Task and Motion Planning

A Brief Tutorial

ICAPS 2026 Summer School

Tom Silver | Assistant Professor, Princeton University

Naman Shah | Research Scientist, Allen AI Institute (AI2)

Goals for Today

Convey **key intuition**!

What is TAMP?
When and why is it hard?
What are main ideas behind common approaches?

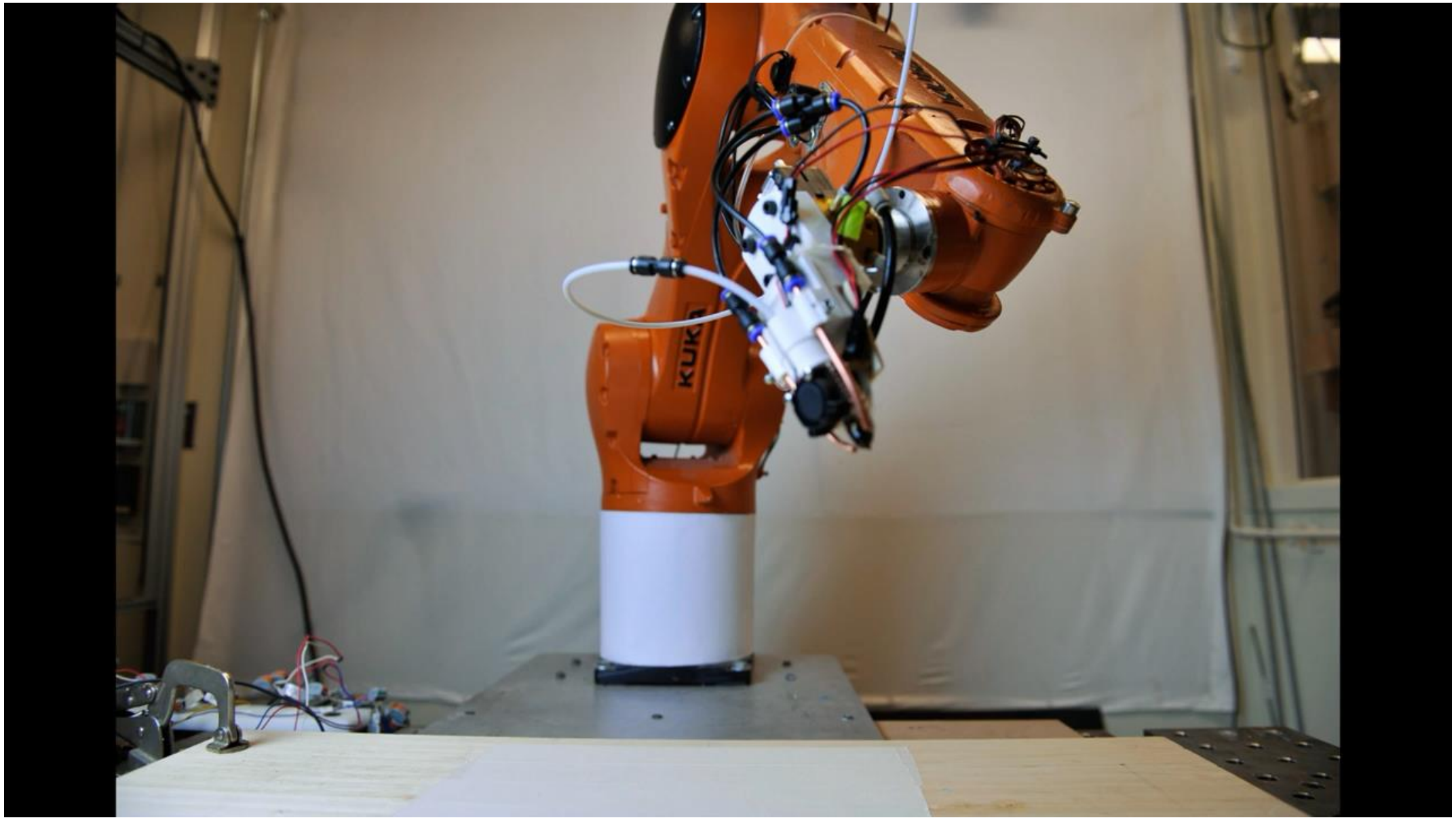
Not an exhaustive survey

But If Surveys Are Your Thing…

“Integrated Task and Motion Planning.” Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, Tomás Lozano-Pérez. ARCRAS 2021.

“A Survey of Optimization-based Task and Motion Planning: From Classical To Learning Approaches.” Zhigen Zhao, Shuo Cheng, Yan Ding, Ziyi Zhou, Shiqi Zhang, Danfei Xu, Ye Zhao. IEEE/ASME Transactions on Mechatronics 2024.

Combined task and motion planning through an extensible planner-independent interface layer. Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, Pieter Abbeel. ICRA 2014.

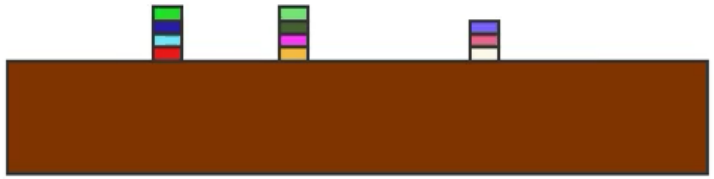


Scalable and Probabilistically Complete Planning for Robotic Spatial Extrusion. Caelan Garrett, Yijiang Huang, Tomás Lozano-Pérez, Caitlin Mueller. RSS 2020.

Morning Agenda

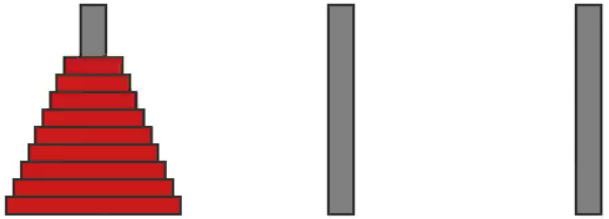
- From Task Planning to Bilevel Planning
- Introduction to Motion Planning
- Multi-Modal Motion Planning and Beyond
- TAMP in the Real World (TipTop)

Recall: Task Planning



Blocks World
Plan length: 28
Planning time: 0.12 s

Sokoban
Plan length: 167
Planning time: 0.25 s



Hanoi
Plan length: 579
Planning time: 0.22 s

Planning with Fast Downward (https://www.fast-downward.org)
Rendering and simulation with PDDLGym (https://github.com/**tomsilver**/pddlgym)

Recall: Task Planning

```
Operator-PickFromTable:

Parameters: [?b - block,
             ?r - robot]

Preconditions: {GripperOpen(?r),
                OnTable(?b)}

Add effects: {Holding(?b)}
Delete effects: {GripperOpen(?r),
                 OnTable(?b)}
```

Recall: Operator Grounding

```
Operator-PickFromTable:

Parameters: [?b - block,
             ?r - robot]

Preconditions: {GripperOpen(?r),
                OnTable(?b)}

Add effects: {Holding(?b)}
Delete effects: {GripperOpen(?r),
                 OnTable(?b)}
```

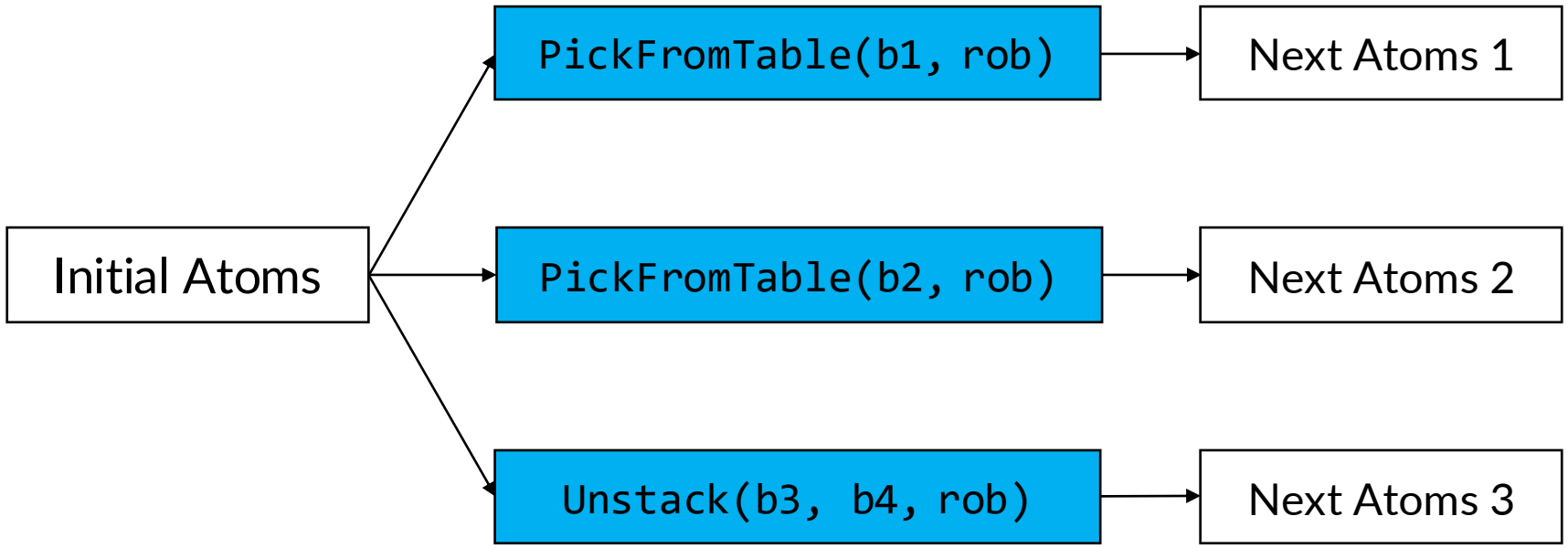
?b → b1

?r → 

Substitution

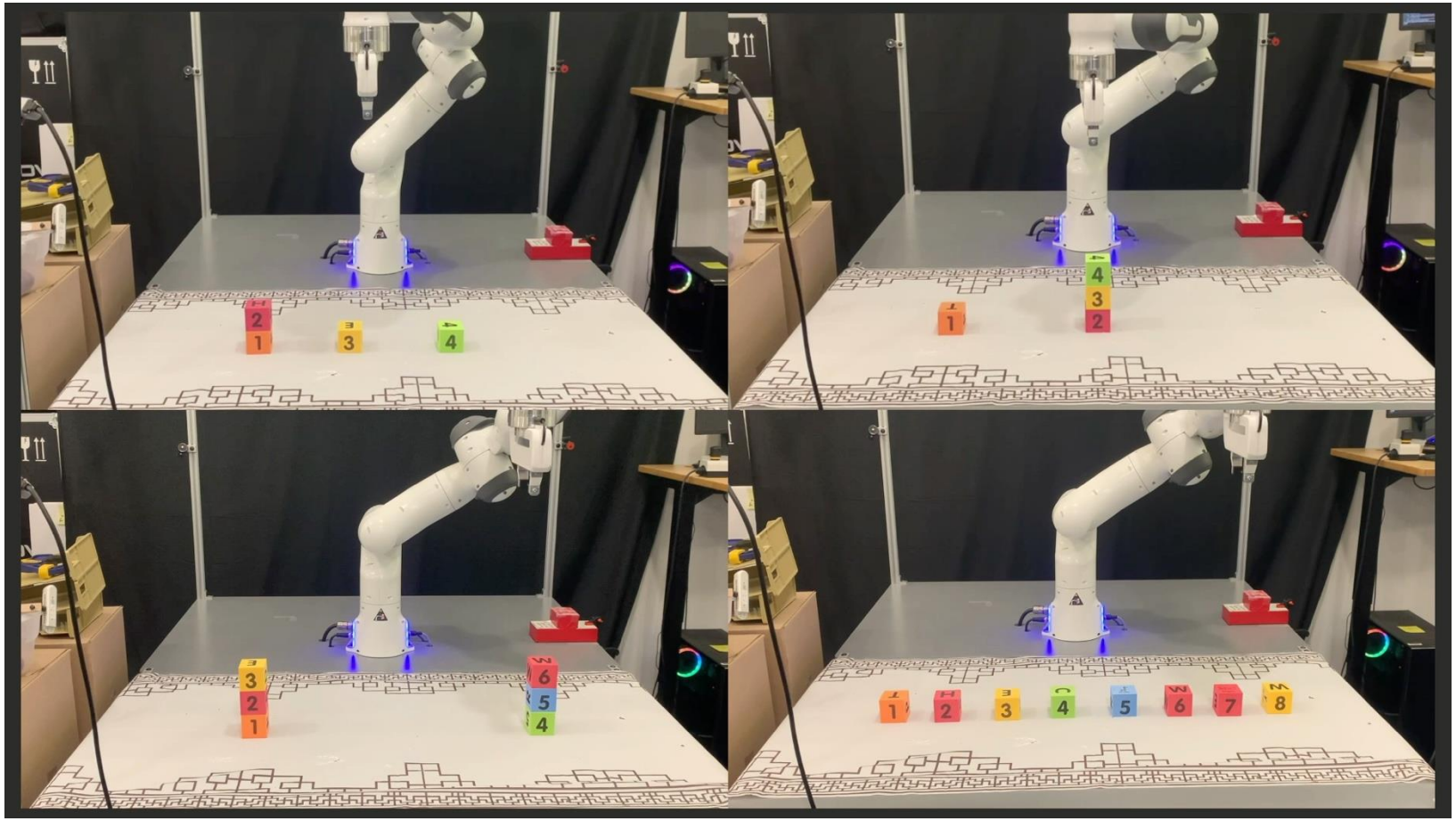
```
Operator-PickFromTable:

Parameters: [ b1 , ]

Preconditions: {GripperOpen( ),
                OnTable( b1 )}

Add effects: {Holding( b1 )}
Delete effects: {GripperOpen( ),
                 OnTable( b1 )}
```

