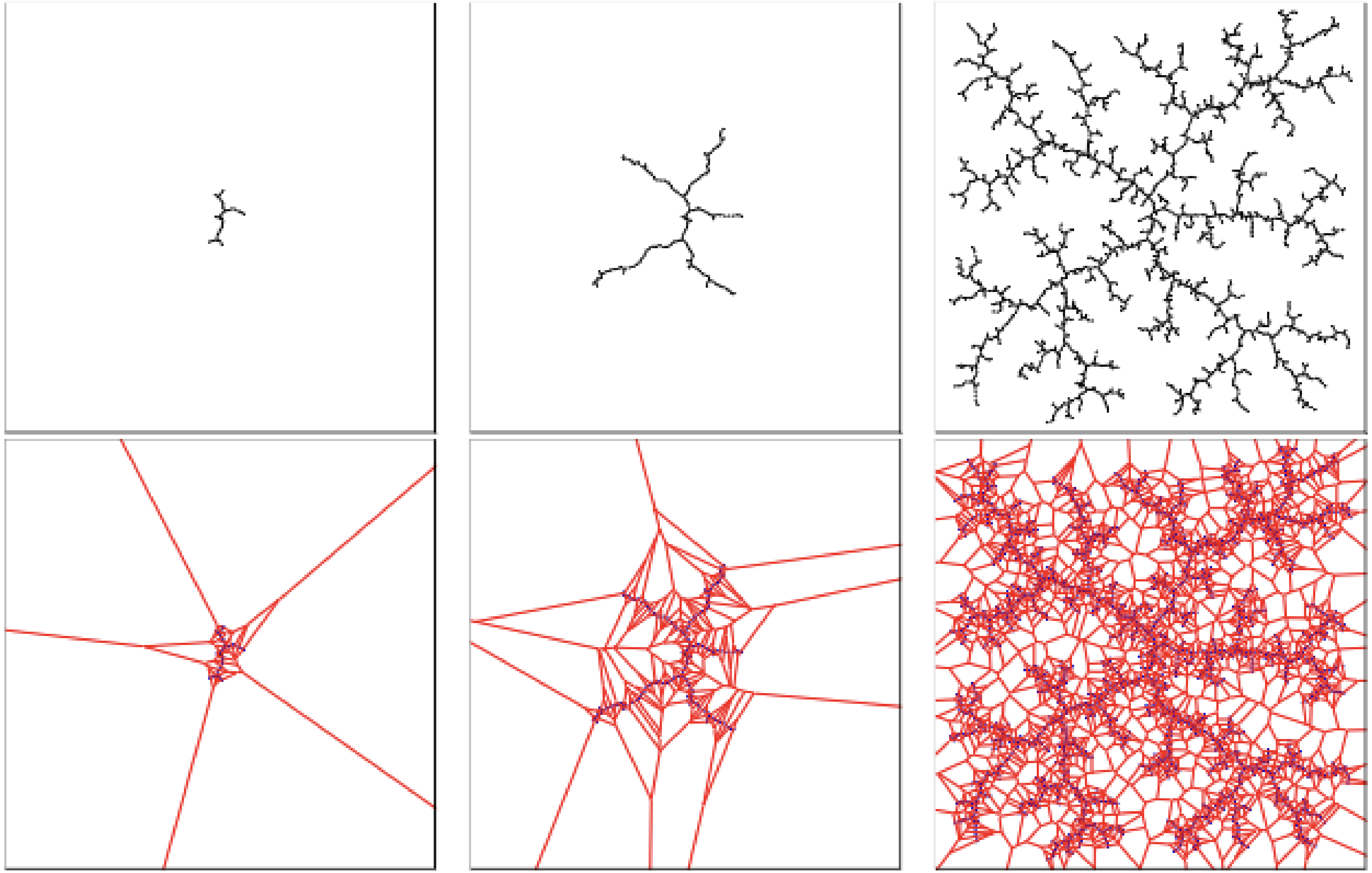
Image credit D. Fox

Voronoi Diagram

Voronoi diagram: nearest-neighbor segmentation. Assign each pixel to the nearest node. We can check the area of each region and explore more in the large ones.



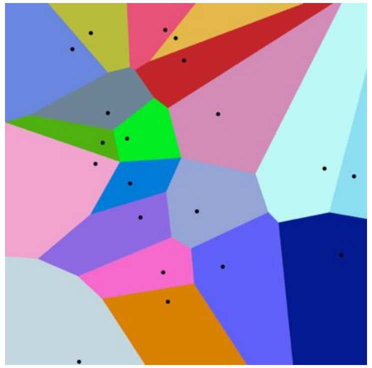
Voronoi Bias Strategy



Sample random configuration with probability p, which is proportional to the volume of its Voronoi cell.

- Bias sampling toward unexplored areas.

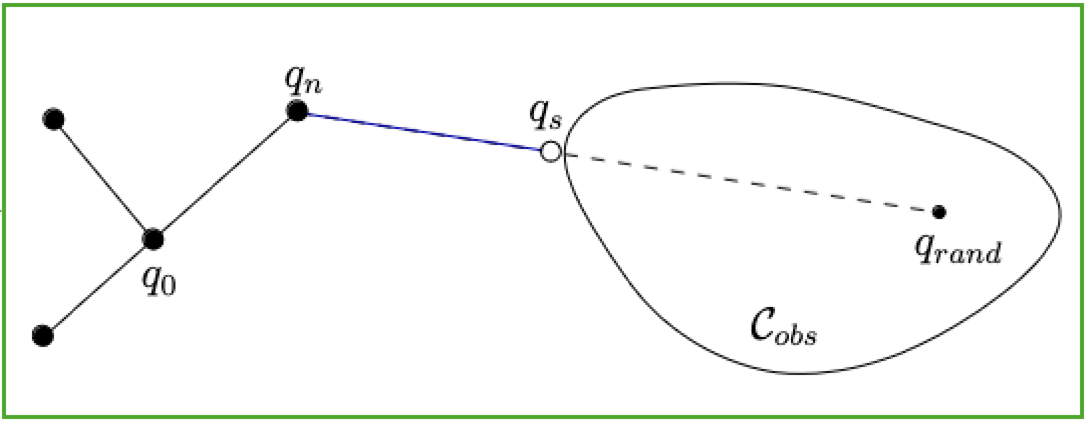
Voronoi diagram: nearest-neighbor segmentation



RRT-Connect: An Efficient Approach to Single-Query Path Planning. J. Kuffner and S. LaValle.

RRT-Connect: Bi-direction RRTs

```
CONNECT(T, q)
  repeat
      S ← EXTEND(T, q);
  until not (S = Advanced)  # new configuration is added
  Return S;
```



Greedily move from q_{near} to q_{rand}

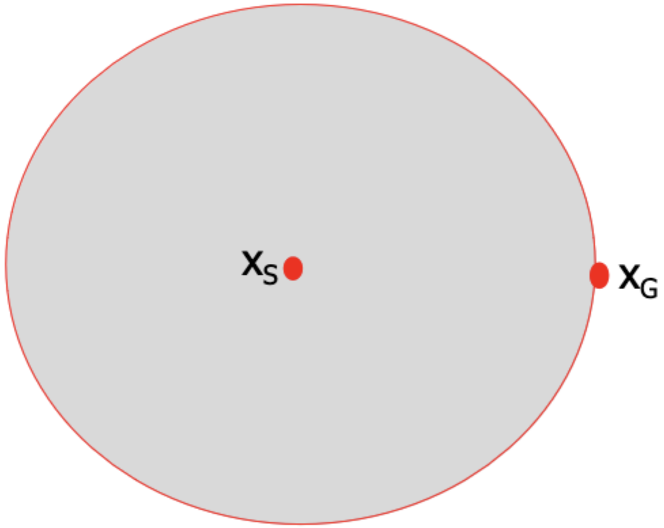
```
RRT_CONNECT_PLANNER(q_init, q_goal)
  T_a.init(q_init); T_b.init(q_goal);
  for k = 1 to K do
      q_rand ← RANDOM_CONFIG();
      if not (EXTEND(T_a, q_rand) = Trapped) then
          if (CONNECT(T_b, q_new) = Reached) then
              Return PATH(T_a, T_b);
      SWAP(T_a, T_b);
  Return Failure
```

Grow two trees, one from the start and another from the goal

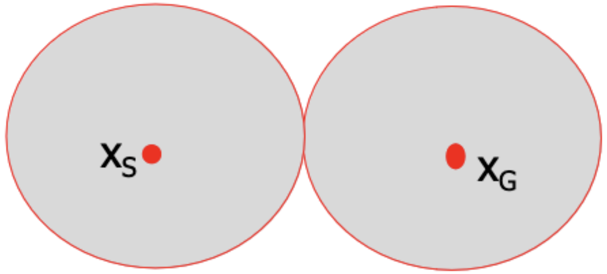
RRT-Connect: An Efficient Approach to Single-Query Path Planning. J. Kuffner and S. LaValle.

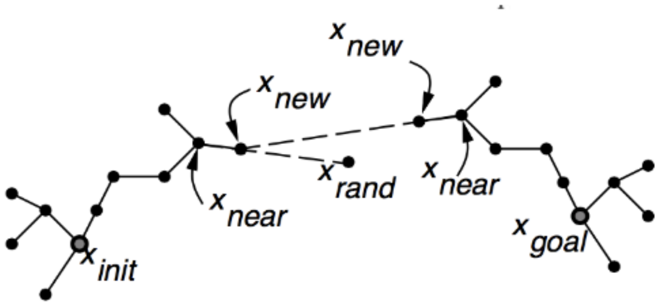
Single Tree vs. Double Trees

- Volume swept out by unidirectional RRT:



- Volume swept out by bi-directional RRT:

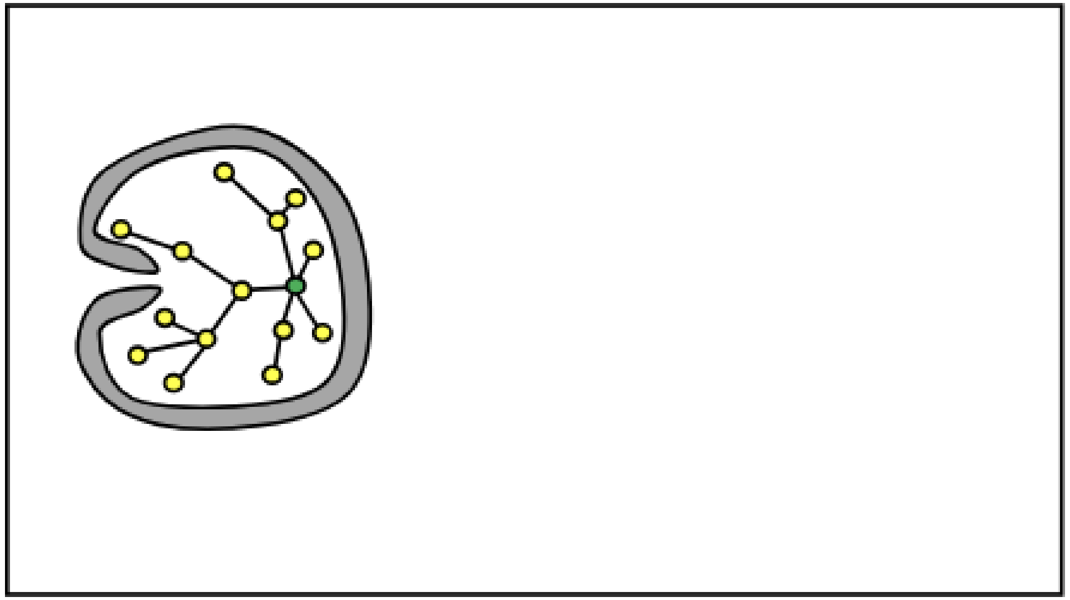


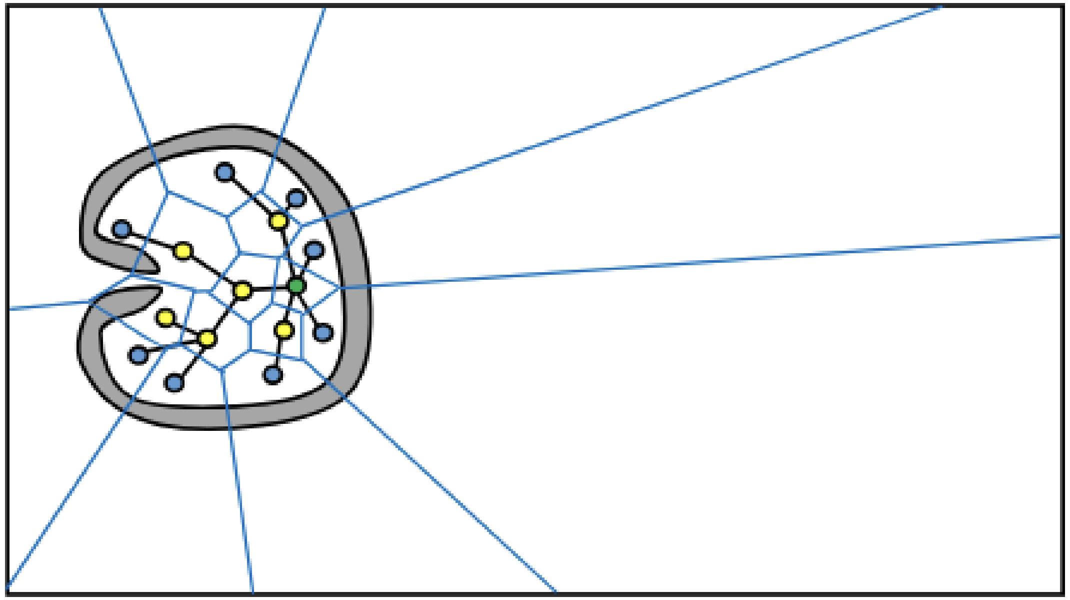


- Difference more and more pronounced as dimensionality increases

Image credit D. Fox

RRT Still Has Problems…



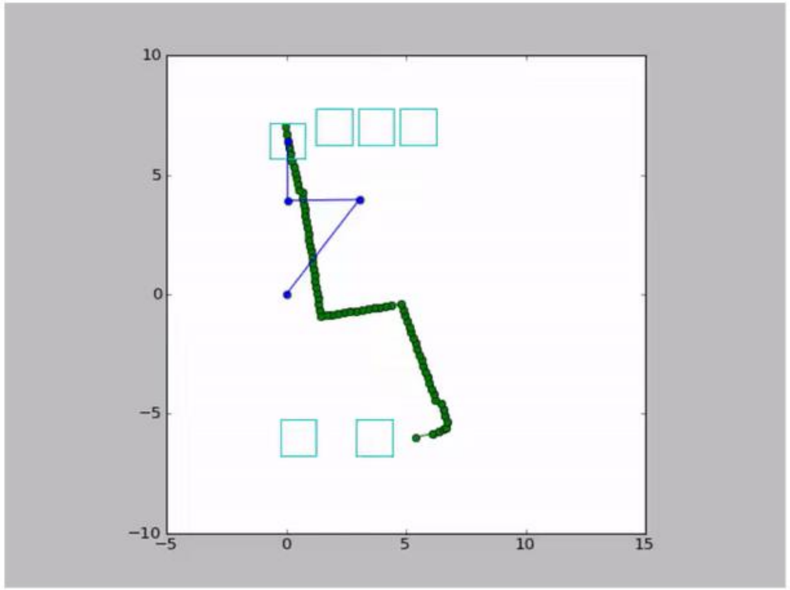


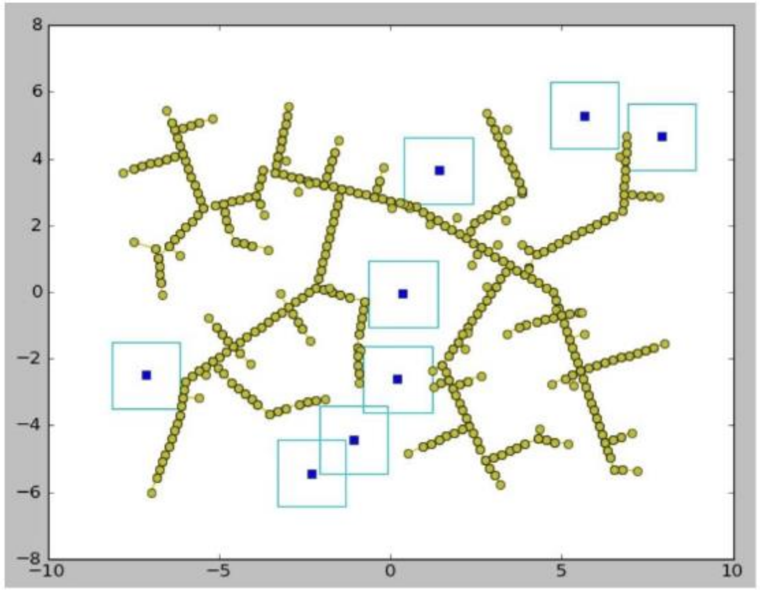
- The “bugtrap” problem: due to the Voronoi bias, RRT frequently attempts infeasible extensions
- To escape the mouth of a bugtrap, we need to sample a very carefully chosen sequence of milstones within the general area that it has already explored
- A tradeoff between exploring new regions and refine the roadmap of explored areas