Recall: Task Planning as Search

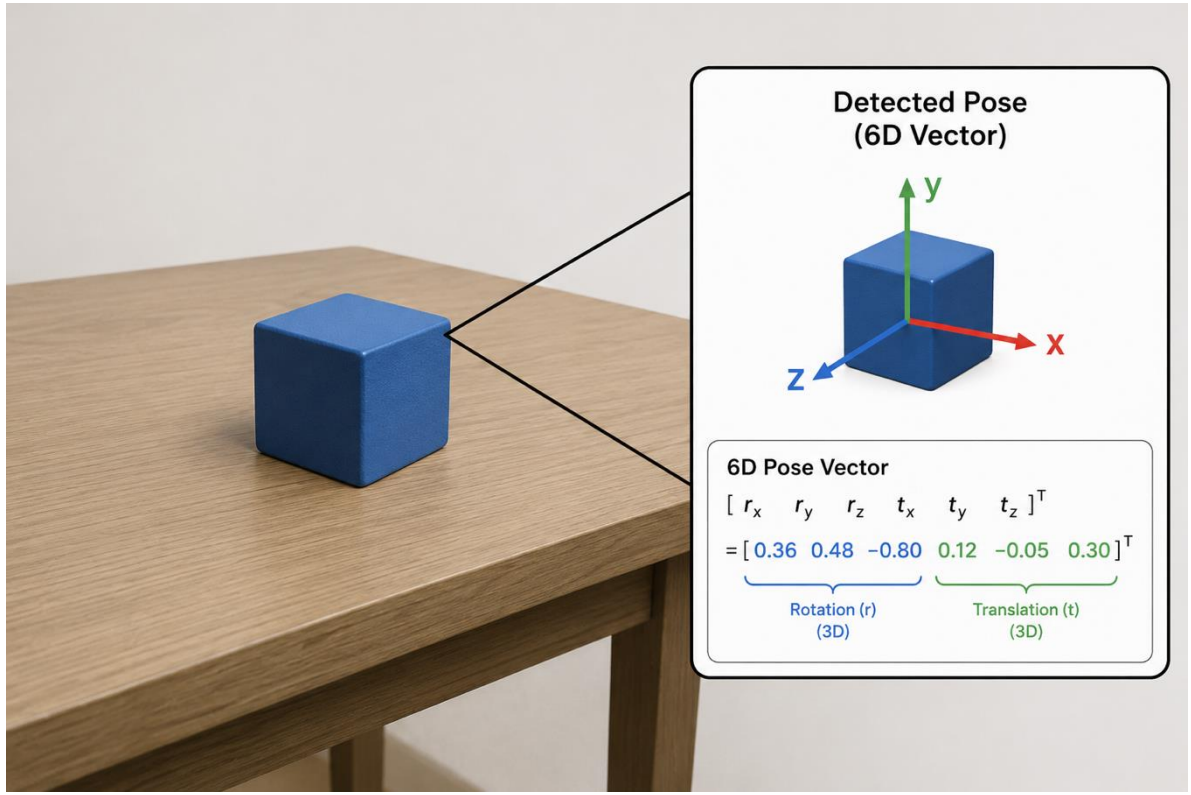


An (partial) transition model

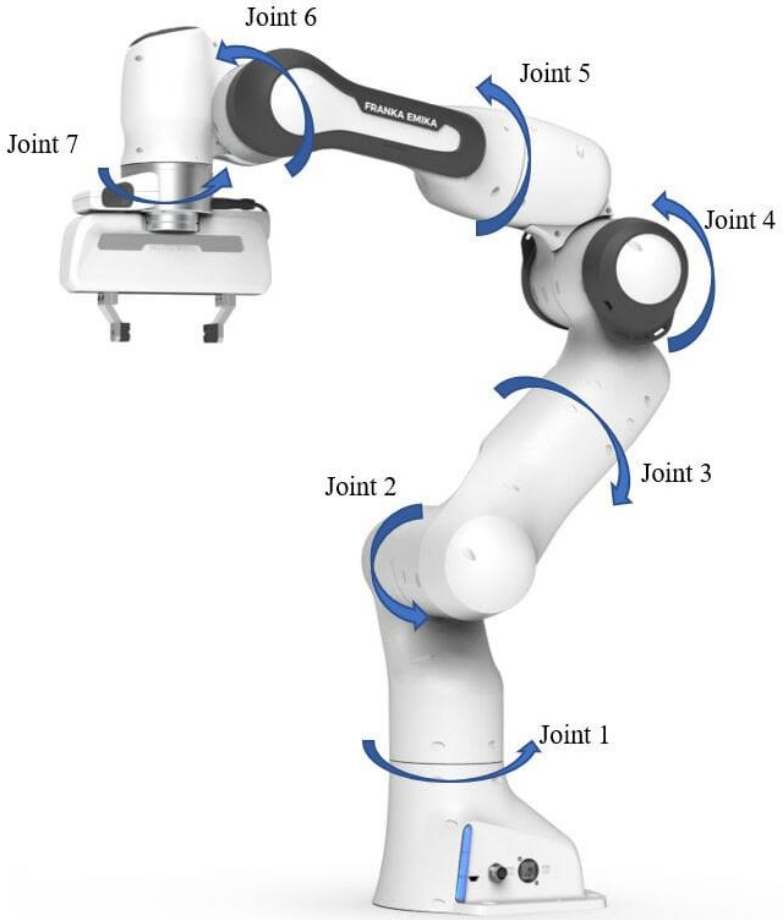
Task Planning on a Robot?



Robotics: Continuous States and Actions

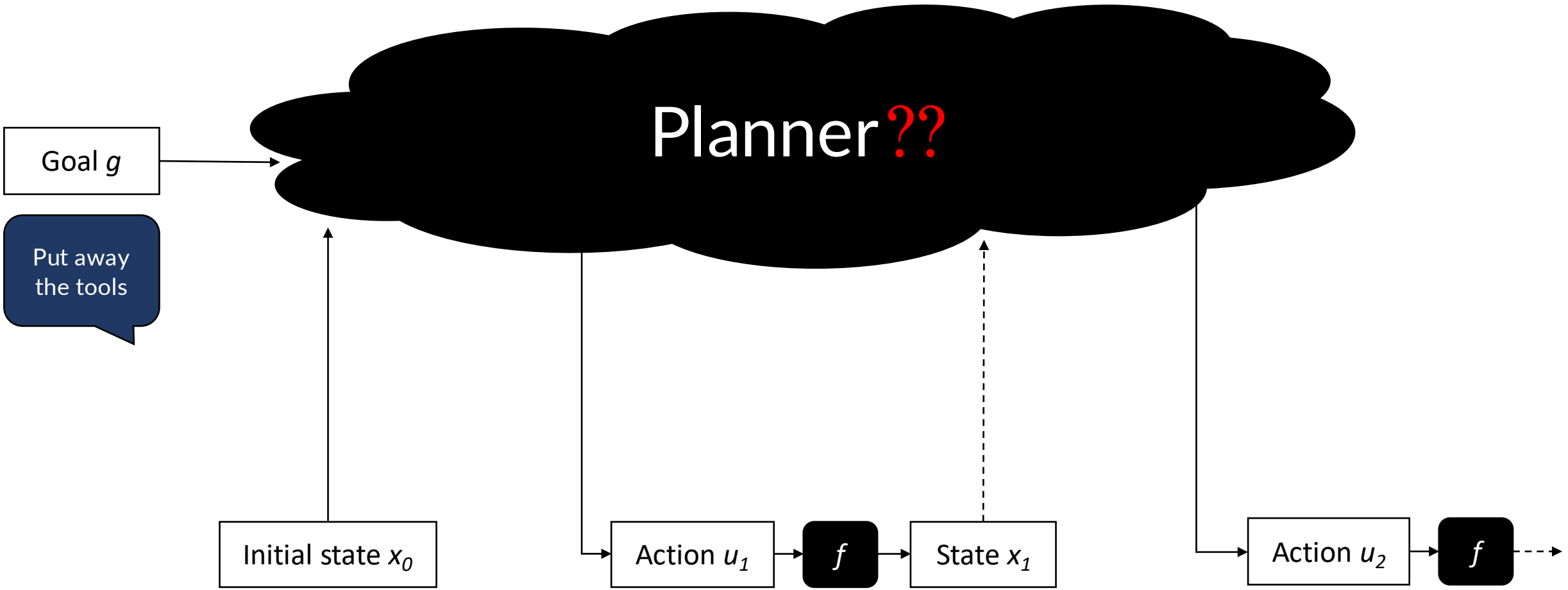


ChatGPT

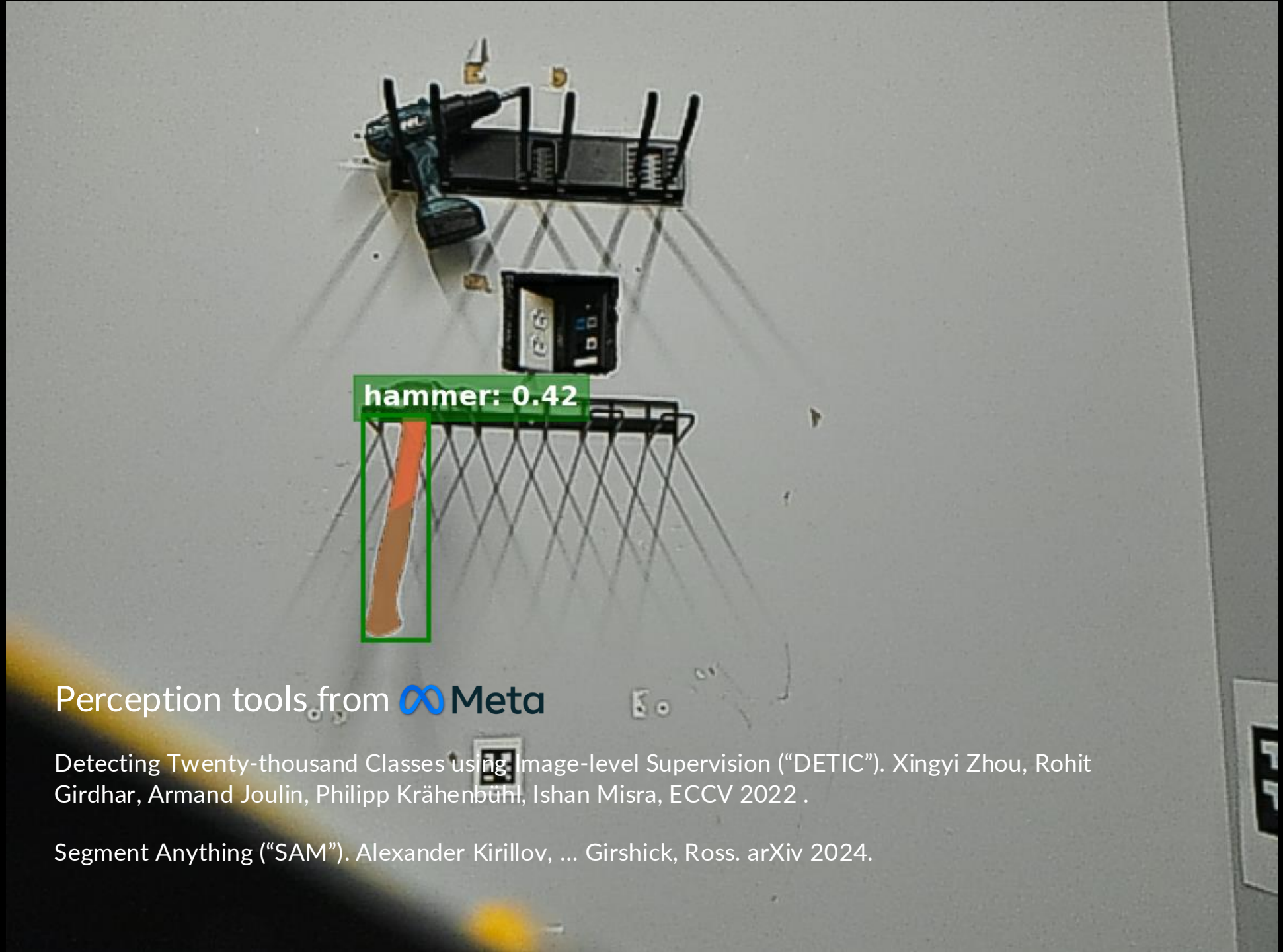


Rogel et al. 2022


Hey Spot, put away the tools!
BostonDynamics


Planner??
Goal g
Put away the tools
Initial state x_0
Action u_1
f
State x_1
Action u_2
f


BostonDynamics



Perception tools from Meta

Detecting Twenty-thousand Classes using Image-level Supervision (“DETIC”). Xingyi Zhou, Rohit Girdhar, Armand Joulin, Philipp Krähenbühl, Ishan Misra, ECCV 2022 .

Segment Anything (“SAM”). Alexander Kirillov, … Girshick, Ross. arXiv 2024.

Object-Centric States

id	type	x	y	height	…
hammer	tool	7.44	5.25	0.32	…

id	type	x	y	radius	…
bucket	container	0.93	6.52	0.28	…

id	type	x	y	gripper	…
Spot	robot	5.25	4.91	0.04	…

… …

Goal g

Put away the tools

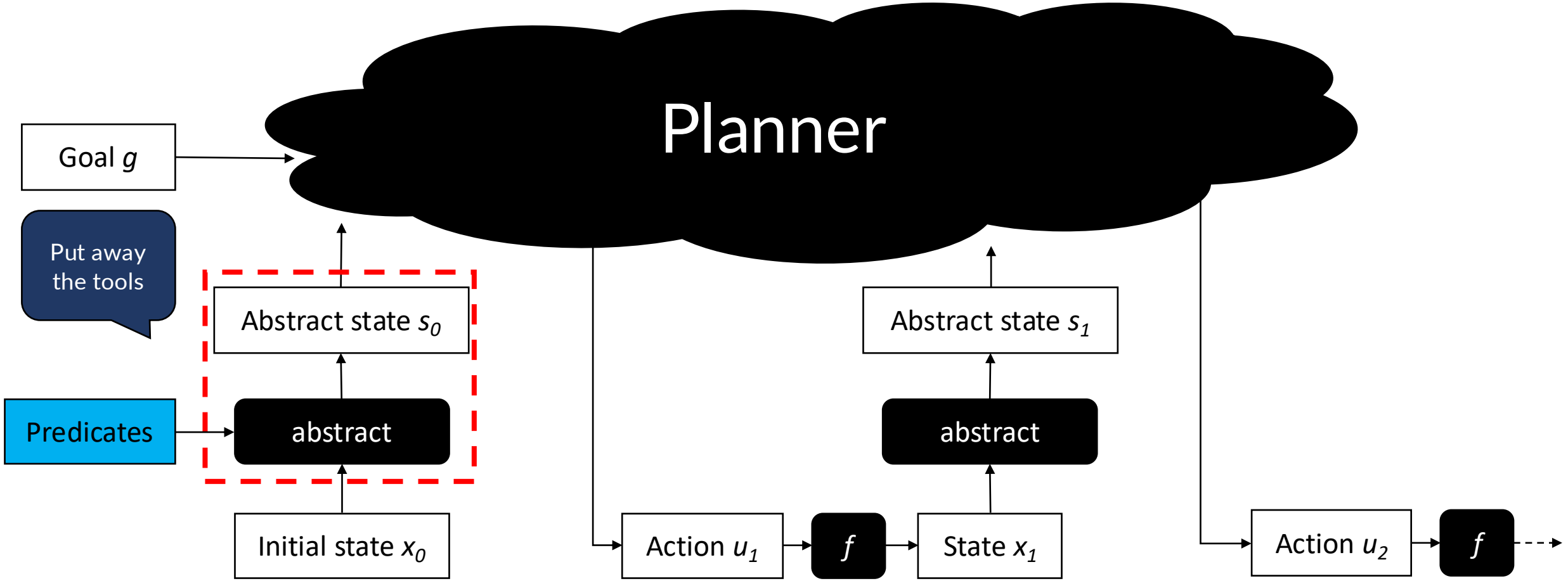
Planner??

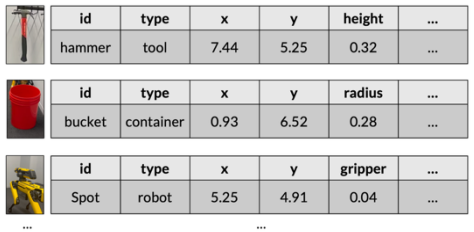
Initial state x_0

Action u_1 → f → State x_1

Action u_2 → f → ...

How would you do it?





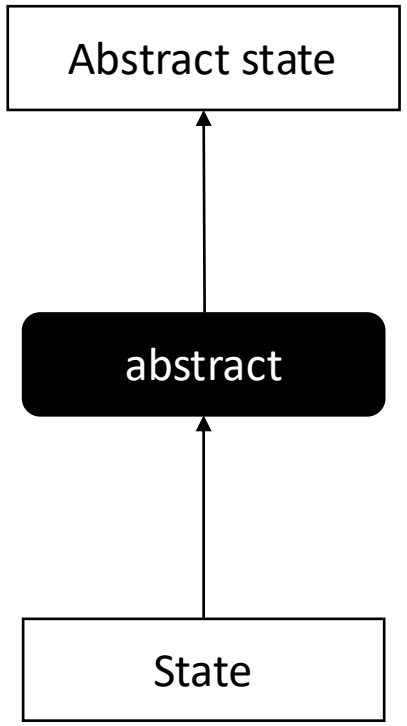
id	type	x	y	height	...
hammer	tool	7.44	5.25	0.32	...

id	type	x	y	radius	...
bucket	container	0.93	6.52	0.28	...

id	type	x	y	gripper	...
Spot	robot	5.25	4.91	0.04	...

State abstraction via predicates

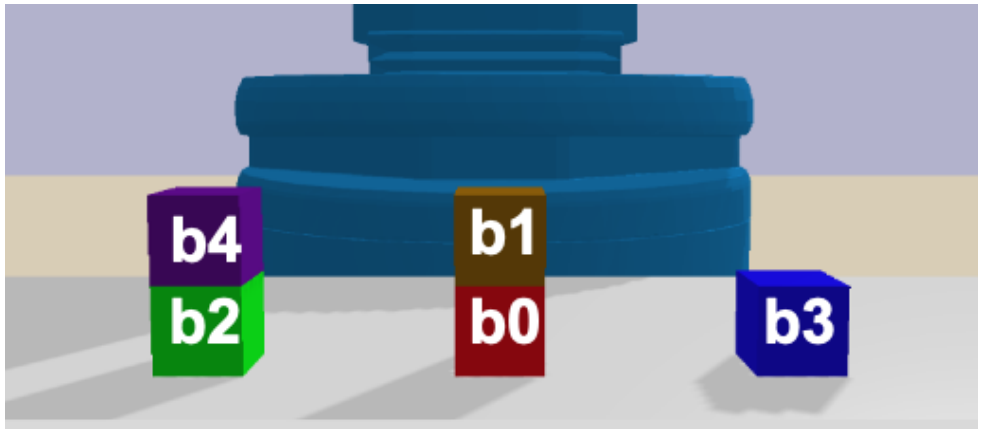
Predicates: **Relational Classifiers** over Object States

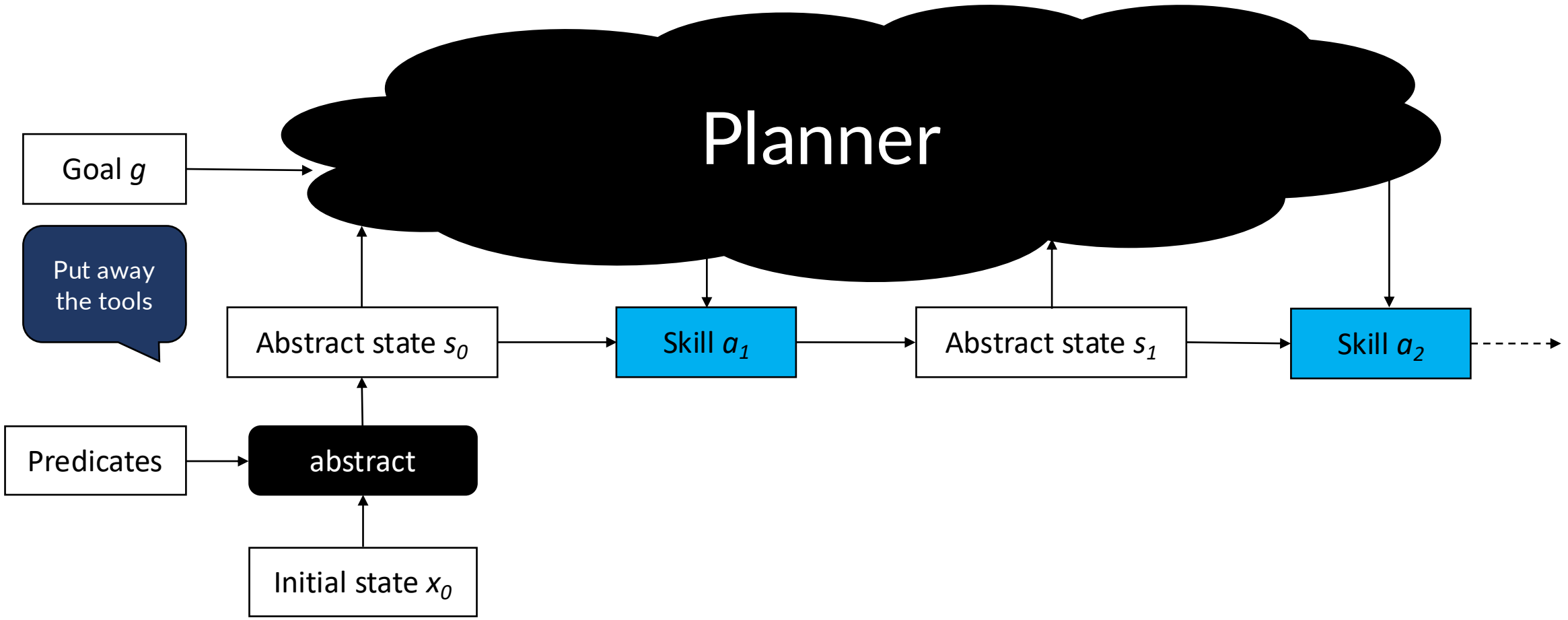



```
OnTable(b2), On(b4, b2)
OnTable(b0), On(b1, b0)
OnTable(b3)
```

```
def classifyOnTable(state, ?block):
    state[?block].z < 1e-5

def classifyOn(state, ?top, ?bot):
    (state[?top].z - state[?bot].z -
     state[?bot].size) < 1e-5 &
    (state[?top].x - state[?bot].x) < 1e-5 &
    (state[?top].y - state[?bot].y) < 1e-5 &
```

	x	y	z	size
rob	0.63	0.11	0.94	0.5
b0	0.74	0.11	0.00	0.1
b1	0.75	0.10	0.20	0.1
b2	0.50	0.11	0.00	0.1
b3	0.99	0.12	0.00	0.1
b4	0.51	0.11	0.20	0.1





id	type	x	y	height	...
hammer	tool	7.44	5.25	0.32	...

id	type	x	y	radius	...
bucket	container	0.93	6.52	0.28	...

id	type	x	y	gripper	...
Spot	robot	5.25	4.91	0.04	...