Slide adapted from K. Hauser

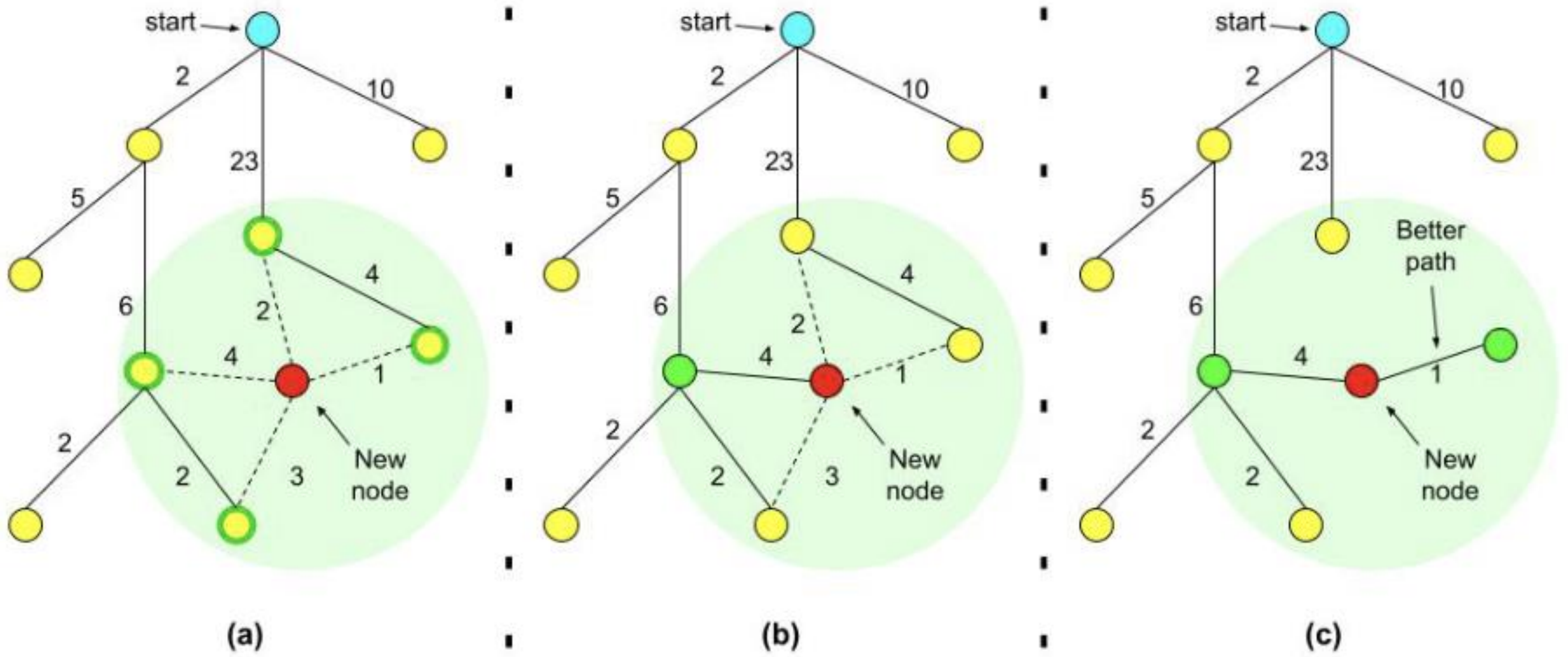
RRT Still Has Problems…

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





RRT*: RRT + Re-Wiring



- Can we find more optimal path passing through q_{new} ?
- Can we find more optimal path to q_{new} ?

Integrating Rigidity Analysis into the Exploration of Protein Conformational Pathways Using RRT* and MC. F. Afrasiabi et al.

RRT*: RRT + Re-Wiring

Algorithm RRT*

1. $T \leftarrow \{s\}$..
2. **for** $i = 1, \ldots, N$ **do**
3. $q_{rand} \leftarrow Sample()$
4. $q_e \leftarrow$Extend-Tree(T, q_{rand}, δ)
5. **if** $q_e \neq nil$ **then** Rewire($T, q_e, |T|$)
6. **if** $d(q_e, g) \leq \delta$ and Visible(q_e, g)**then**
7. Add edge $q_e \rightarrow g$ to T
8. $c(g)$ = cost of optimal path from s to g, if g is connected, and ∞ otherwise
9. **return** "no path"

Slide credit K. Hauser

RRT*: RRT + Re-Wiring

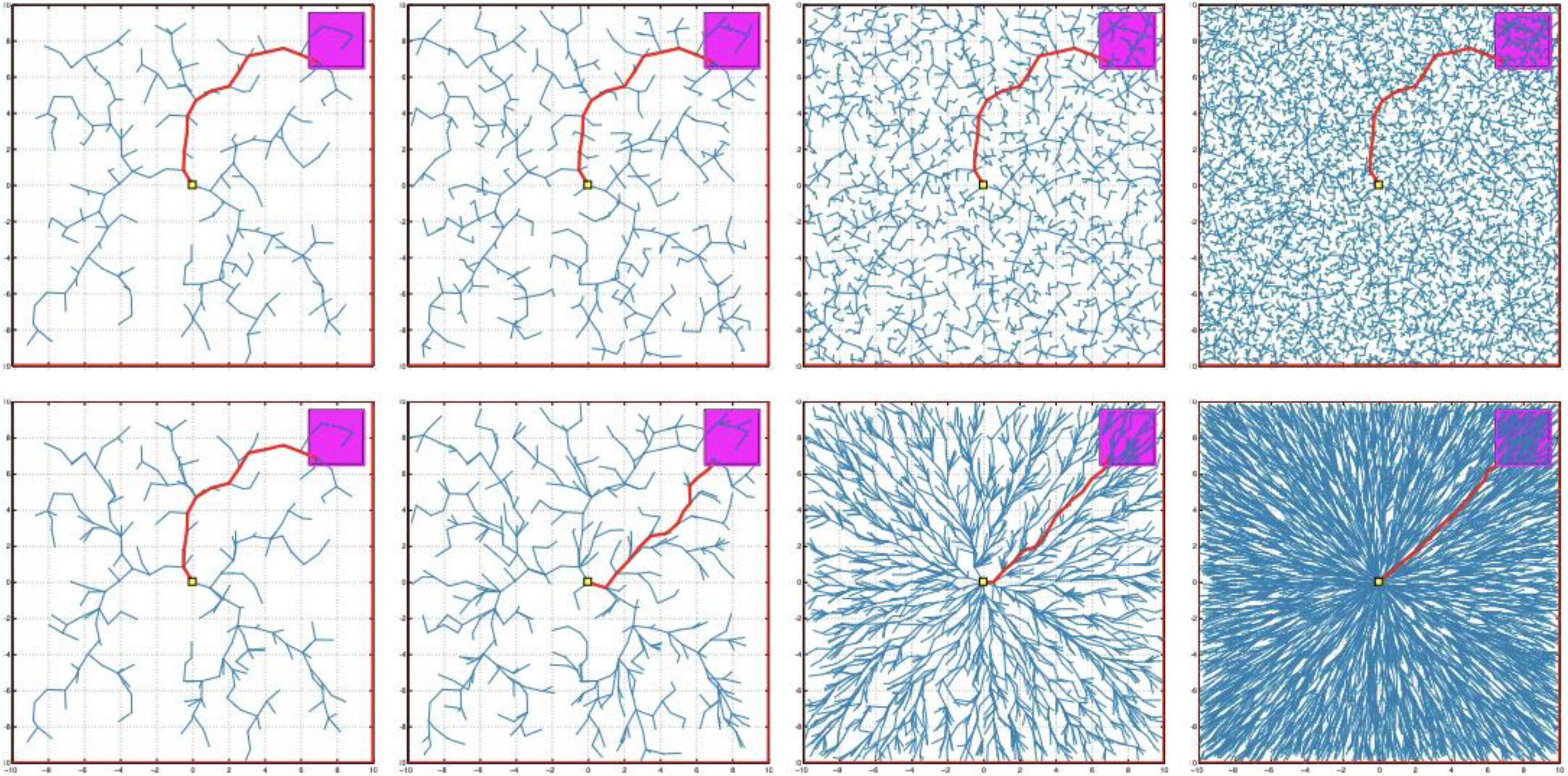
Algorithm Rewire(T, q_{new}, n)

1. Neighbors $\leftarrow$ Set of $k^\star(n)$-nearest neighbors in T, or points in $R^\star(n)$-neighborhood.
2. **for** q $\in$Neighbors sorted by increasing $c(q)$ **do**
3. **if** $c(q_{new}) + d(q_{new}, q) < c(q)$ **then** (optimal path to q passes through q_{new})
4. $c(q) \leftarrow c(q_{new}) + d(q_{new}, q)$
5. Update costs of descendants of q.
6. **if** $c(q) + d(q, q_{new}) < c(q_{new})$ **then** (optimal path to q_{new} passes through q)
7. $c(q_{new}) \leftarrow c(q) + d(q, q_{new})$
8. Set parent of q_{new} to q.
9. Revise costs and parents of descendants of q_{new}.

RRT vs. RRT*

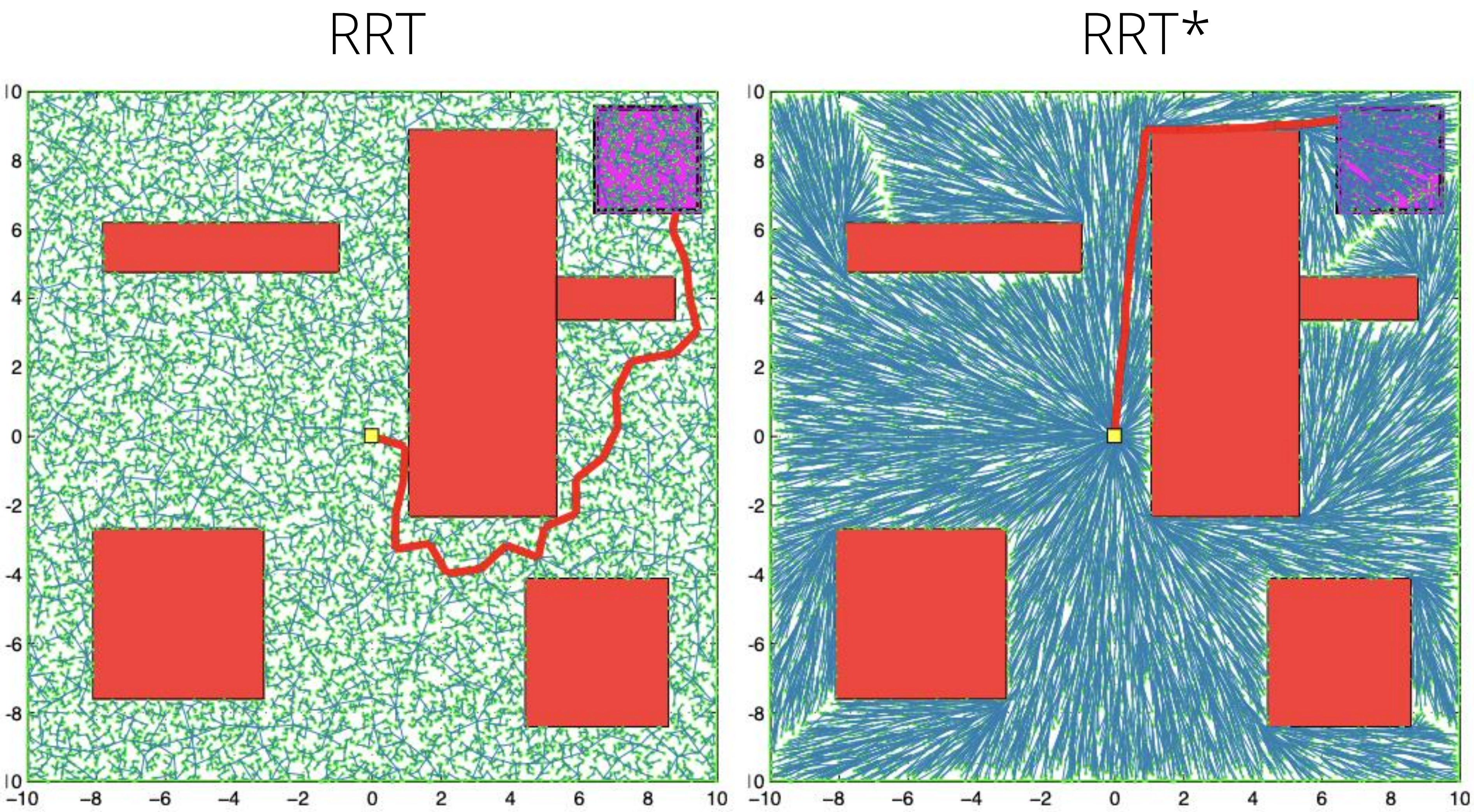
RRT

RRT*



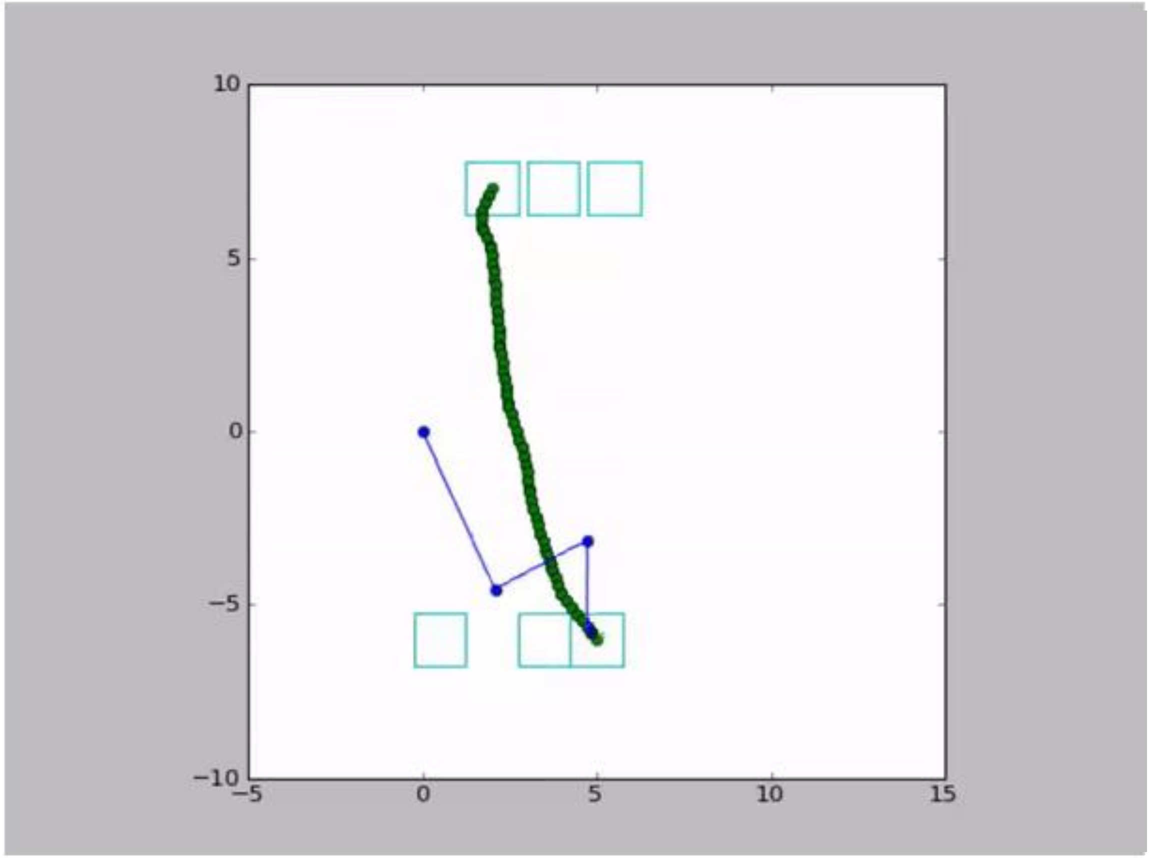
Sampling-based Algorithms for Optimal Motion Planning. S. Karaman and E. Frazzoli.

RRT vs. RRT*



Sampling-based Algorithms for Optimal Motion Planning. S. Karaman and E. Frazzoli.