Action abstraction via skills

Skill **Operators** (STRIPS)

Arguments

List of typed variables

Preconditions

What must be true in order to use this operator?

Add/Delete Effects

How is the abstract state changed by this operator?

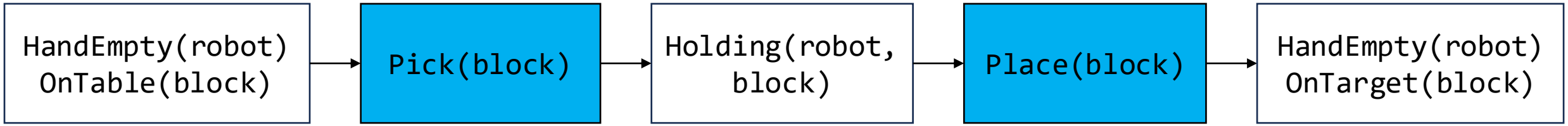
Operator-PickFromTable:

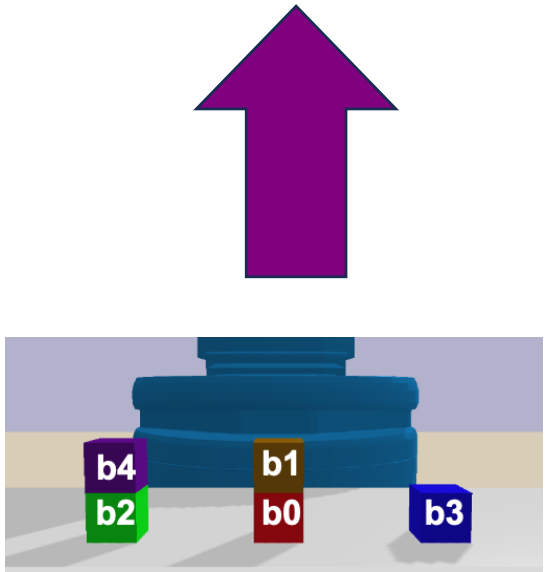
Arguments: [?b - block, ?r - robot]

Preconditions: {GripperOpen(?r), OnTable(?b)}

Add effects: {Holding(?b)}
Delete effects: {GripperOpen(?r), OnTable(?b)}

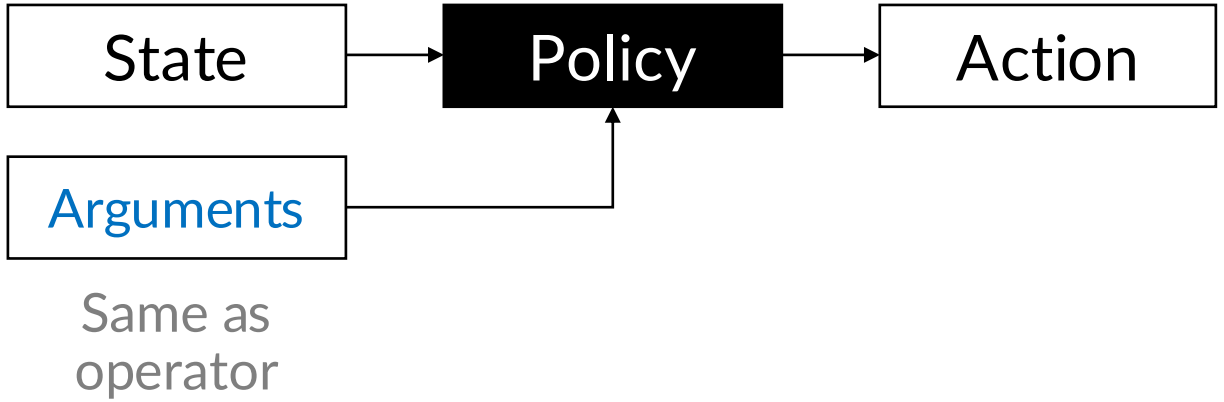
How to “Execute” an Abstract Plan?





How would you do it?

Implementing Action Abstraction with Relational Policies



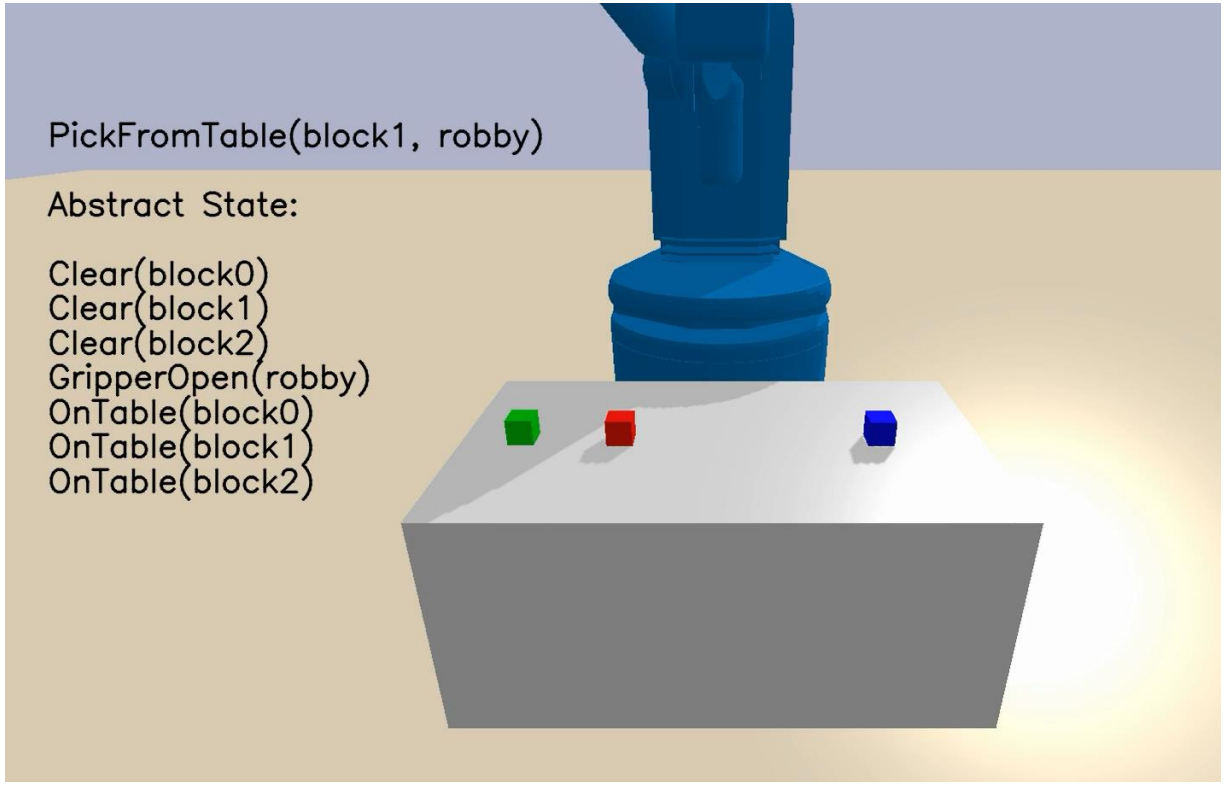
Same as operator

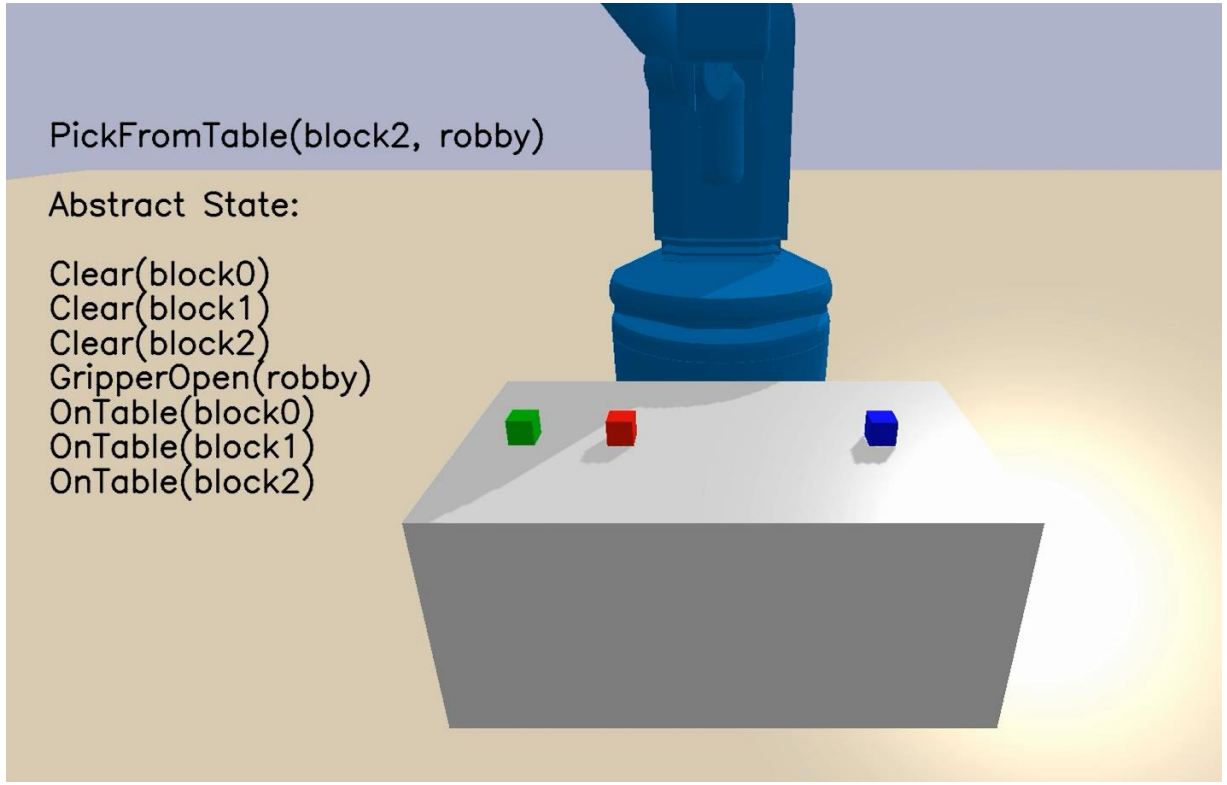
```
def policyPickFromTable(state, ?b, ?r):
    dx = (state[?b].x - state[?r].x)
    dy = (state[?b].y - state[?r].y)
    dz = (state[?b].z - state[?r].z)
    return [dx, dy, dz]
```

Simplified example

The policy should *achieve* the operator effects when the operator preconditions hold