RRT*



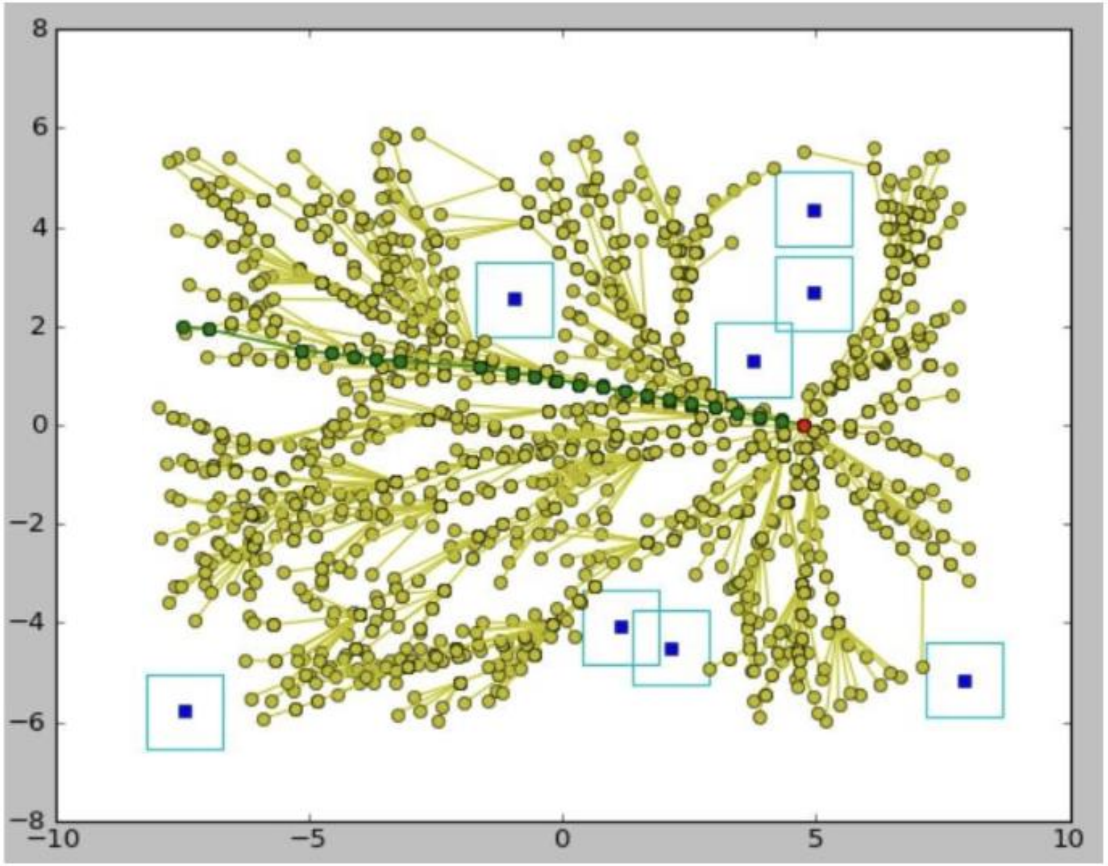


Image credit T. Chinenov

Virtual Potential Fields

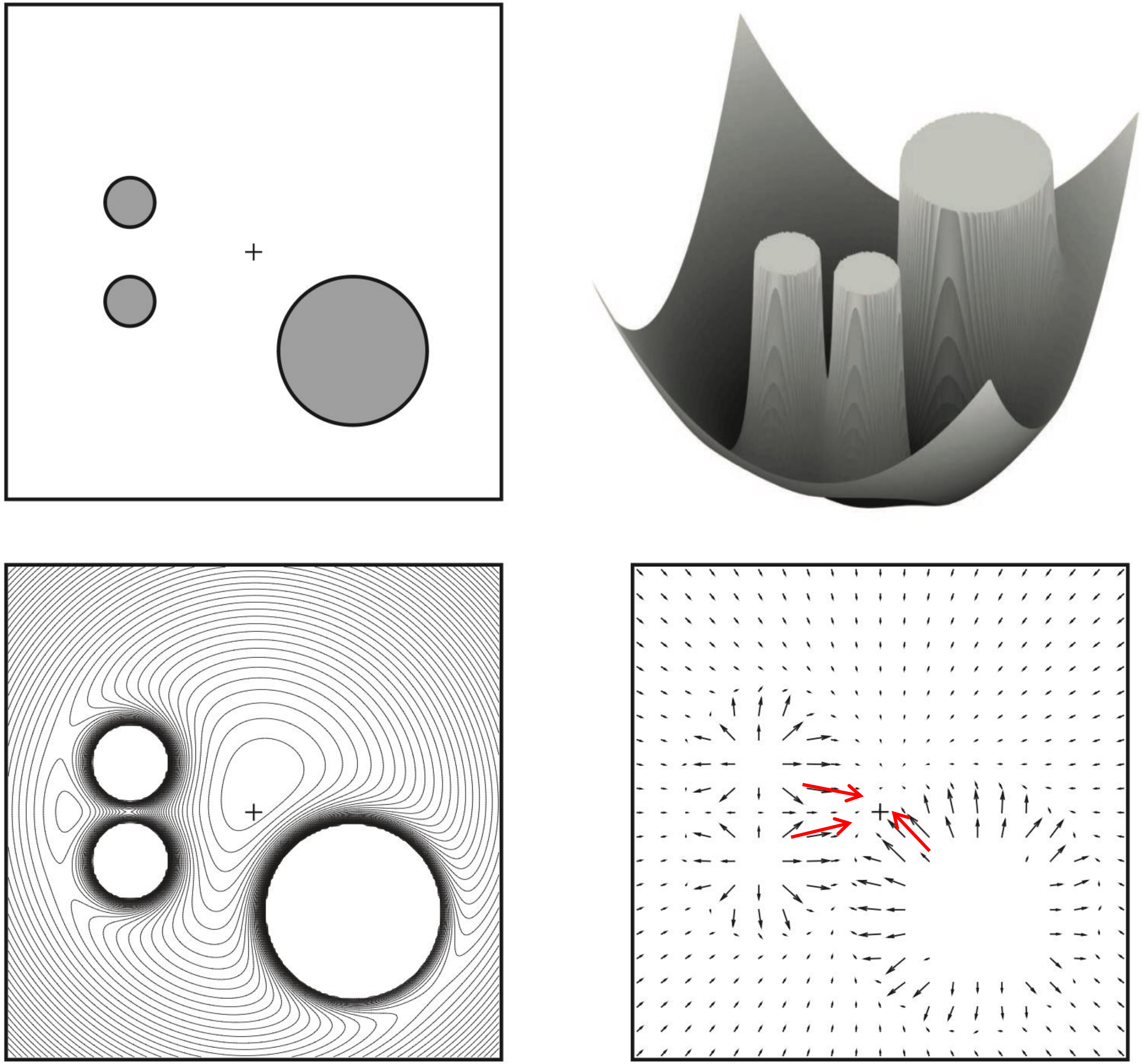
- From physics we know that a potential field $P(q)$ defined over $\boldsymbol{C}$ induces a force $F = -\frac{\partial P}{\partial q}$ that drives an object from high to low potential.
- In robotics, we can define a potential field and derive the corresponding force, with which we drive the configuration $\boldsymbol{q}$ from high to low potential.

- The potential field of reaching a goal:

$$\mathcal{P}_{\text{goal}}(q) = \frac{1}{2}(q - q_{\text{goal}})^{\mathrm{T}} K (q - q_{\text{goal}}), \qquad F_{\text{goal}}(q) = -\frac{\partial \mathcal{P}_{\text{goal}}}{\partial q} = K(q_{\text{goal}} - q);$$

- The potential field induced by a $\boldsymbol{C}$-obstacle $\boldsymbol{\beta}$

$$\mathcal{P}_{\mathcal{B}}(q) = \frac{k}{2d^2(q, \mathcal{B})} \qquad F_{\mathcal{B}}(q) = -\frac{\partial \mathcal{P}_{\mathcal{B}}}{\partial q} = \frac{k}{d^3(q, \mathcal{B})} \frac{\partial d}{\partial q}.$$



Pushing configuration to regions with low potential

$$q_{t+1} = q_t + \frac{\Delta t}{m}\Sigma F$$

Non-Linear Optimization

- We can consider path planning as an optimization problem

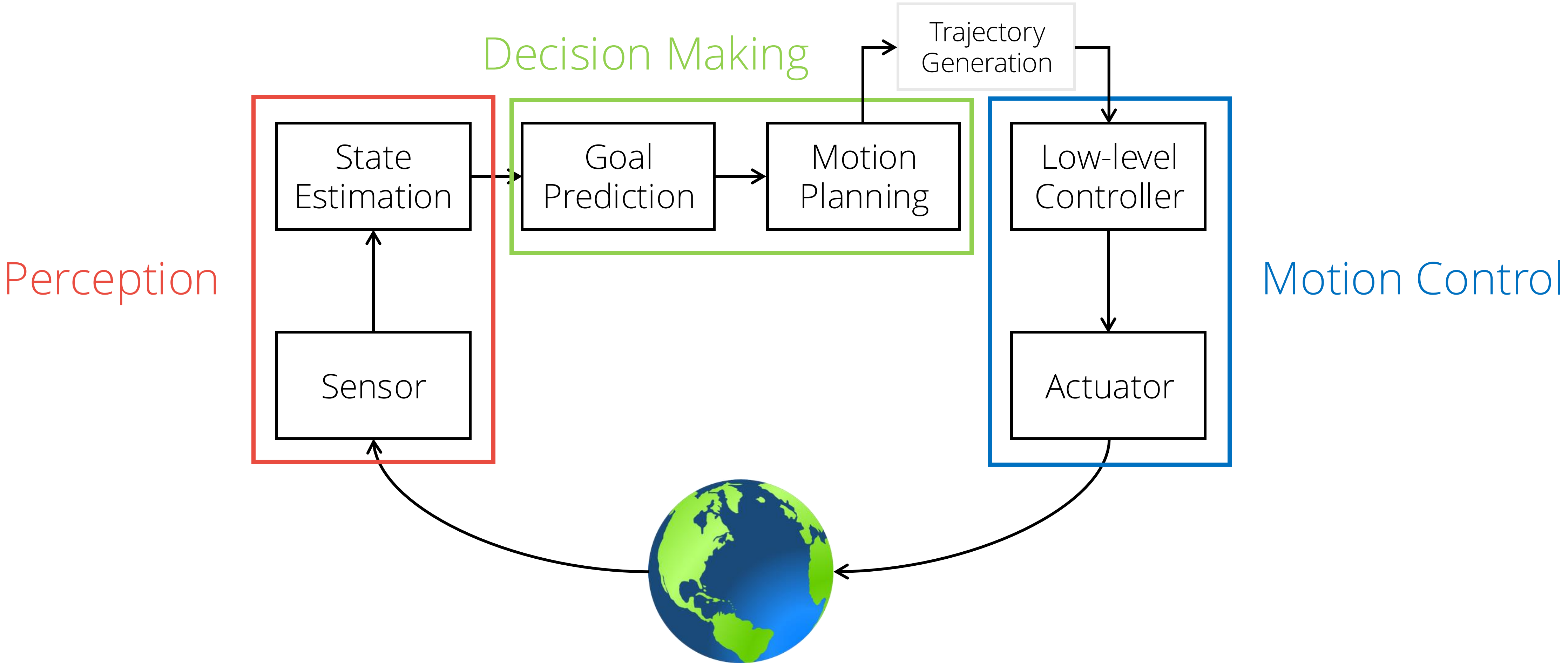
$$\begin{aligned}
&\text{find} && u(t), q(t), T && && (10.6)\\
&\text{minimizing} && J(u(t), q(t), T) && && (10.7)\\
&\text{subject to} && \dot{x}(t) = f(x(t), u(t)), && \forall t \in [0, T], && (10.8)\\
& && u(t) \in \mathcal{U}, && \forall t \in [0, T], && (10.9)\\
& && q(t) \in \mathcal{C}_{\text{free}}, && \forall t \in [0, T], && (10.10)\\
& && x(0) = x_{\text{start}}, && && (10.11)\\
& && x(T) = x_{\text{goal}}. && && (10.12)
\end{aligned}$$

- We can parametrize control $u(t)$ and path $q(t)$ with the coefficients of (1) a polynomial, (2) a truncated Fourier series, (3) spline, (4) wavelet, or (5)piecewise constant acceleration segments in time.

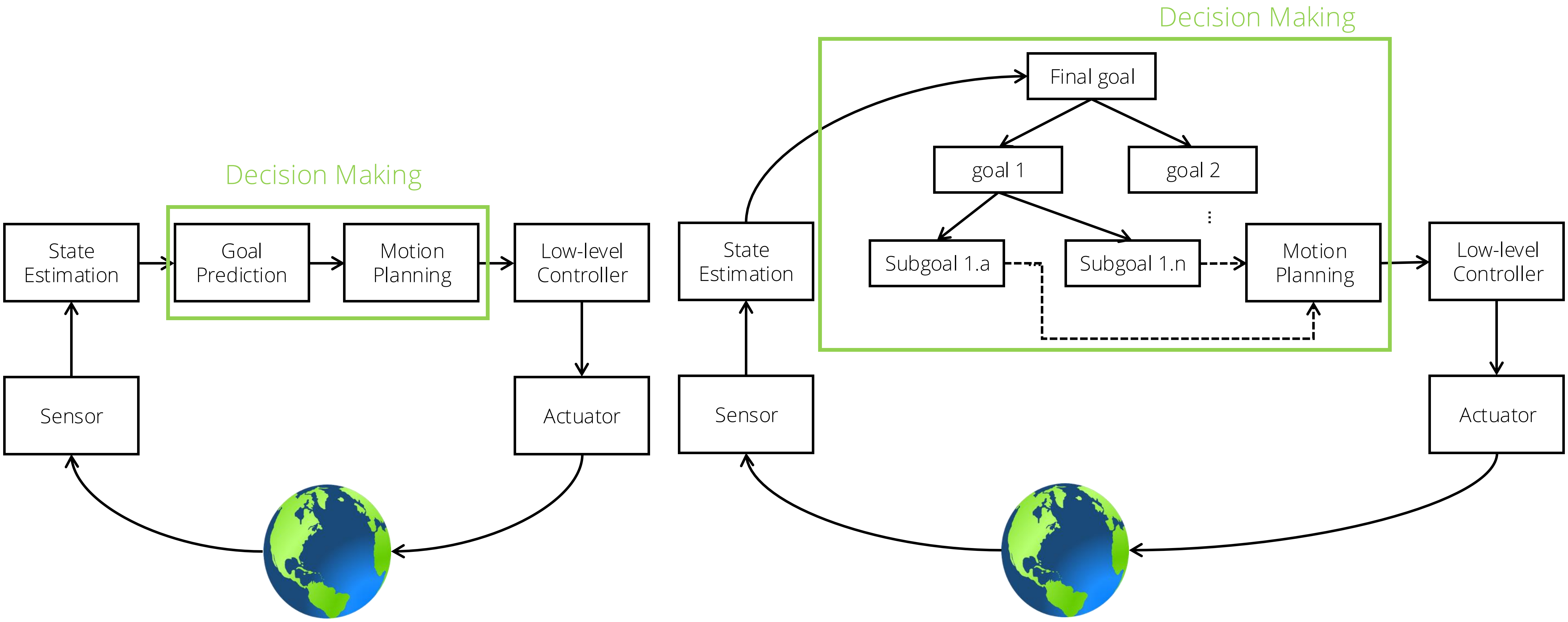
$$\boxed{u_i(t) = \sum_{j=0}^{p} a_j t^j.}$$

For more on Motion Planning, check "Planning Algorithms" by Steven M. LaValle

Action Requires Deciding a Goal, Planning to Achieve the Goal, and Controlling Motion to Follow the Plan

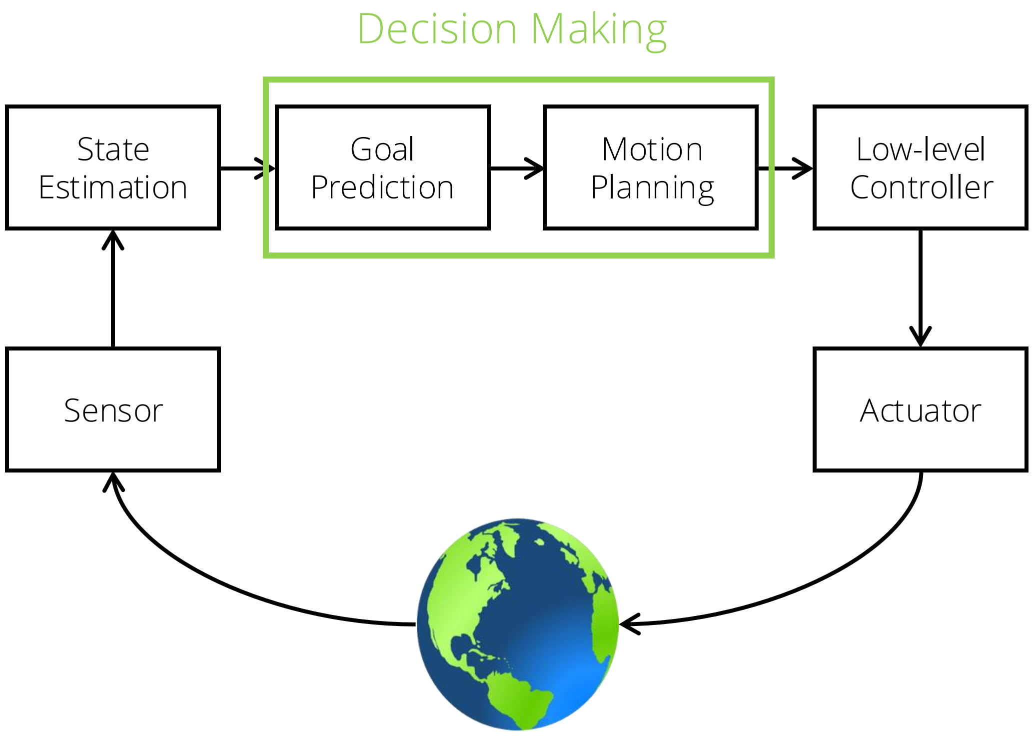


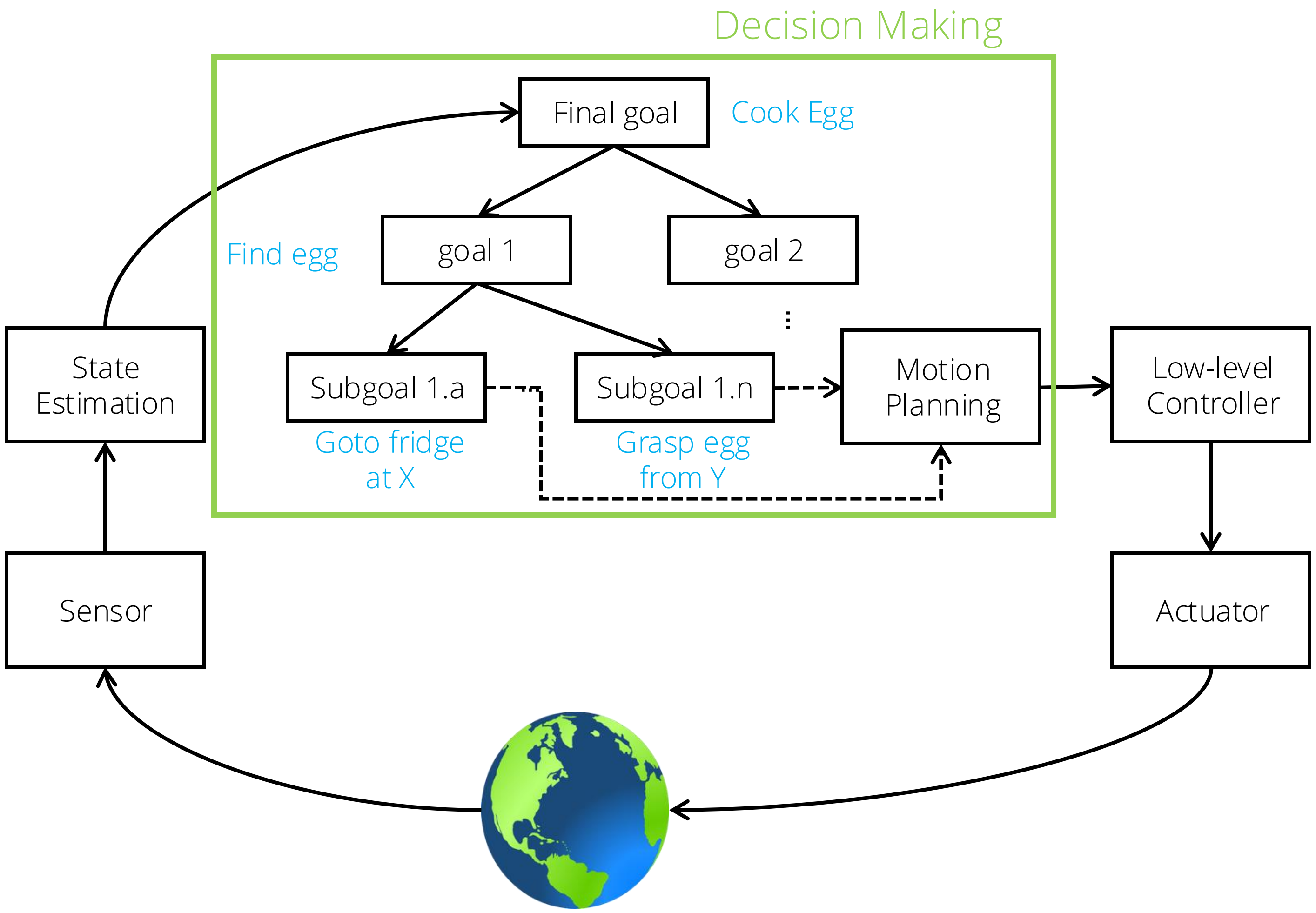
Decision Making is in fact Hierarchical



Oversimplified!

Decision Making is in fact Hierarchical



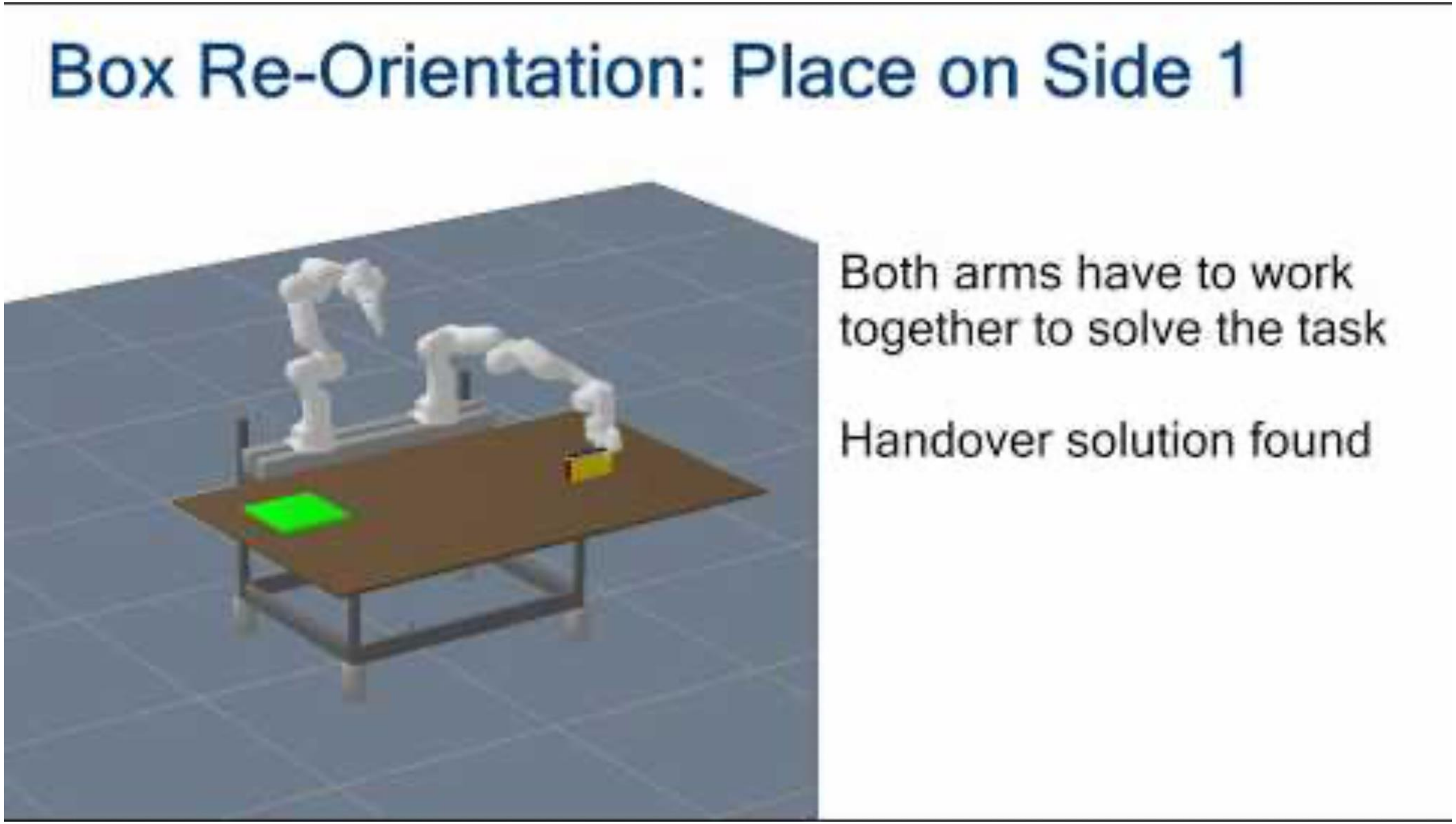


Oversimplified!

Decision Making Should be Adaptive w.r.t the Kinematic and Geometric Feasibility

https://youtu.be/tNHjpXP8RFo?si=Sokgfvgef0T18Tuy

Decision Making Should be Adaptive w.r.t the Kinematic and Geometric Feasibility



https://youtu.be/VndjJtzl7ho?si=Uqc5rLSI2rPLbMIe

We Need to Decide What are the Tasks, the Order of the Tasks, the Goal per Task, the Motion/Path to Achieve the Goal

Task: Put the mustard in the blue region

SubTask1: Grasp the mustard

- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

SubTask2: Lift up the mustard

- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

SubTask3: Carry the mustard above the blue region

- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

SubTask4: Put down the mustard

- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

We Need to Decide What are the Tasks, the Order of the Tasks, the Goal per Task, the Motion/Path to Achieve the Goal

Task: Put the mustard in the blue region

The plan is not feasible, since the robot can't reach the occluded mustard

SubTask1: Grasp the mustard
- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

SubTask2: Lift up the mustard
- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

SubTask3: Carry the mustard above the blue region
- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

SubTask4: Put down the mustard
- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

We Need to Replan Based on Kinematic and Geometric Feasibility

Task: Put the mustard in the blue region

SubTask 1: Grasp the Cheezit

⋮

SubTask N: Grasp the mustard
- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

SubTask N+1: Lift up the mustard
- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

SubTask N+2: Carry the mustard above the blue region
- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

SubTask N+3: Put down the mustard
- Subgoal configuration of the gripper pose
  - ➢ Motion planning for the trajectory

Classical Task Planning: Decide Which Tasks and the Ordering of the Tasks

Task: Put the mustard in the blue region

SubTask 1: Grasp the Cheezit

⋮

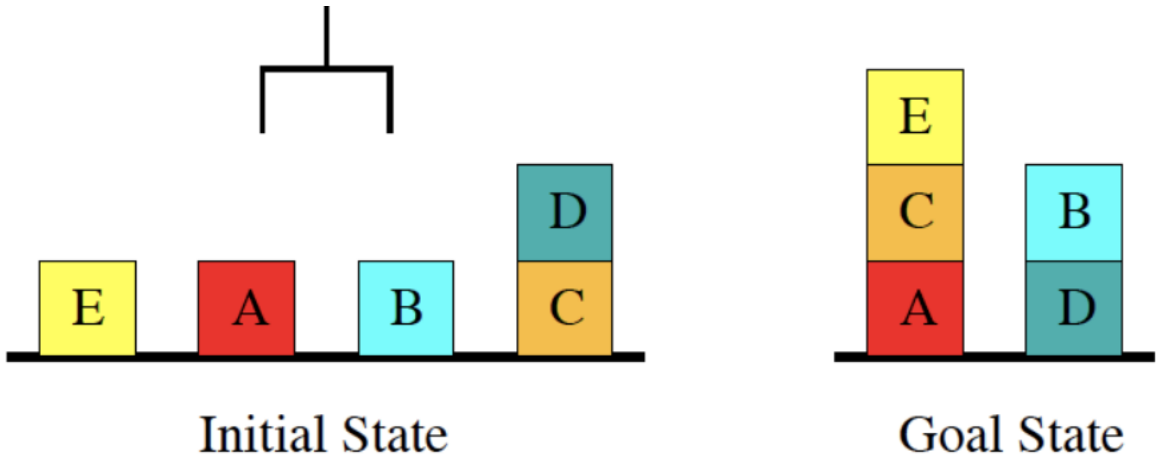
SubTask N: Grasp the mustard
- Precondition
- Effect

SubTask N+1: Lift up the mustard
- Precondition
- Effect

SubTask N+2: Carry the mustard above the blue region
- Precondition
- Effect

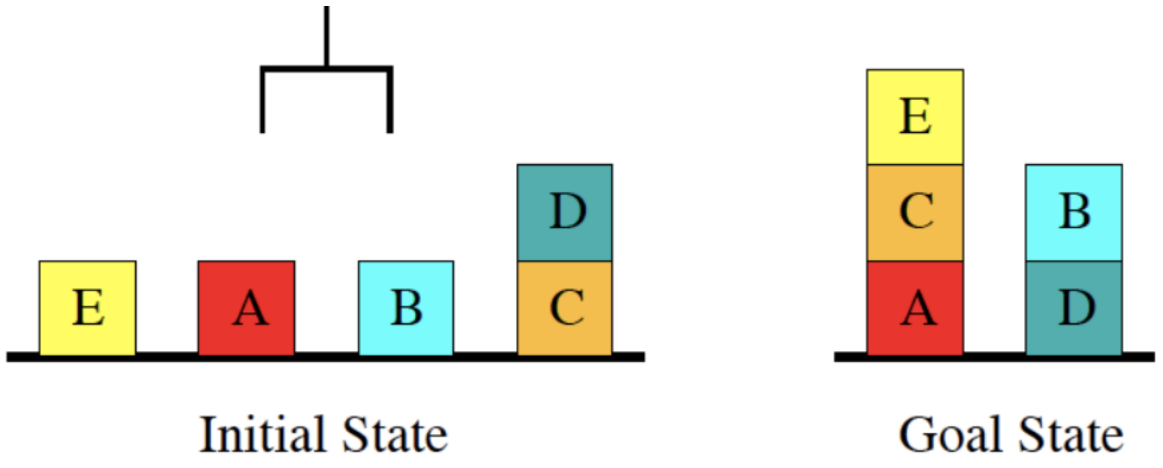
SubTask N+3: Put down the mustard
- Precondition
- Effect ← leads to the final goal (mustard in the blue region)

State Representations of Objects



- The world is abstracted into a <u>discrete</u> space with <u>many variables</u> (e.g. A, B, C, D, E)

State Representations of Objects

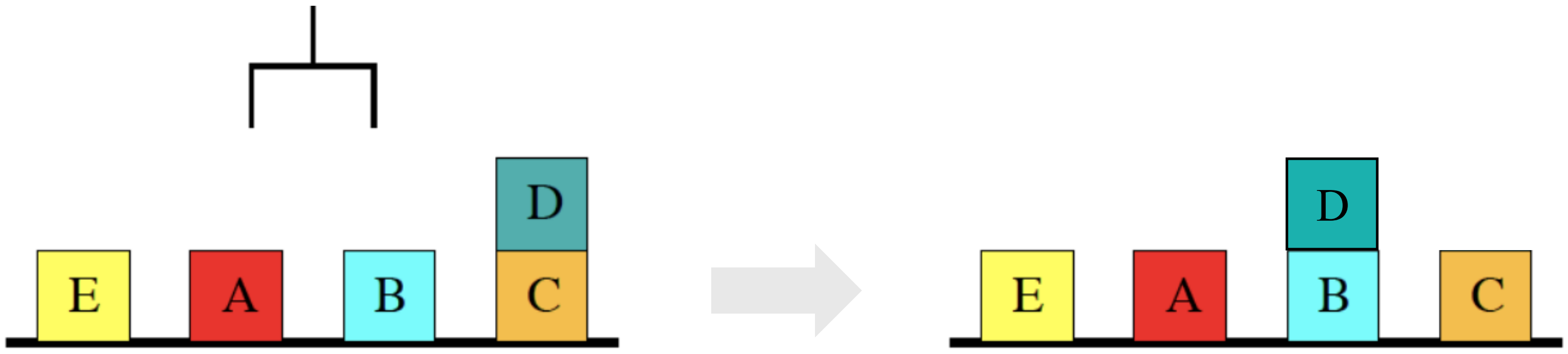


- The world is abstracted into a <u>discrete</u> space with <u>many variables</u> (e.g. A, B, C, D, E)

- State representations of objects:
  - ➢ **Predicate**: Boolean function
  - ➢ **Facts** (literals): instantiated predicates
  - ➢ **States**: set of facts

```
(On ?b1 ?b2)=True/False
(On D C)
{¬(On A B),(On D C), …}
```

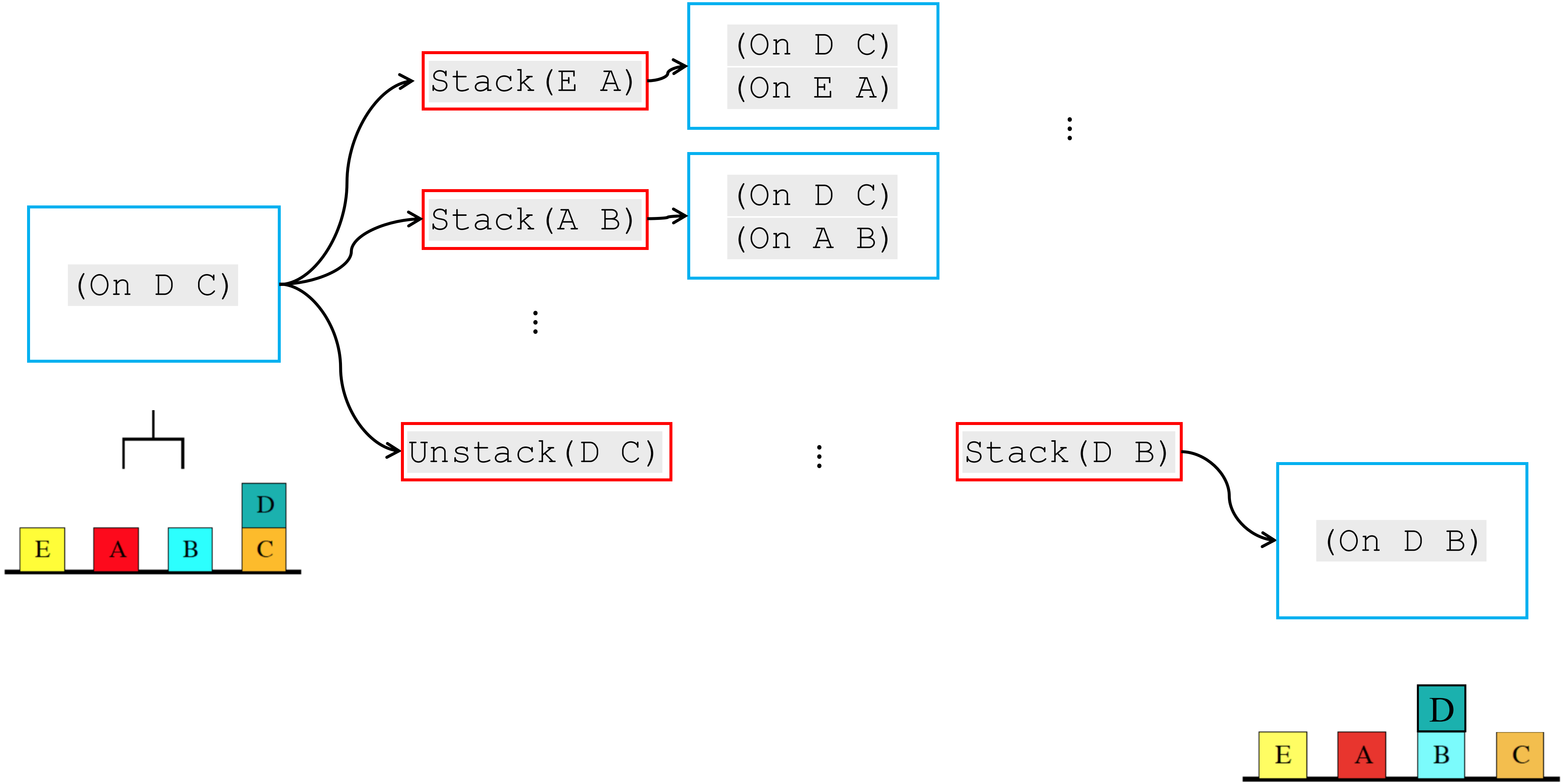
Actions Lead to Transitions between States




```
(:action stack
  :parameters (?b1 ?b2)
  :precondition (and
    (Holding ?b1) (Clear ?b2))
  :effect (and
    (ArmEmpty)
    (On ?b1 ?b2) (Clear ?b1)
    (not (Holding ?b1))
    (not (Clear ?b2))))
```