Example Policy Executions



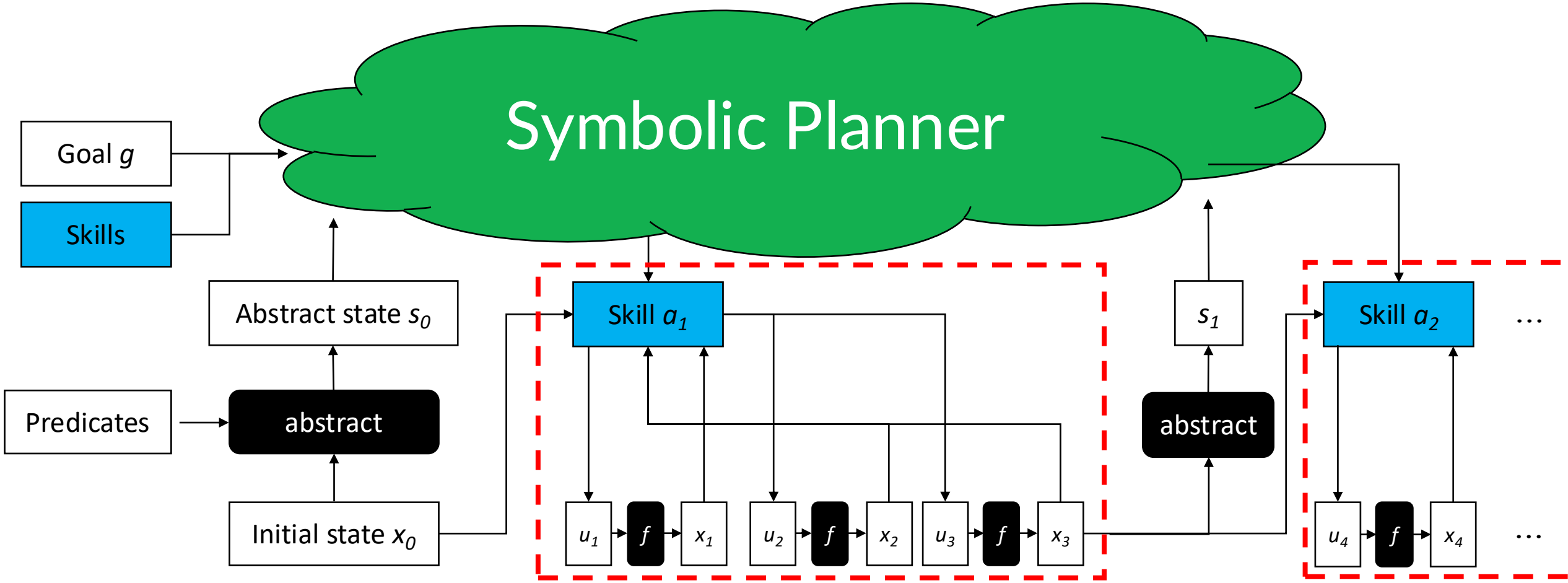


Skills: abstract actions that bring the robot from one abstract state to another

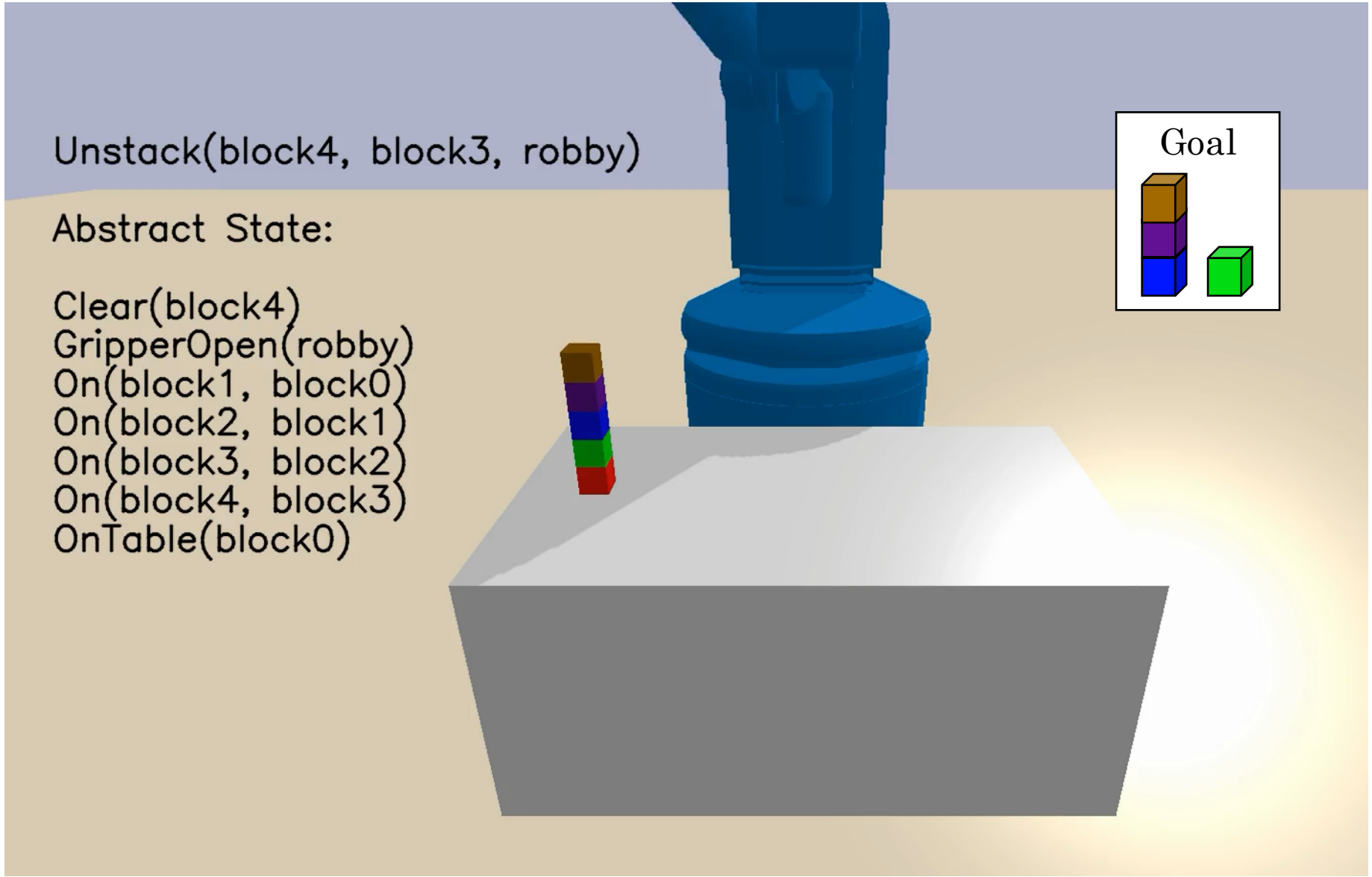
What abstract state transition?

How should I get there?

A skill has an **operator** and a **policy**



Action abstraction via skills


Unstack(block4, block3, robby)
Abstract State:
Clear(block4)
GripperOpen(robby)
On(block1, block0)
On(block2, block1)
On(block3, block2)
On(block4, block3)
OnTable(block0)
Goal

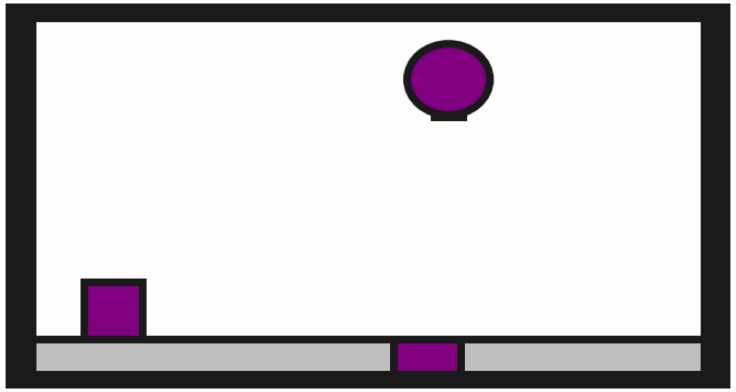
Domain for Tonight's Lab

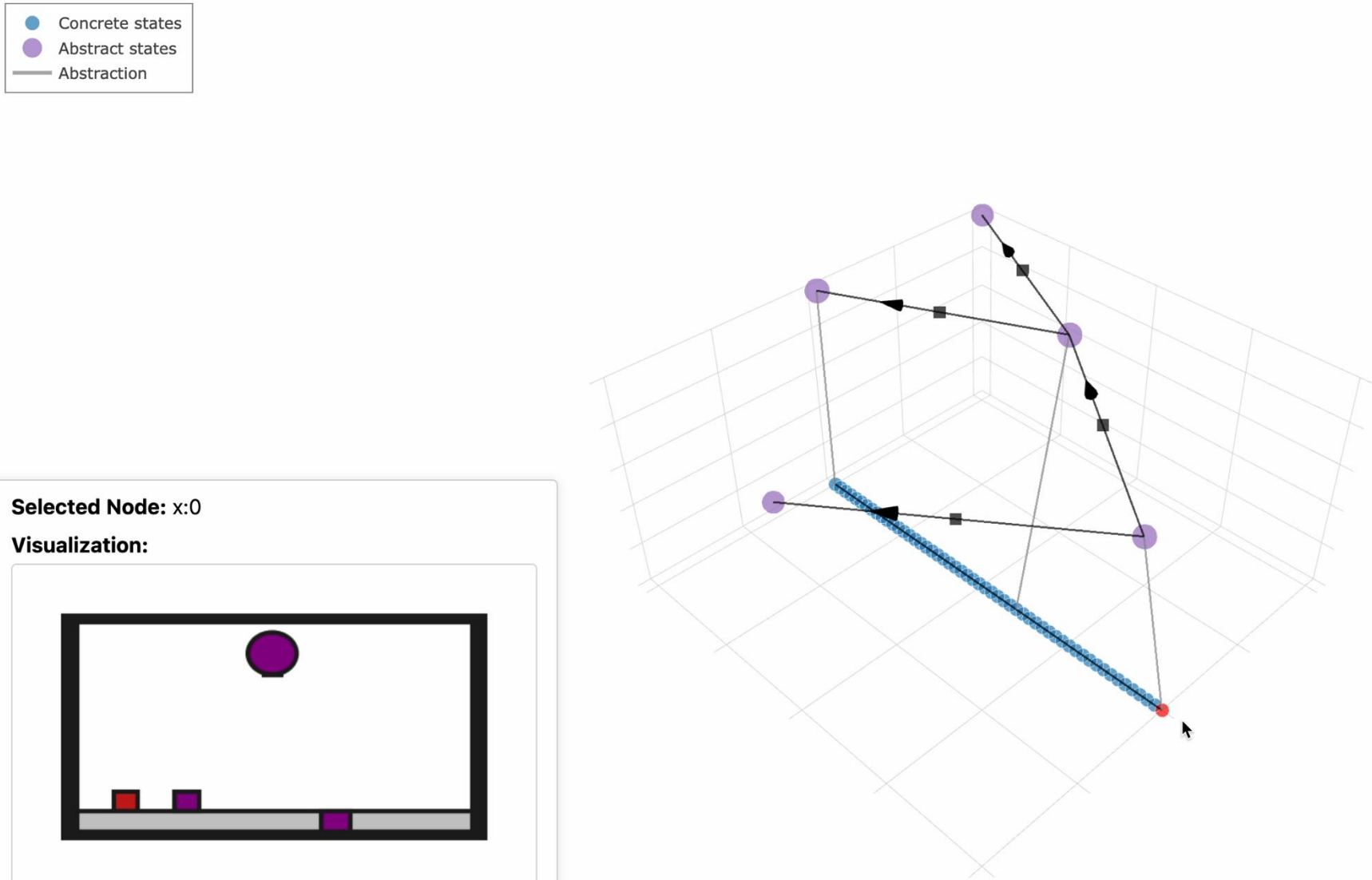
State space: Robot config, block pose (8D)

Action space: Pose change, vacuum (5D)

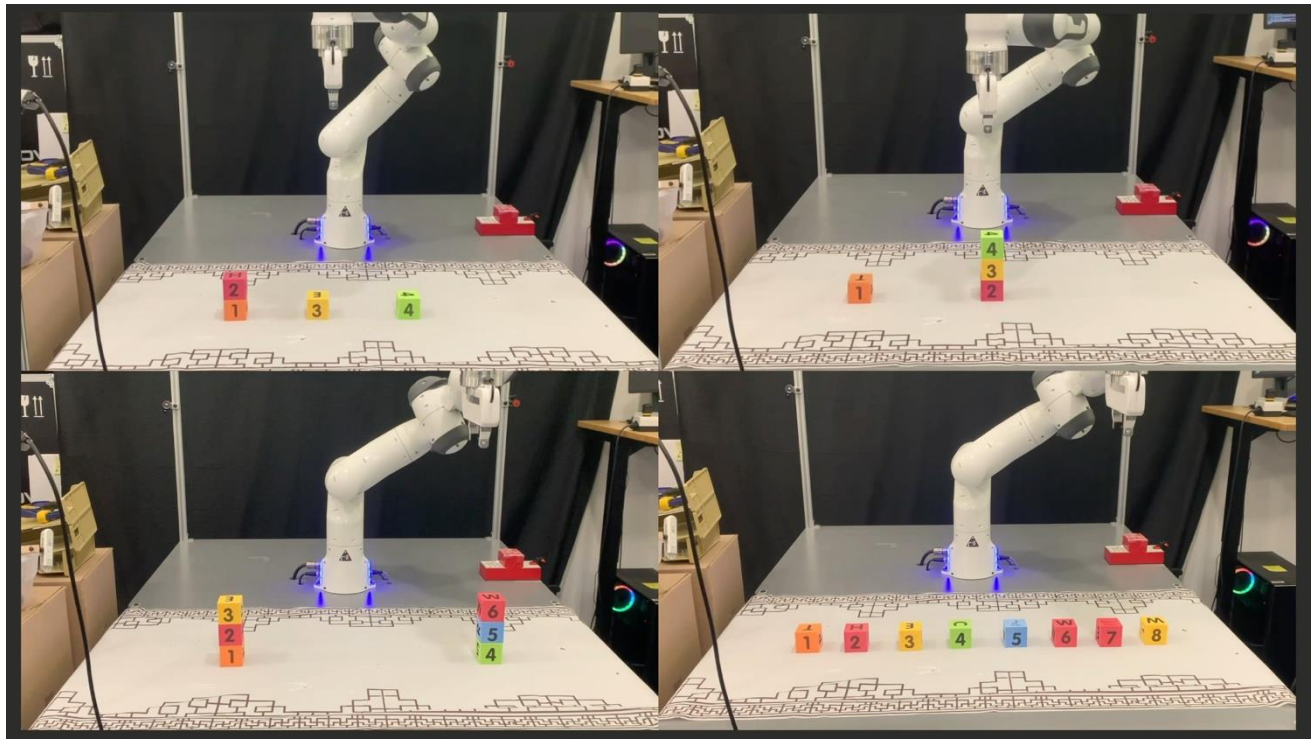
Transition function: Apply action but disallow collisions (no change)