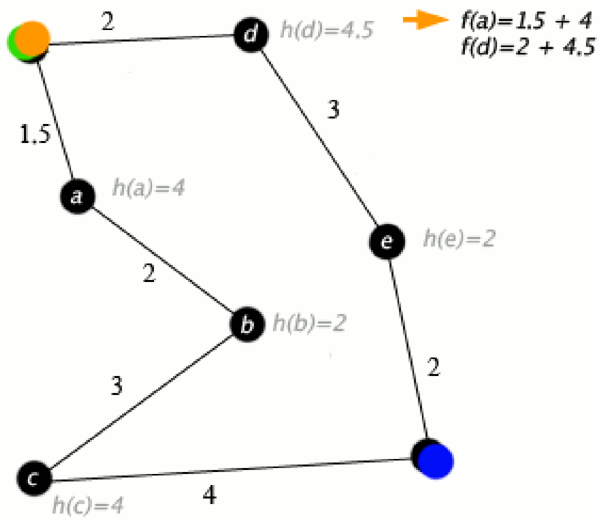
- Action:
  - Preconditions test feasibility of the action
  - Effects describe changes to a set of states
  - Parameters show the set of states involved in the action

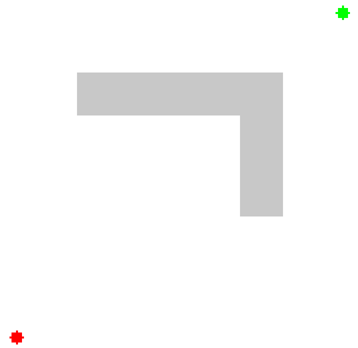
Task Planning: Search a sequence of actions that convert the initial states to the goal states



Forward Best-First Search

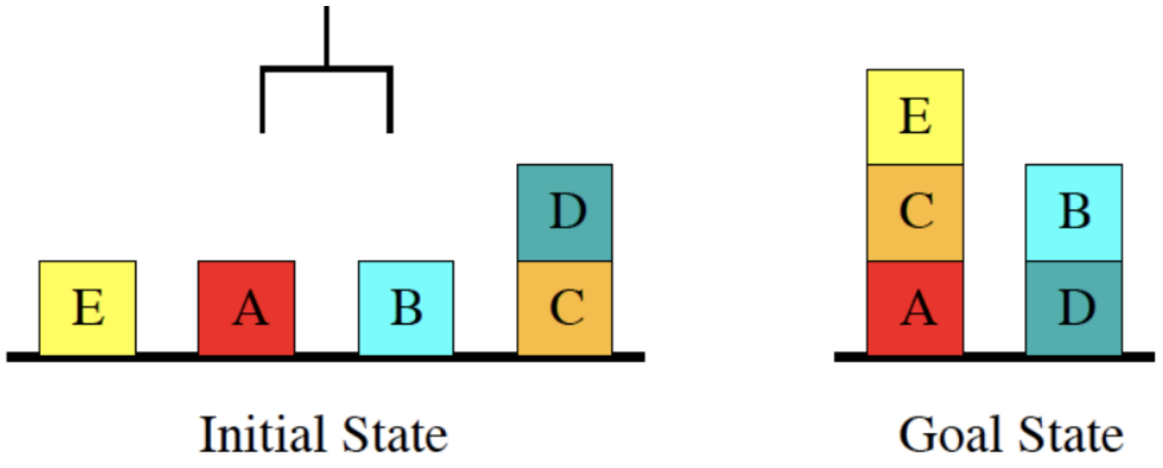
- For a state s
  - **Path cost:** $g(s)$
  - **Heuristic estimate:** $h(s)$ ← How close to the goal
  - Open list **sorted** by priority $f(s)$
- **Weighted A*:** $f(s) = g(s) + wh(s)$
  - Uniform cost search: $w = 0 \implies f(s) = g(s)$
  - A* search: $w = 1 \implies f(s) = g(s) + h(s)$
  - **Greedy** best-first search: $w = \infty \implies f(s) = h(s)$
- How do we estimate $h(s)$?
  - No obvious metric (no metric-space embedding)





Video credit https://en.wikipedia.org/wiki/A*_search_algorithm

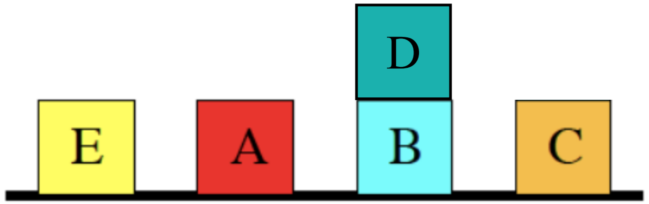
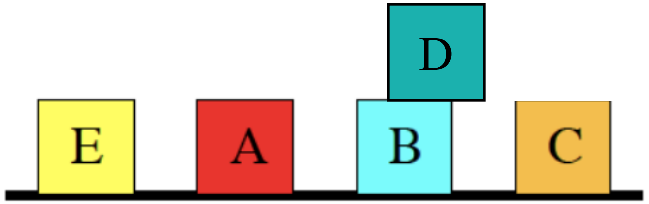
Classical Task Planning



- Initial states: `(On D C)`
- Goal states: `{(On E C),(On C A),(On B D)}`
- Actions:
  1. `Unstack(D, C)`
  2. `Stack(D, B)`
  3. `Stack(C, A)`
  4. `Stack(E, C)`
  5. `Unstack(D, B)`
  6. `Stack(B, D)`

Slide adapted from C. R. Garrett and C. Paxton

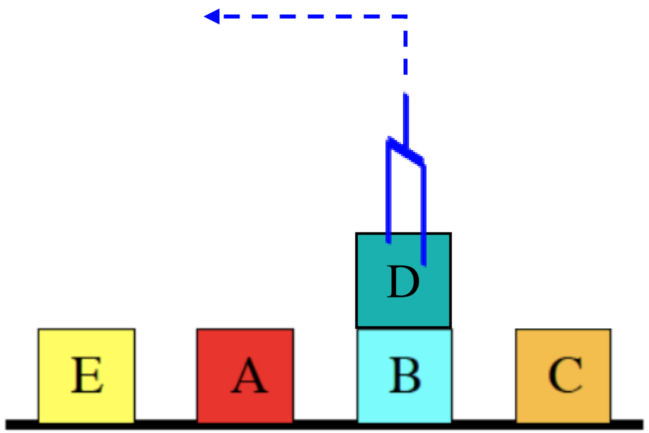
However, Discretized States and Actions Oversimplify the World and Robot-Object Interaction

	Abstract states	Continuous object configurations
	`(On D B)`	(On D B, p_x=0.0, p_y=1.0)
	`(On D B)`	(On D B, p_x=0.2, p_y=1.0)

However, Discretized States and Actions Oversimplify the World and Robot-Object Interaction

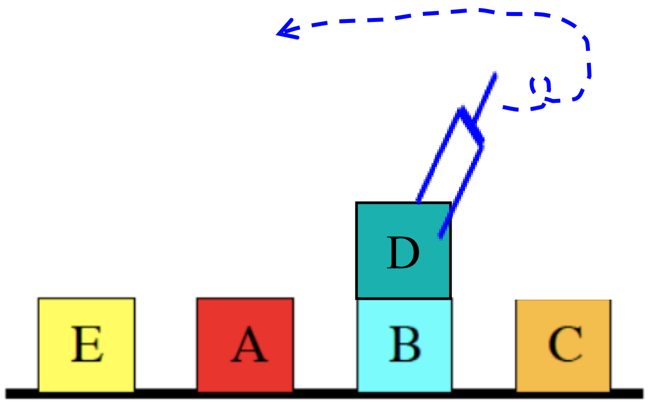
Abstract action

Continuous object configurations



`Unstack(D B)`

```
Unstack(D B)
```
$p_{gripper}$=<0.0, 1.0, 0°>
trajectory=τ



`Unstack(D B)`

```
Unstack(D B)
```
$p_{gripper}$=<0.2, 1.0, 30°>
trajectory=τ

Parameterize States with Continuous Variables

```
(:derived (On ?b1 ?b2)
  (exists (?p1) (?p2) (and (Above ?p2 ?p1)
                           (AtPose ?b1 ?p1)
                           (AtPose ?b2 ?p2))))
```

Parameters:

`?b:` block
`?p:` 6DoF object pose

Static Predicates:

`AtPose:` is block `?b` at pose `?p`
`Above:` is pose `?p1` above pose `?p2`

Parameterize States with Continuous Variables

```
(:action pick
 :parameters (?b ?p ?g ?q)
 :precondition (and (Kin ?b ?p ?g ?q)
                    (AtPose ?b ?p)
                    (Empty)
                    (AtConf ?q))
 :effect (and (Holding ?b ?g)
              (not (AtPose ?b ?p)
              (not (Empty))))
```

Parameters:

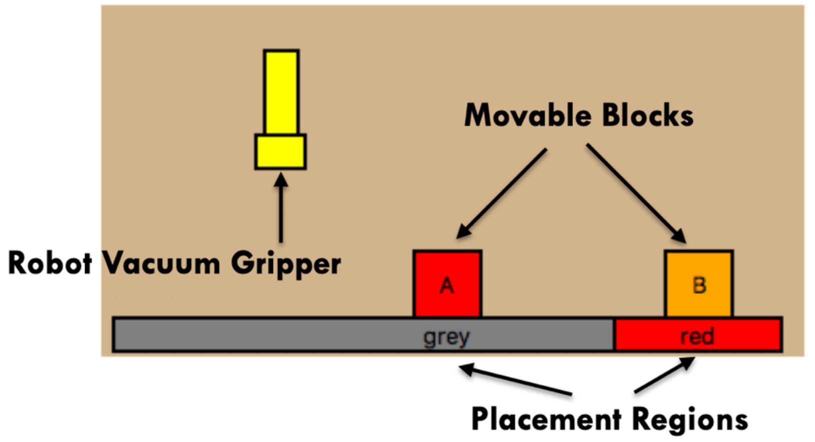
`?b:` block
`?p:` 6DoF object pose
`?g:` 6DoF robot's end-effector pose
`?q:` Robot's configuration

Static Predicates:

`Kin:` Are a grasp `?g` and robot configuration `?q` valid when block `?b` is at pose `?p`
`AtPose:` is block `?b` at pose `?p`
`Empty:` is the robot's end-effector is empty
`Holding:` is block `?b` hold by a grasp `?g`
`AtConf:` is the robot at configuration `?q`

Task and Motion Planning: Plan a sequence of Actions and their Continuous Parameters

Task: Put block A in the red region

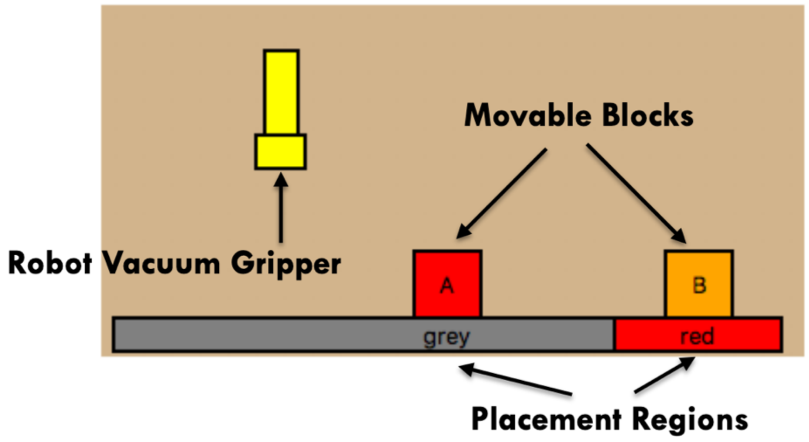


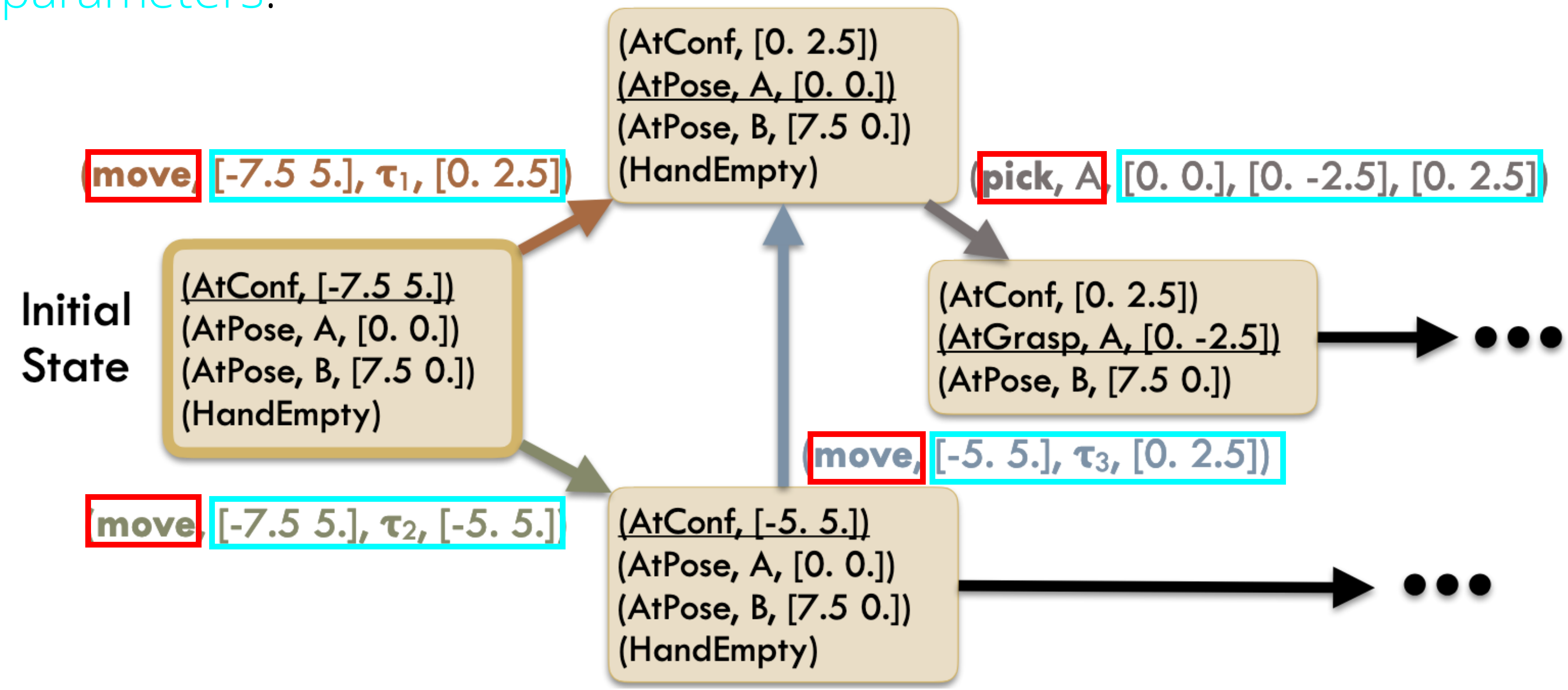
- **Static initial facts** - value is **constant** over time
  - (Block, A), (Block, B), (Region, red), (Region, grey),
- **Fluent initial facts** - value **changes** over time
  - (AtConf, [-7.5 5.]), (HandEmpty),
    (AtPose, A, [0. 0.]), (AtPose, B, [7.5 0.])
- **Goal formula:** `(exists (?p) (and (Contained A ?p red) (AtPose A ?p)))`

Task and Motion Planning: Plan a sequence of Actions and their Continuous Parameters

We need to decide the discrete action class and its continuous parameters!