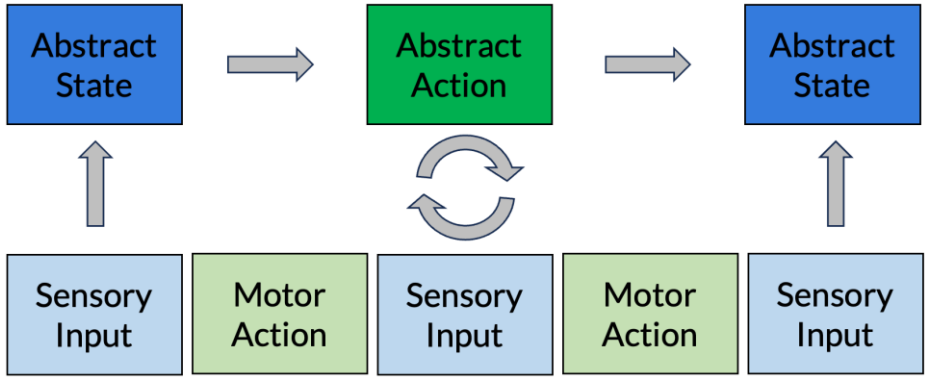
Goal: Block on target region

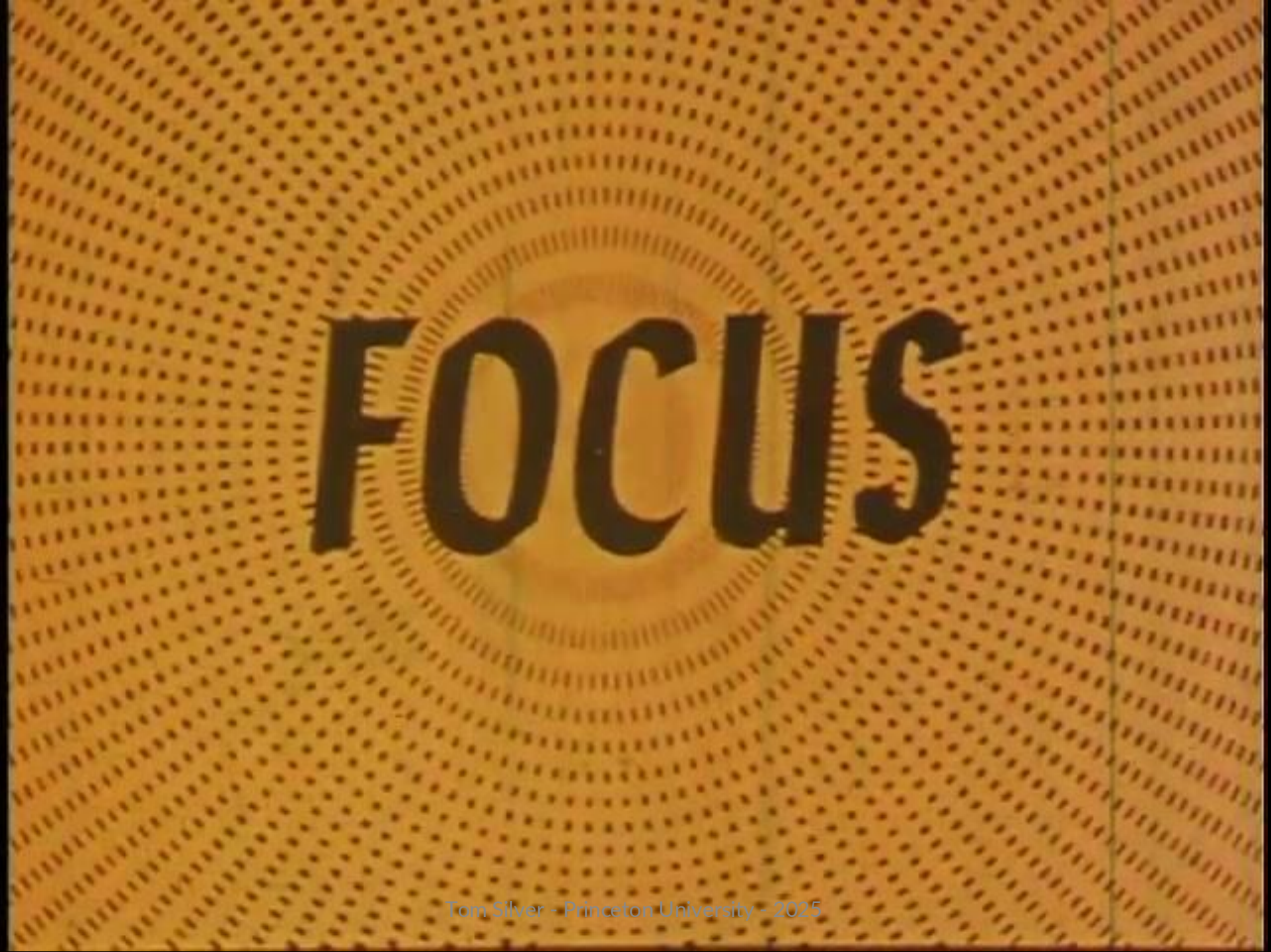


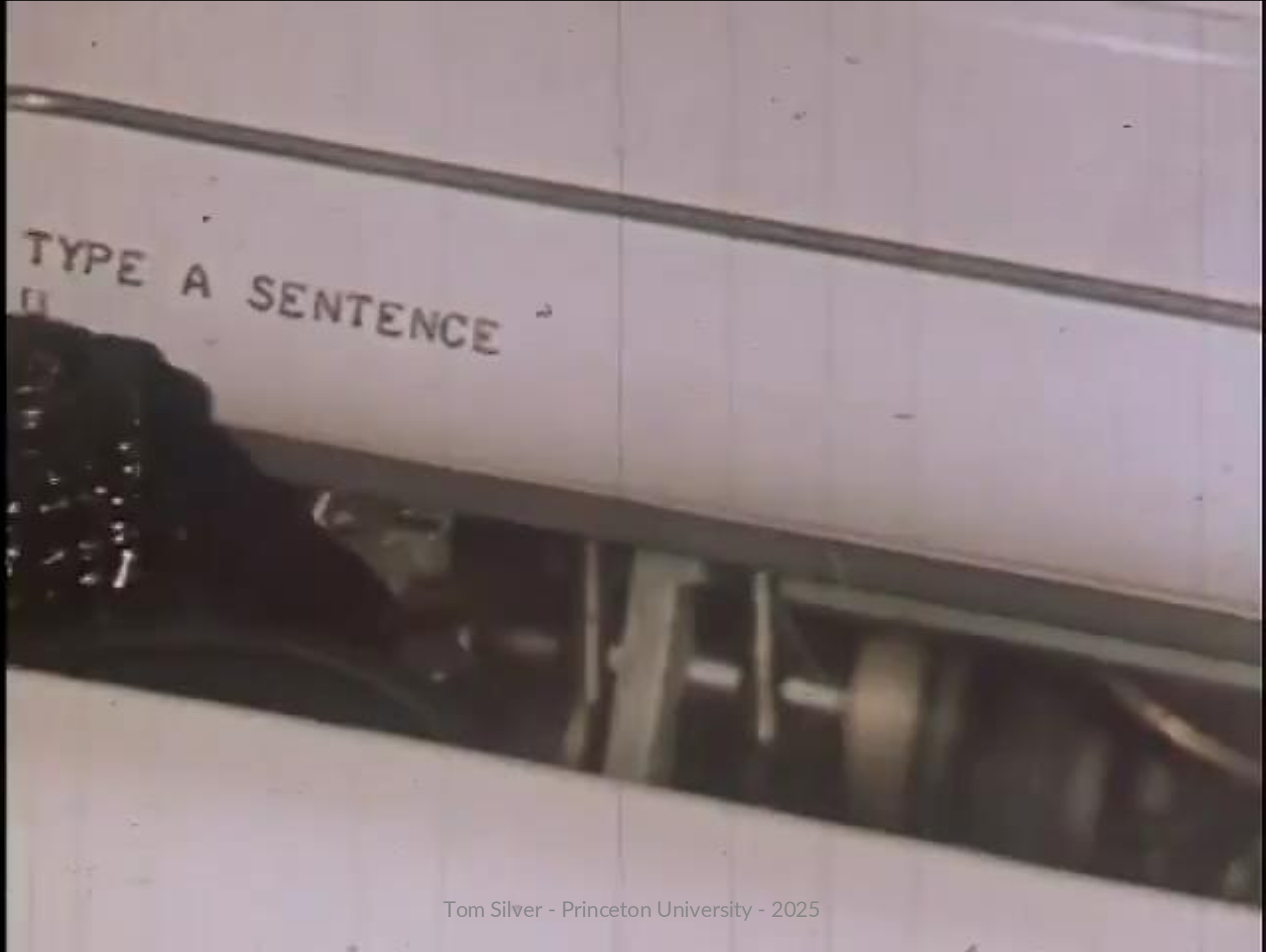

Concrete states
Abstract states
Abstraction
Selected Node: x:0
Visualization:
Timeline: 1 / 54
< Prev
Next >
Play
Pause
Speed (ms): 200
Red = current; darker = visited.
Can hold prev/next to auto-step.
Zoom
+
−
Reset view
Buttons zoom past the scroll-wheel limit.
Nodes: 59

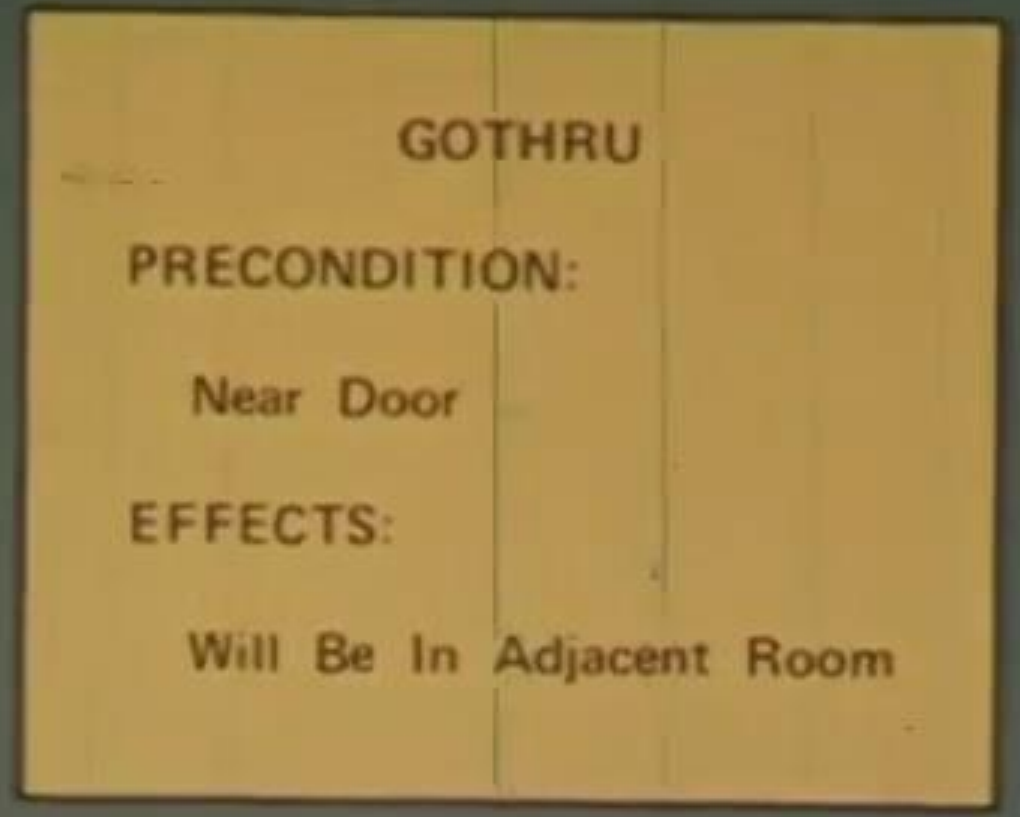
Task Planning on a Robot?

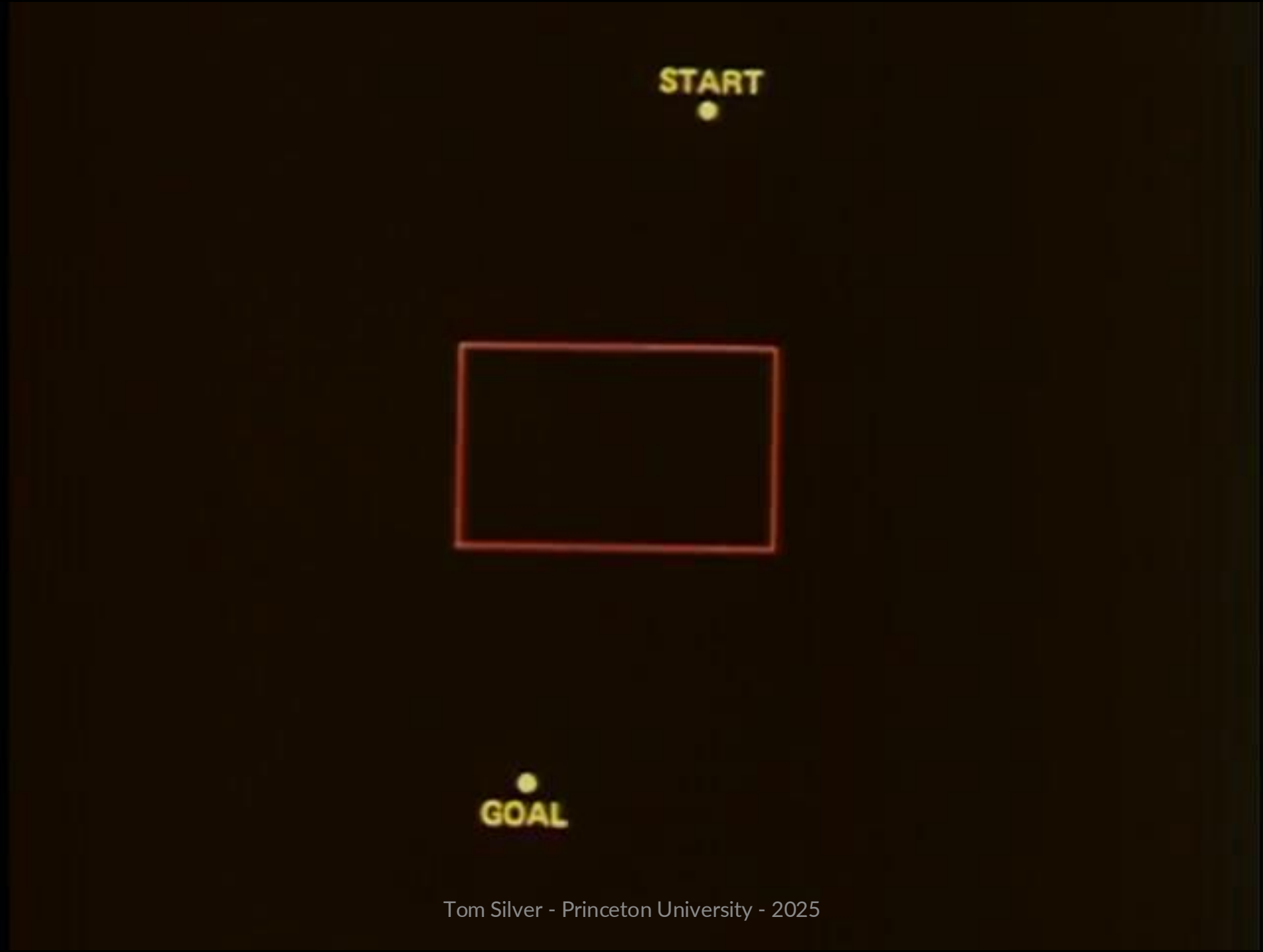


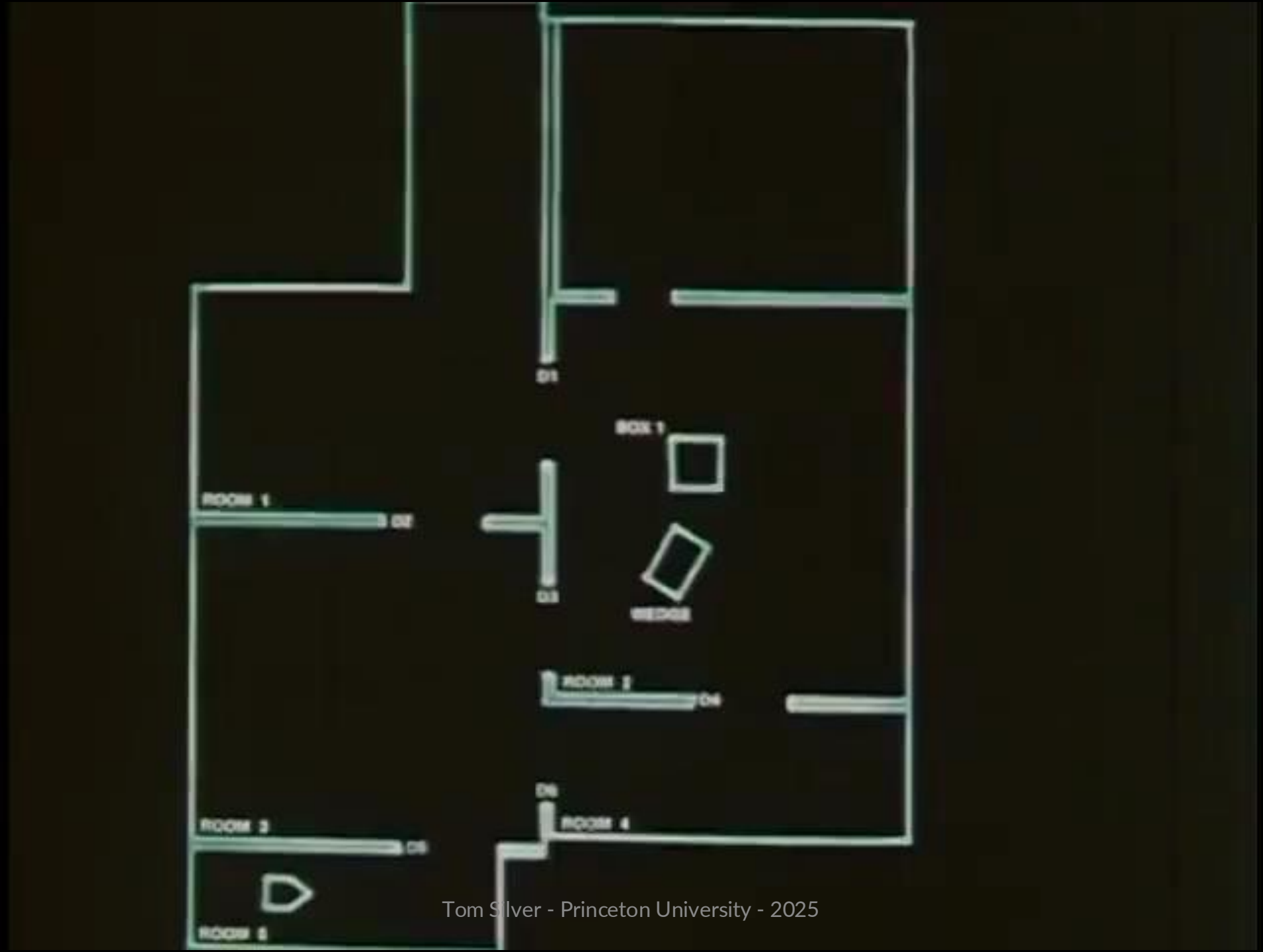



FOCUS

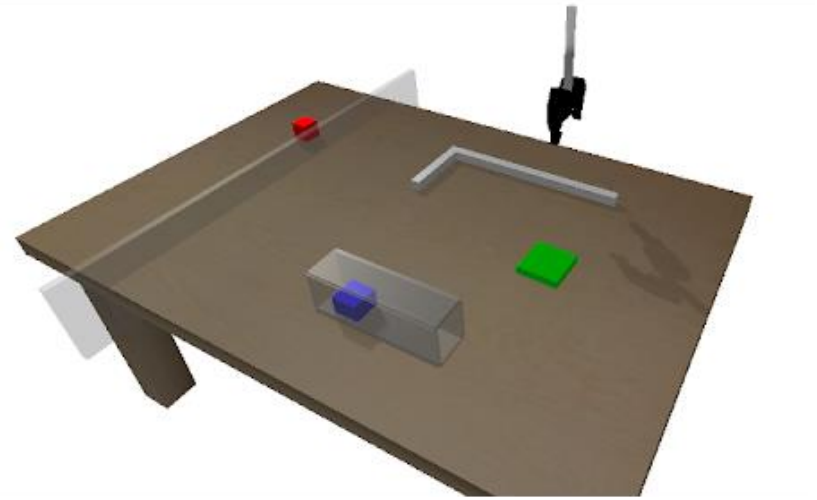

TYPE A SENTENCE


GOTHRU
PRECONDITION:
Near Door
EFFECTS:
Will Be In Adjacent Room


START
GOAL


D1
BOX 1
ROOM 1
D2
D3
WEDGE
ROOM 2
D4
D6
ROOM 3
D5
ROOM 4
ROOM 5

The abstractions might be liars…



“Deep Affordance Foresight: Planning Through What Can Be Done in the Future.” Danfei Xu, Ajay Mandlekar, Roberto Martin-Martin , Yuke Zhu, Silvio Savarese and Li Fei-Fei. ICRA 2021.

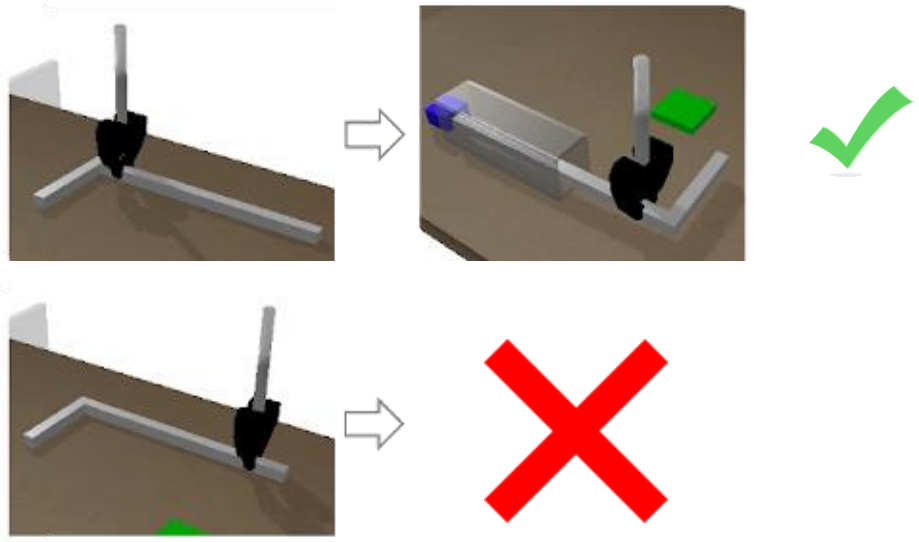
Operator-PushOutOfTube:

Arguments: [, ,]

Preconditions: {Holding(,), InTube( )}

Add effects: {OutOfTube( )}

Delete effects: {InTube( )}



What should we do?