Task: Put block A in the red region





No a Priori Discretization

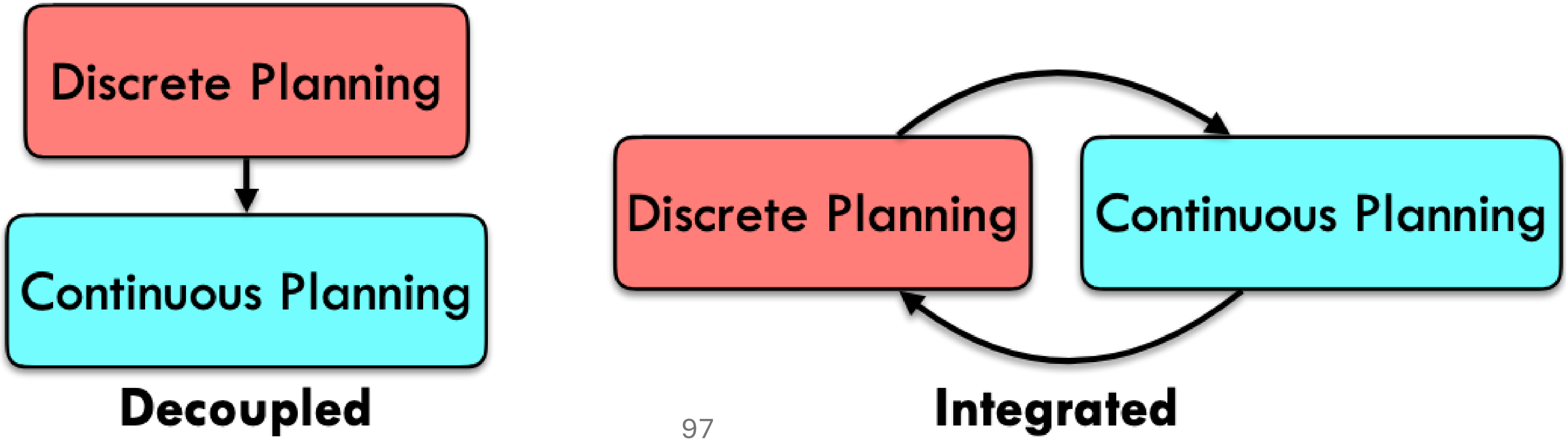
- **Values given at start:**
  - **1 initial configuration:** (Conf, **[-7.5 5.]**)
  - **2 initial poses:** (Pose, A, **[0. 0.]**), (Pose, B, **[7.5 0.]**)
  - **2 grasps:** (Grasp, A, **[0. -2.5]**), (Grasp, B, **[0. -2.5]**)
- **Planner needs to find:**
  - Action classes: pick, move, place …
  - **1 pose within a region:** `(Contain A ?p red)`
  - **1 collision-free pose:** `(CFree A ?p ? B ?p2)`
  - **4 grasping configurations:** `(Kin ?b ?p ?g ?q)`
  - **4 robot trajectories:** `(Motion ?q1 ?t ?q2)`

Slide credit C. R. Garrett

Decoupled vs Integrated TAMP

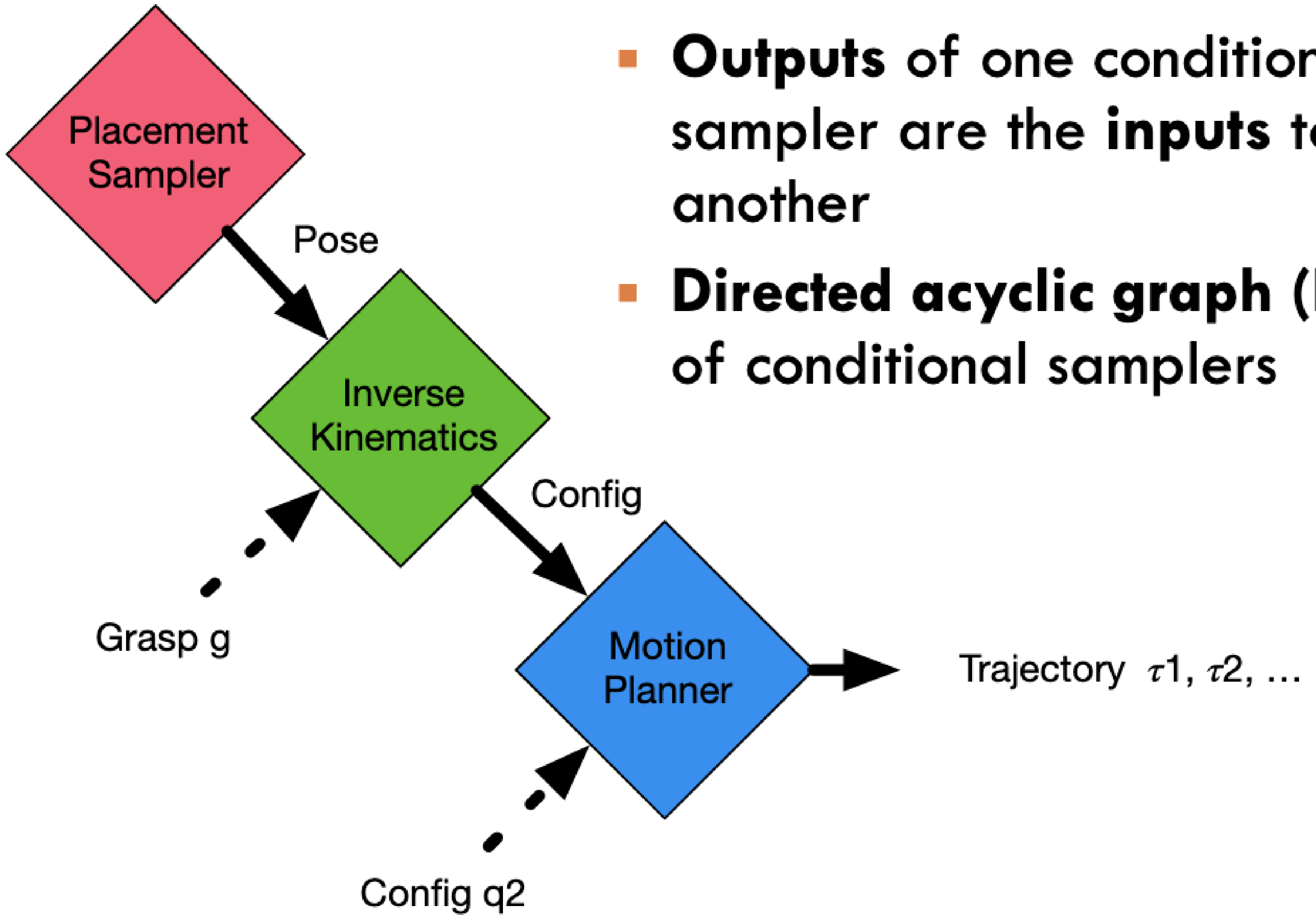
36

- **Decoupled:** discrete (task) planning **then** continuous (motion) planning
- Requires a strong **downward refinement** assumption
  - **<u>Every</u>** correct discrete plan can be **refined** into a correct continuous plan (from hierarchal planning)
- **Integrated:** **<u>simultaneous</u>** discrete & continuous planning



Slide credit C. R. Garrett

Obtain Continuous Action Parameters by Sampling

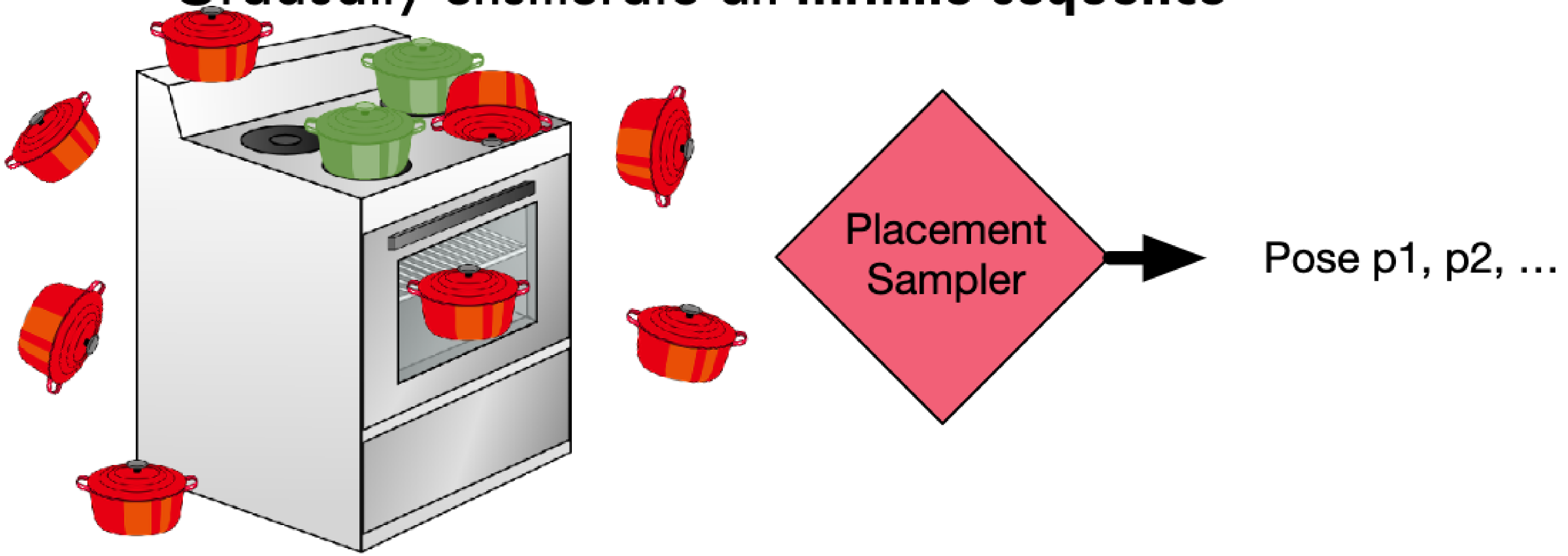


- **Outputs** of one conditional sampler are the **inputs** to another
- **Directed acyclic graph (DAG)** of conditional samplers

Slide credit C. R. Garrett

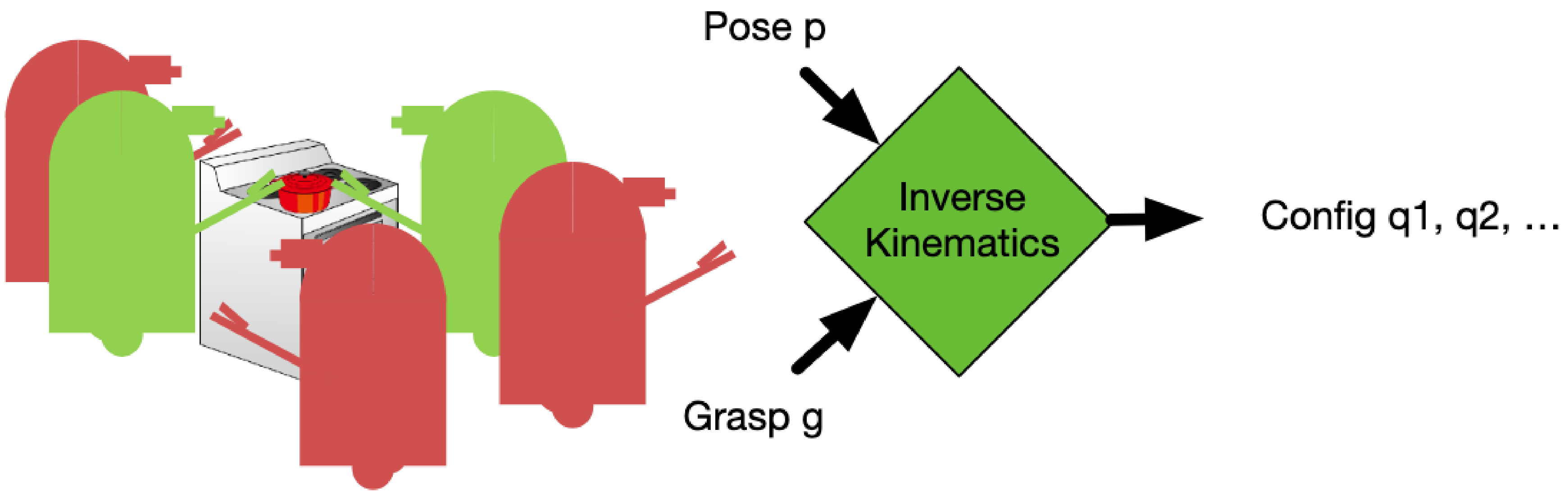
What Samplers Do We Need?

- **Low-dimensional** placement stability constraint (`Contain`)
  - *i.e.* 1D manifold embedded in 2D pose space
- Directly **sample values that satisfy the constraint**
- May need **arbitrarily many** samples
  - Gradually enumerate an **infinite sequence**



Intersection of Constraints

- **Kinematic constraint** (`Kin`) involves poses, grasps, and configurations
- **Conditional samplers** - samplers with inputs



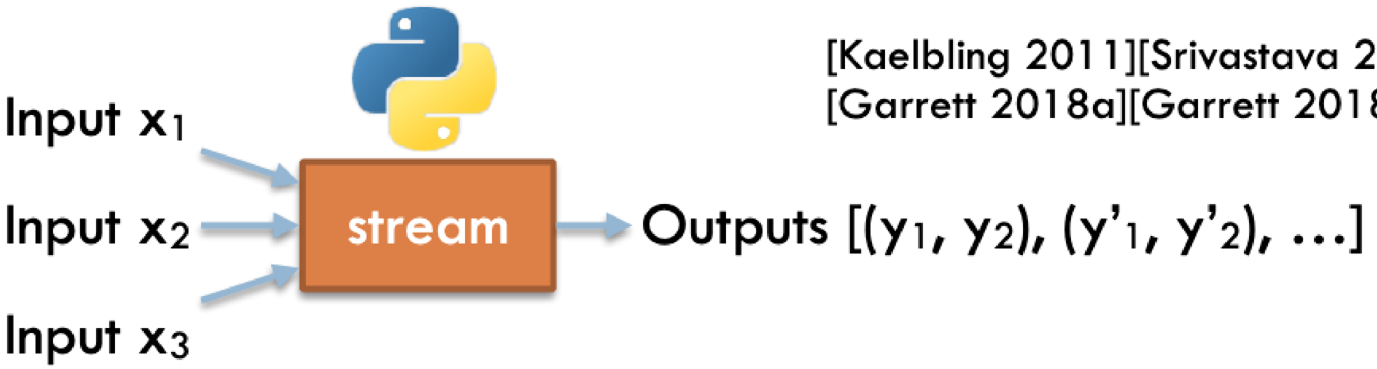
Stream: a function to a generator

- **Advantages**
  - Programmatic implementation
  - Compositional
  - Supports infinite sequences

```python
def stream(x1, x2, x3):
    i = 0
    while True:
        y1 = i*(x1 + x2)
        y2 = i*(x2 + x3)
        yield (y1, y2)
        i += 1
```

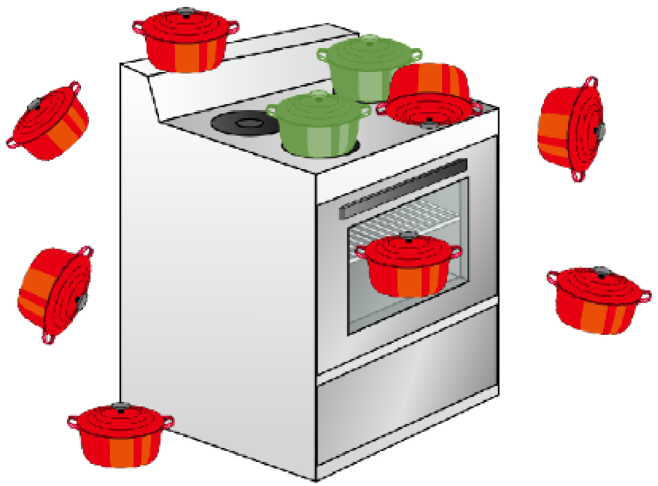
- **Stream** - function from an **input object tuple** (x_1, x_2, x_3) to a (potentially infinite) sequence of **output object tuples** [(y_1, y_2), (y'_1, y'_2), …]

[Kaelbling 2011][Srivastava 2014]
[Garrett 2018a][Garrett 2018b]

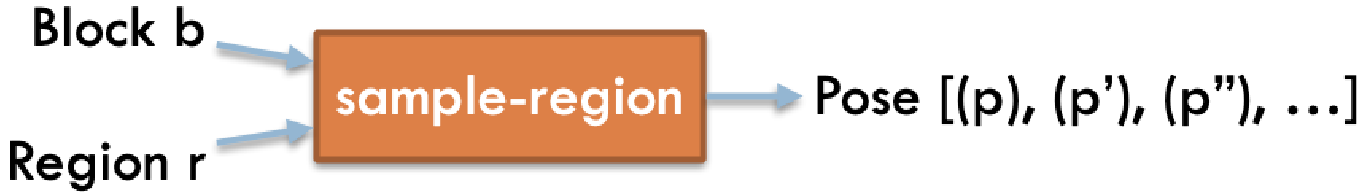


Sampling Contained Poses


```
(:stream sample-region
  :inputs (?b ?r)
  :domain (and (Block ?b) (Region ?r))
  :outputs (?p)
  :certified (and (Pose ?b ?p) (Contain ?b ?p ?r)))
```



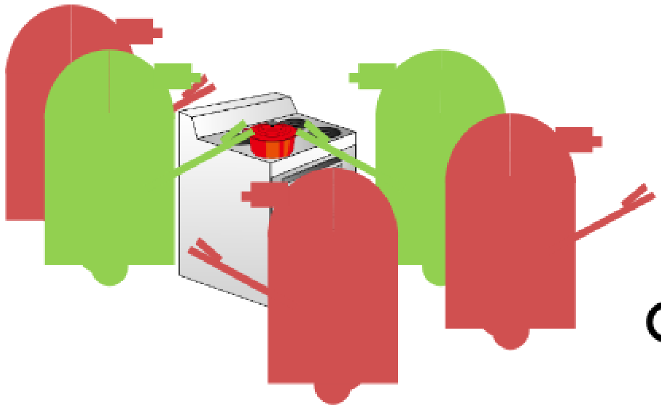
```python
def sample_region(b, r):
    x_min, x_max = REGIONS[r]
    w = BLOCKS[b].width
    while True:
        x = random.uniform(x_min + w/2,
                           x_max - w/2)
        p = np.array([x, 0.])
        yield (p,)
```

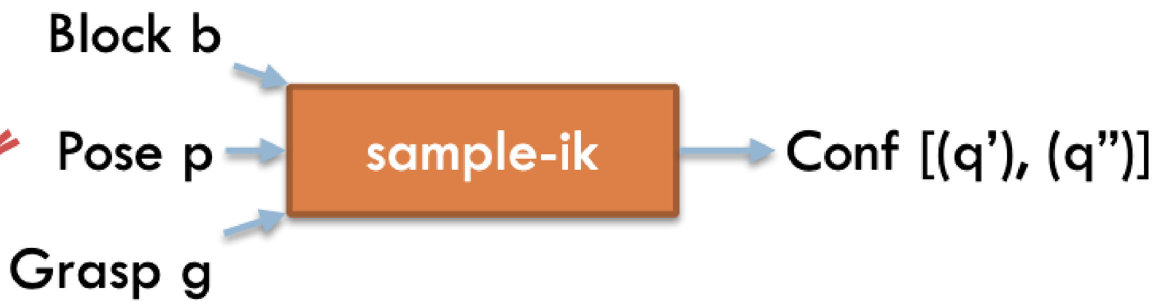


Sampling IK Solutions

- **Inverse kinematics** (IK) to produce robot grasping configuration
  - Trivial in 2D, non-trial in general (e.g. 7 DOF arm)

```
(:stream sample-ik
  :inputs (?b ?p ?g)
  :domain (and (Pose ?b ?p) (Grasp ?b ?g))
  :outputs (?q)
  :certified (and (Conf ?q) (Kin ?b ?p ?g ?q)))
```





Calling a Motion Planner

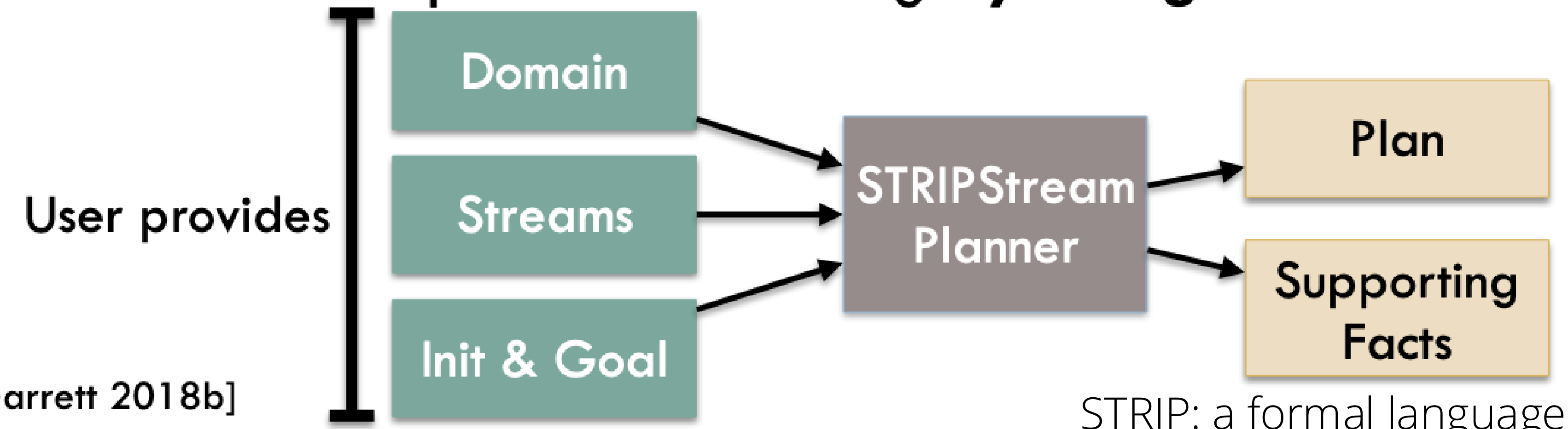
74

- "Sample" (e.g. via a PRM) **multi-waypoint trajectories**
- Include **joint limits & fixed obstacle collisions,** but not movable object collisions

```
(:stream sample-motion
   :inputs (?q1 ?q2)
   :domain (and (Conf ?q1) (Conf ?q2))
   :outputs (?t)
   :certified (and (Traj ?t) (Motion ?q1 ?t ?q2)))
```

STRIPStream = STRIPS + Streams

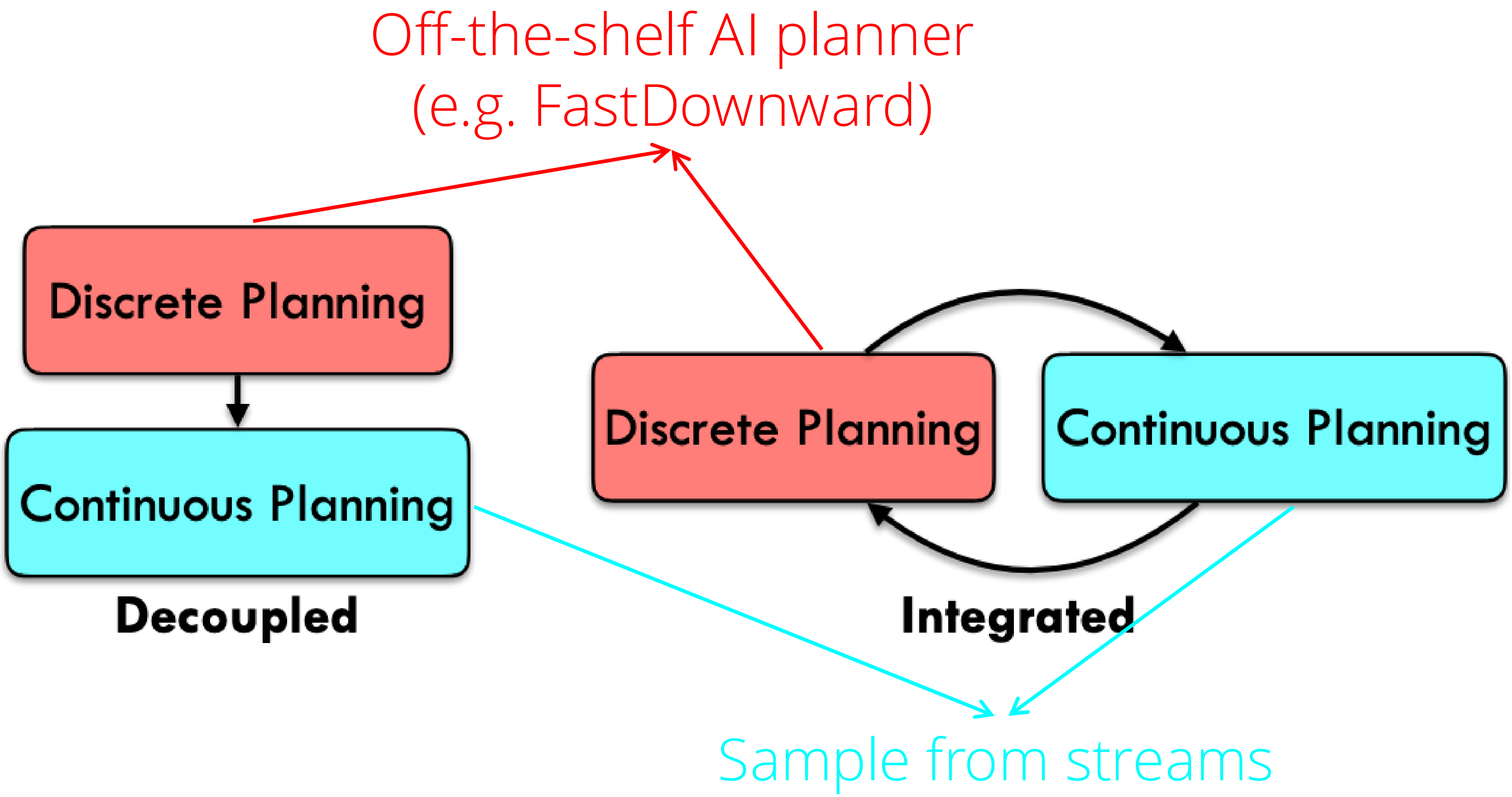
- **Domain dynamics** *(domain.pddl): declares* actions
- **Stream properties** (*stream.pddl*)
  - Declares stream inputs, outputs, and certified facts
- **Problem** and **stream implementation** (*problem.py*)
  - Initial state, **Python constants,** & goal formula
  - Stream implementation using **Python generators**



[Garrett 2018b]

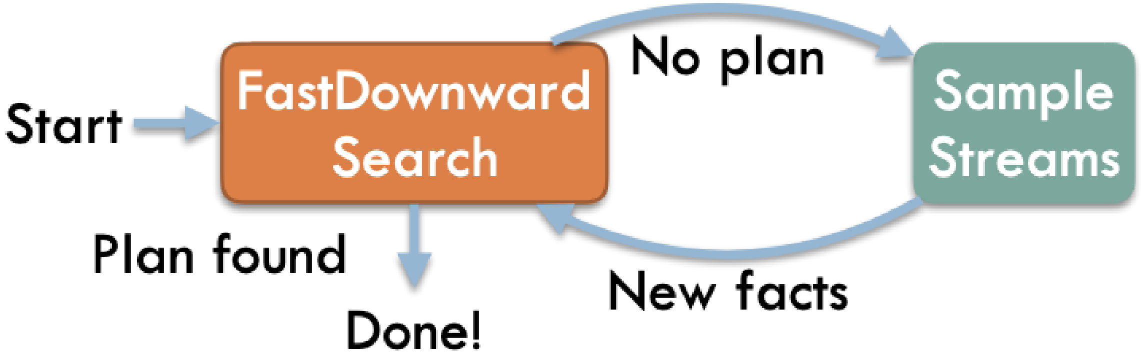
STRIP: a formal language for expressing planning problems

Obtain Continuous Action Parameters by Sampling



Incremental Algorithm

- Incrementally construct all possible initial facts
- Periodically check if a solution exists
- Repeat:
  1. **Compose** and **evaluate** a finite number of streams to unveil more facts in the initial state
  2. **Search** the current PDDL problem for plan
  3. **Terminate** when a plan is found



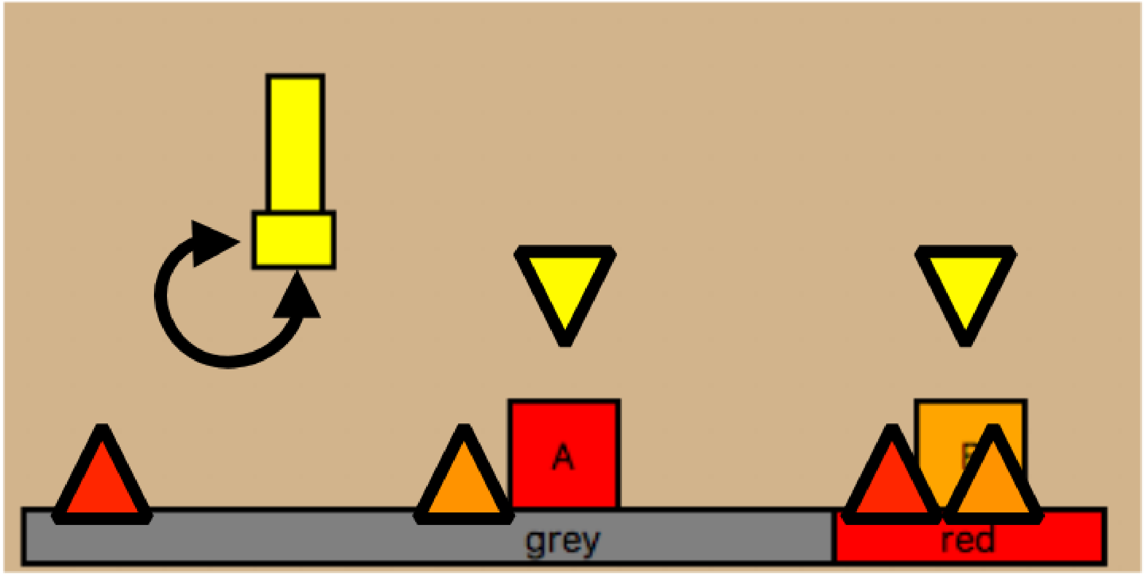
[Garrett 2018a]
[Garrett 2018b]

Slide credit C. R. Garrett

Incremental: Sampling Iteration 1

Iteration 1 - 14 stream evaluations

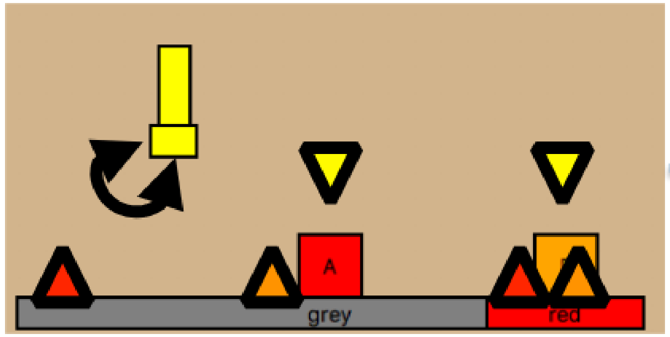
- **Sampled:**
  - 2 new robot configurations:
  - 4 new block poses:
  - 2 new trajectories:



Slide credit C. R. Garrett

Incremental: Search Iteration 1

- Pass current discretization to FastDownward
- If **infeasible**, the current set of samples is insufficient

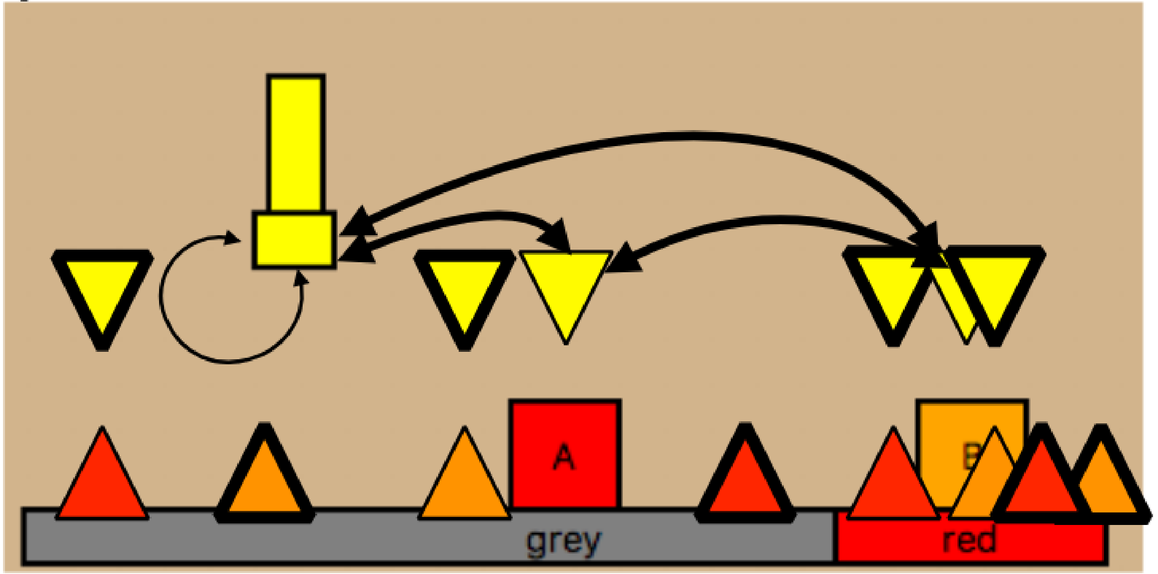


FastDownward Search → Infeasible!

Incremental: Sampling Iteration 2

Iteration 2 - 54 stream evaluations

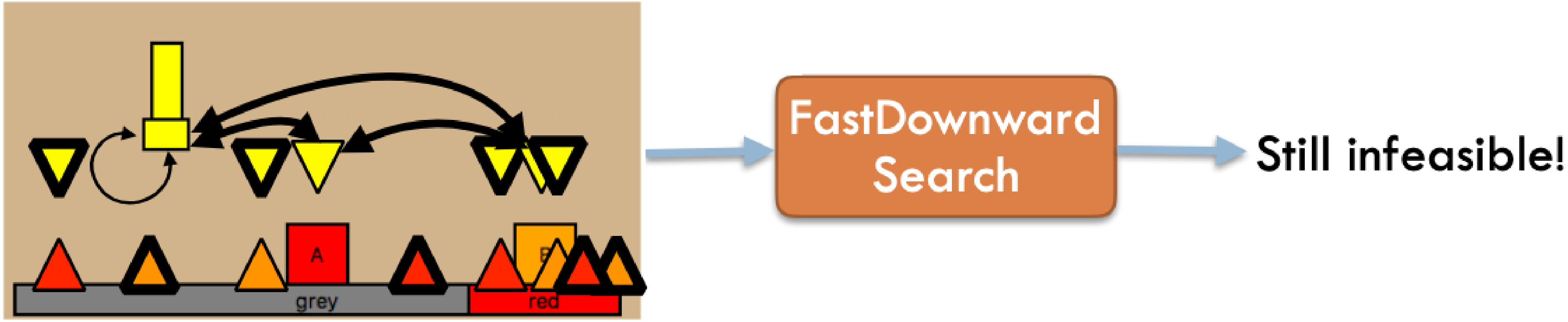
- **Sampled:**
  - 4 new robot configurations:
  - 4 new block poses:
  - 10 new trajectories:



Slide credit C. R. Garrett

Incremental: Search Iteration 2

- Pass current discretization to FastDownward
- If **infeasible,** the current set of samples is insufficient



Incremental Example: Iterations 3-4

Iteration 3 - 118 stream evaluations
Iteration 4 - 182 stream evaluations

Solution:
1) **move** [-7.5 5.] [[-7.5 5.], [-7.5 5.], [7.5 5.], [7.5 2.5]] [7.5 2.5]
2) **pick B** [7.5 0.] [0. -2.5] [7.5 2.5]
3) **move** [7.5 2.5] [[7.5 2.5], [7.5 5.], [10.97 5.], [10.97 2.5]] [10.97 2.5]
4) **place B** [10.97 0.] [0. -2.5] [10.97 2.5]
5) **move** [10.97 2.5] [[10.97 2.5], [10.97 5.], [0. 5.], [0. 2.5]] [0. 2.5]
6) **pick A** [0. 0.] [0. -2.5] [0. 2.5]
7) **move** [0. 2.5] [[0. 2.5], [0. 5.], [7.65 5.], [7.65 2.5]] [7.65 2.5]
8) **place A** [7.65 0.] [0. -2.5] [7.65 2.5]

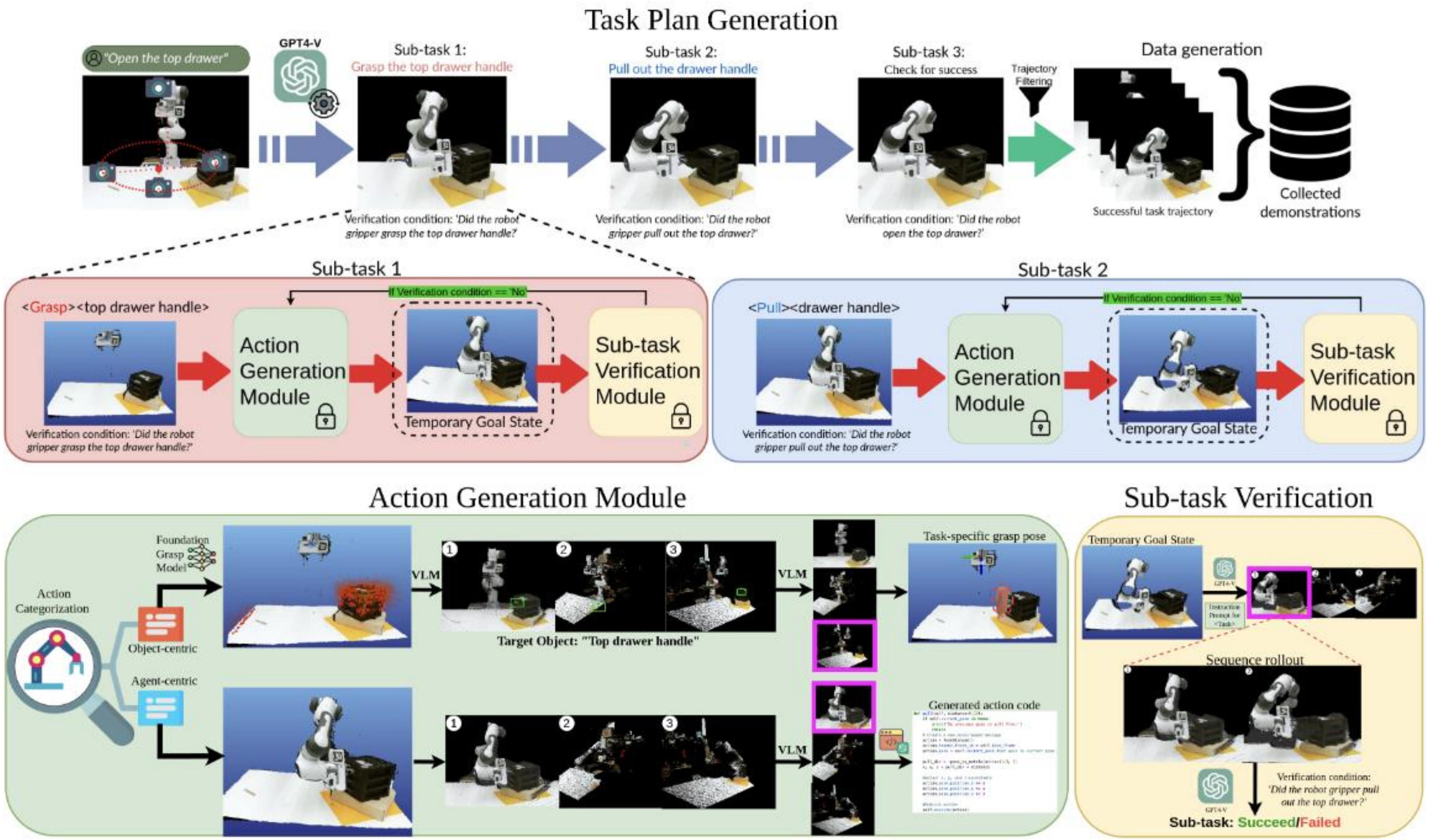
- **Drawback** - many unnecessary samples produced
  - **Computationally expensive** to generate
  - Induces **large discrete-planning problems**

What are the Assumptions / Limitations?

- Limitations:
  - TAMP needs to hand craft samplers for (sub)goal configurations. How to generalize to novel objects / scene.
  - TAMP needs to pre-define action classes. How to generalize to unseen tasks?
  - TAMP assumes deterministic actions, which produce the same intended effect all the time
  - TAMP assumes perfect perception. Robots know the perfect object states.
  - TAMP has heavy computational overhead.

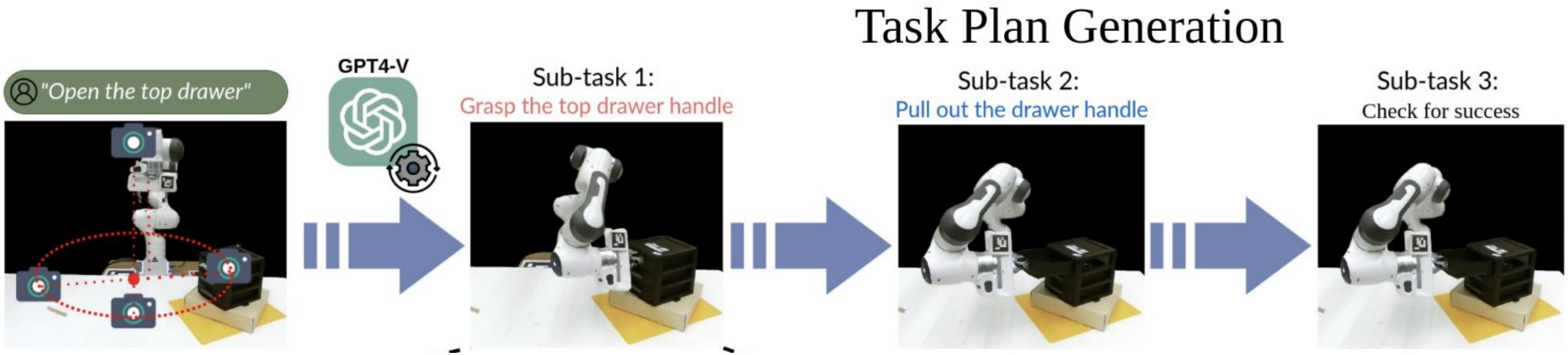
- What are the modern ways to do TAMP?

Automate TAMP with VLM



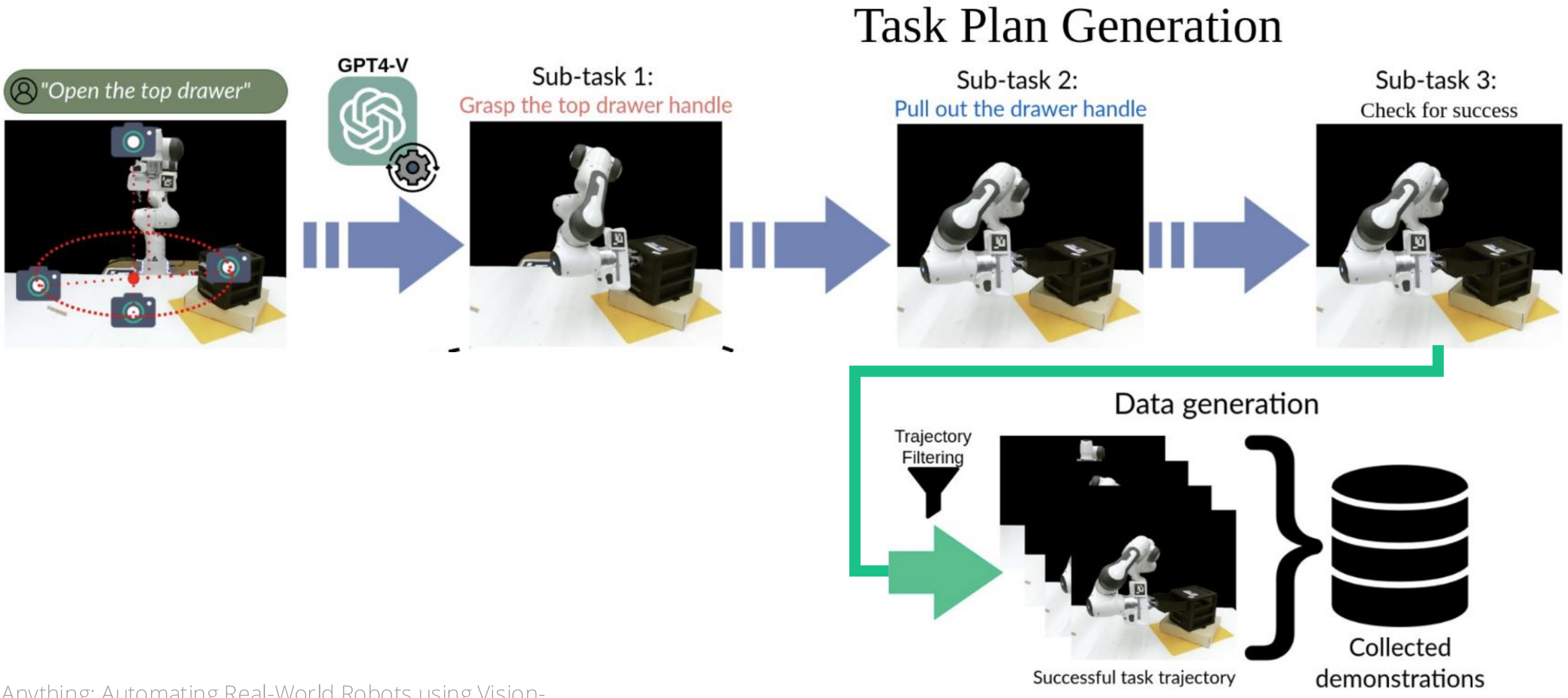
Manipulate Anything: Automating Real-World Robots using Vision-Language Models. Duan et al.

LLMs that Perceive, Plan Subtasks, Determine the Motion, Monitor the Progress



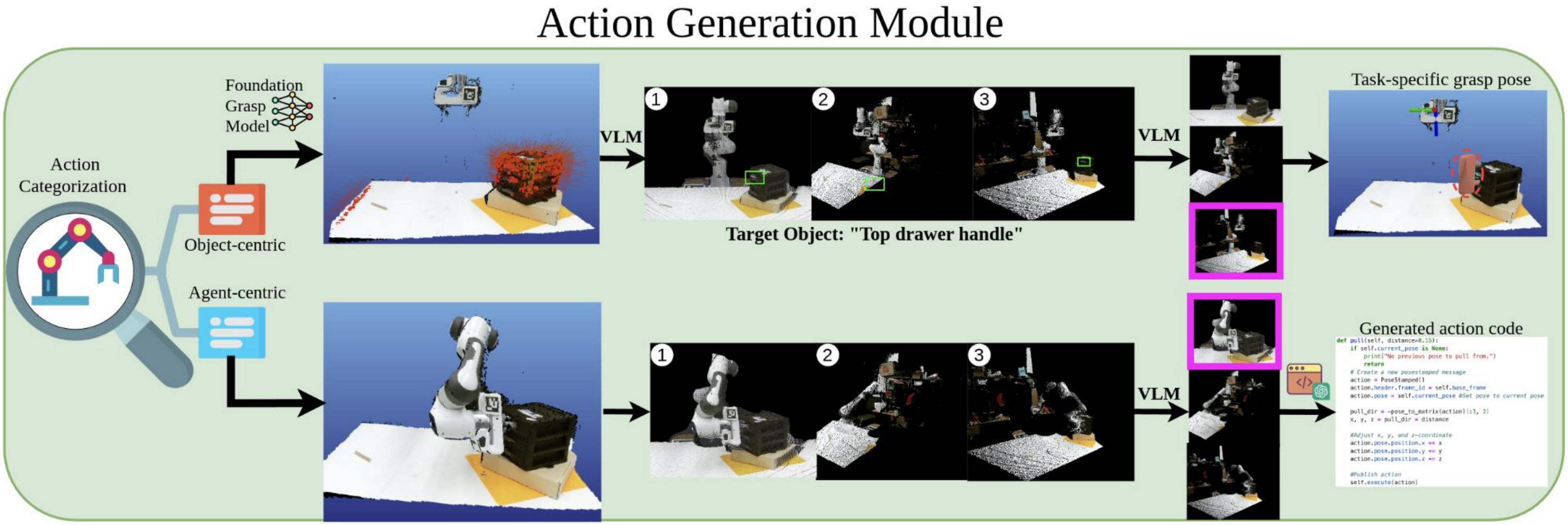
Manipulate Anything: Automating Real-World Robots using Vision-Language Models. Duan et al.

LLMs that Perceive, Plan Subtasks, Determine the Motion, Monitor the Progress



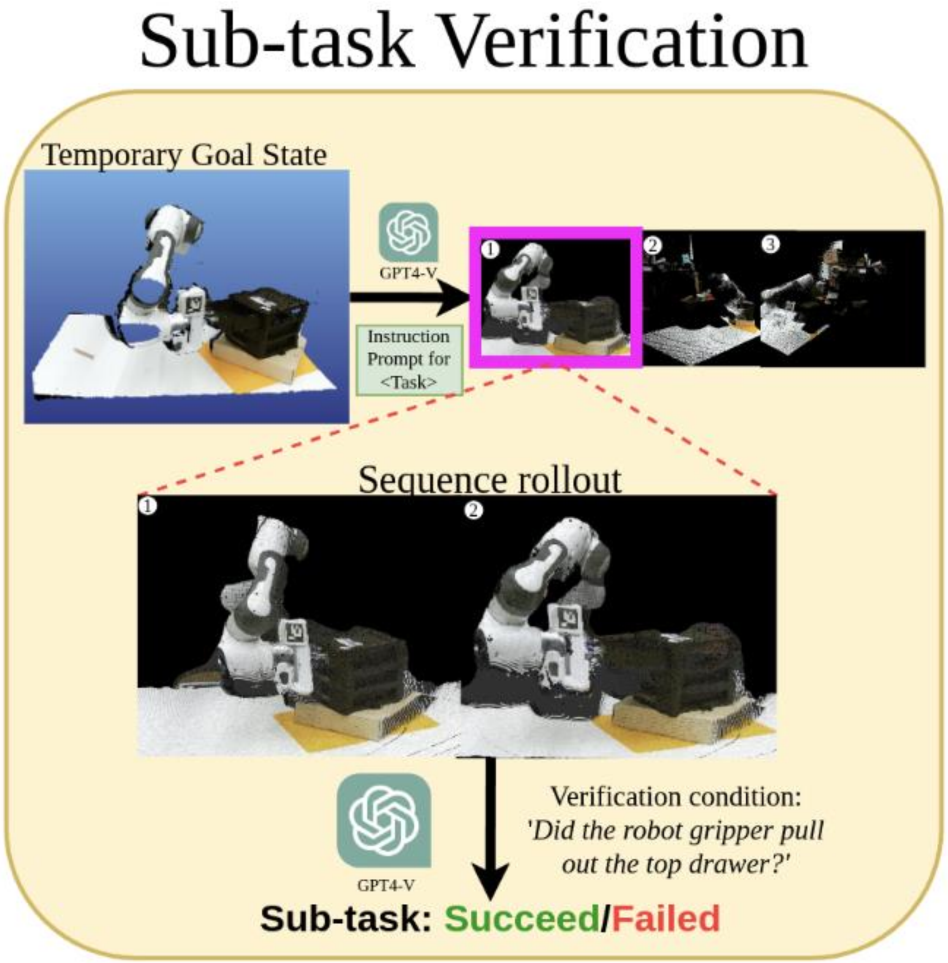
Manipulate Anything: Automating Real-World Robots using Vision-Language Models. Duan et al.

LLMs that Perceive, Plan Subtasks, Determine the Motion, Monitor the Progress



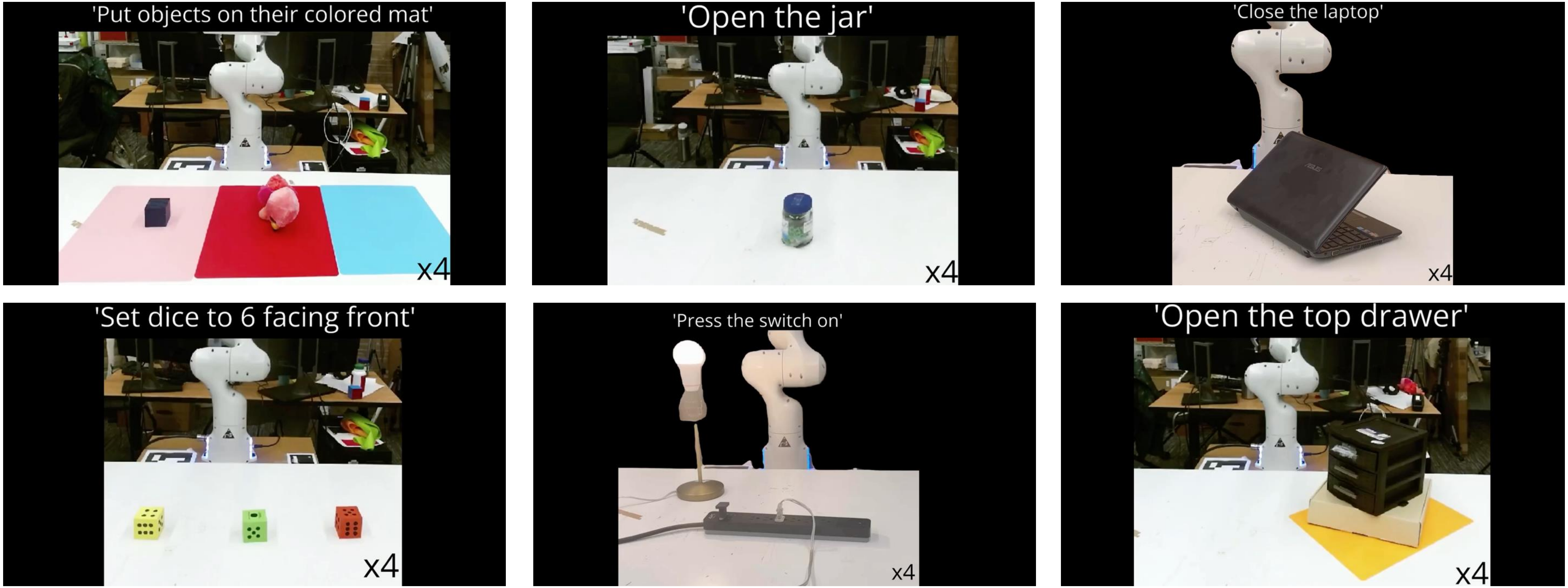
Manipulate Anything: Automating Real-World Robots using Vision-Language Models. Duan et al.

LLMs that Perceive, Plan Subtasks, Determine the Motion, Monitor the Progress



Manipulate Anything: Automating Real-World Robots using Vision-Language Models. Duan et al.

Manipulate Anything: Automating Real-World Robots using Vision-Language Models. Duan et al.



Manipulate Anything: Automating Real-World Robots using Vision-Language Models. Duan et al.