Possible Conclusions from this Example

1. Insufficient predicates → learn new predicates

   HoldingBottom, HoldingTop, etc.

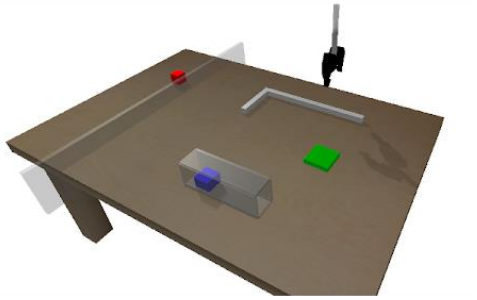
2. Insufficient policies → learn better policies

   Put down the tool and regrasp if needed

3. Insufficient planner → be less trusting of the abstractions

   View abstractions as *guidance* for low-level planning

Parameterized Skill Policies



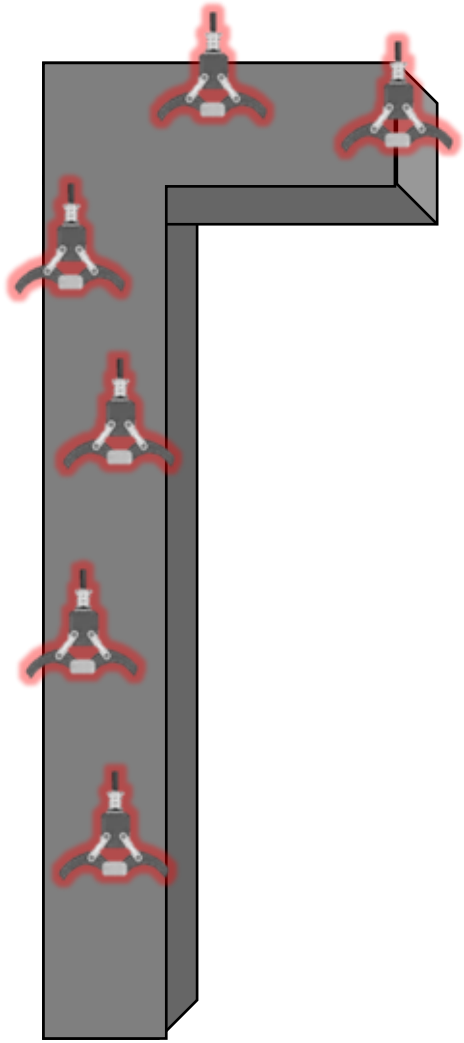
Operator-PickTool:

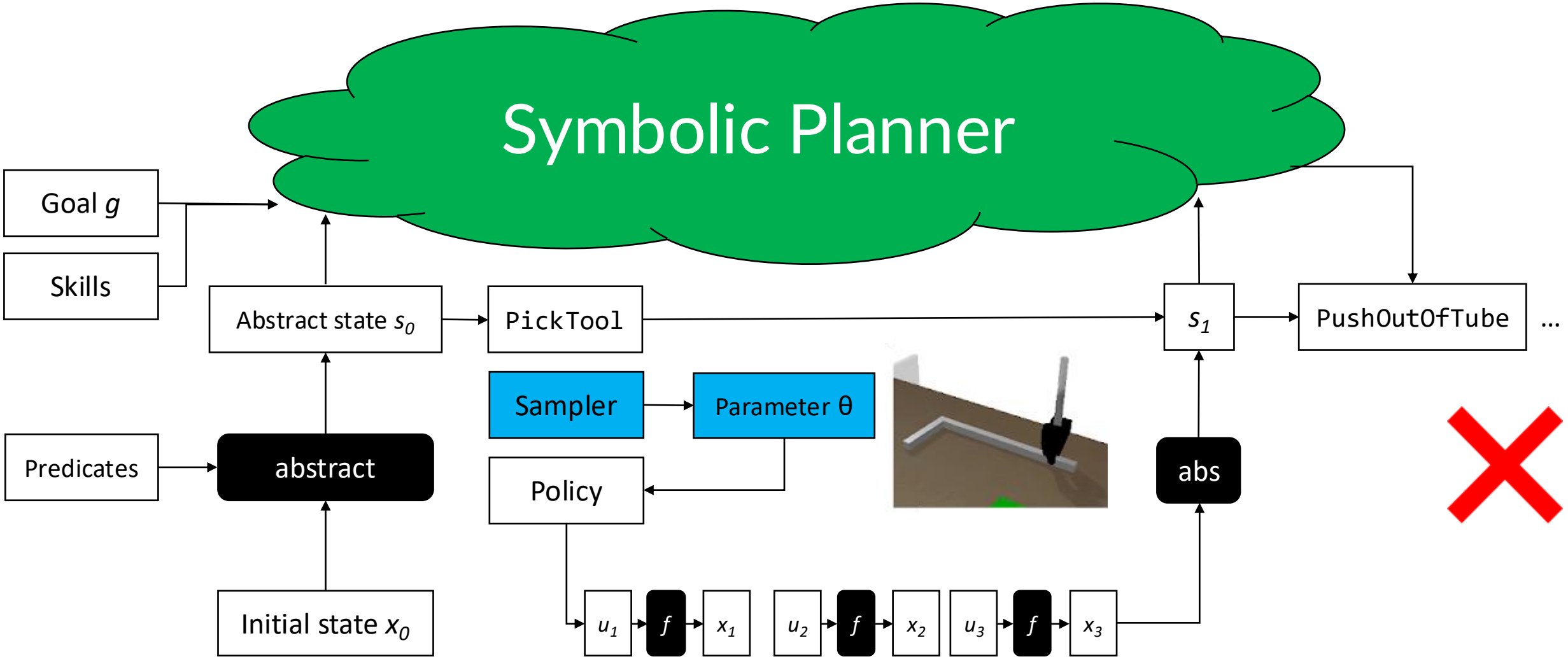
Arguments: [ , ]

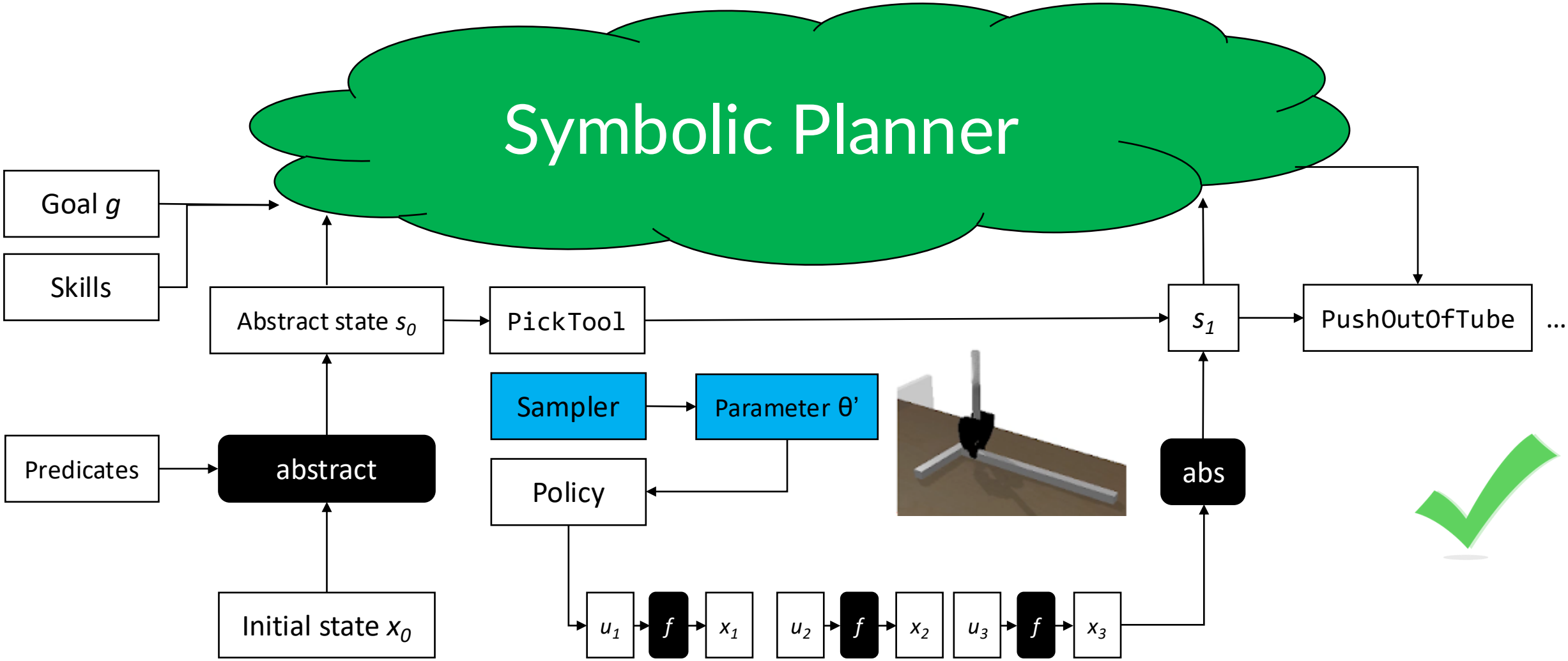
Preconditions: {GripperOpen()}

Add effects: {Holding( , )
Delete effects: {GripperOpen()}

Different Parameters

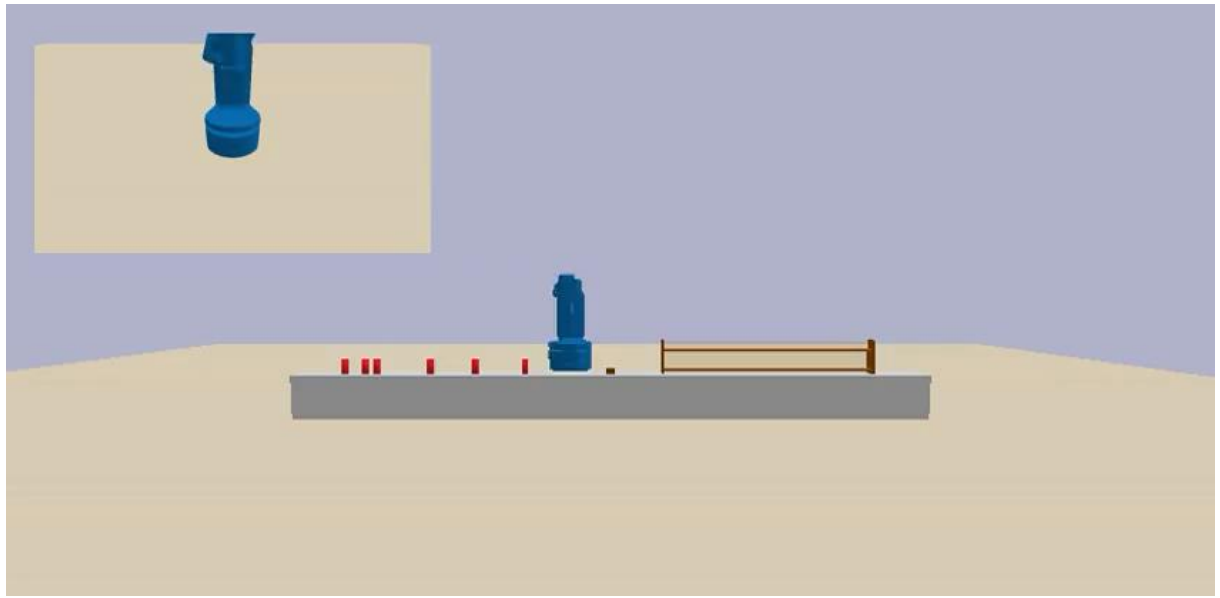



Symbolic Planner
Goal g
Skills
Abstract state s_0
PickTool
s_1
PushOutOfTube
...
Sampler
Parameter θ
Predicates
abstract
Policy
abs
Initial state x_0
u_1
f
x_1
u_2
f
x_2
u_3
f
x_3


Symbolic Planner
Goal g
Skills
Abstract state s_0
PickTool
s_1
PushOutOfTube
...
Sampler
Parameter θ'
Predicates
abstract
Policy
abs
Initial state x_0
u_1
f
x_1
u_2
f
x_2
u_3
f
x_3

Let's Be Clear About How Terrible Backtracking Sampling Can Be

`num_sampling_attempts_per_step`: the number of samples to draw at each step before backtracking

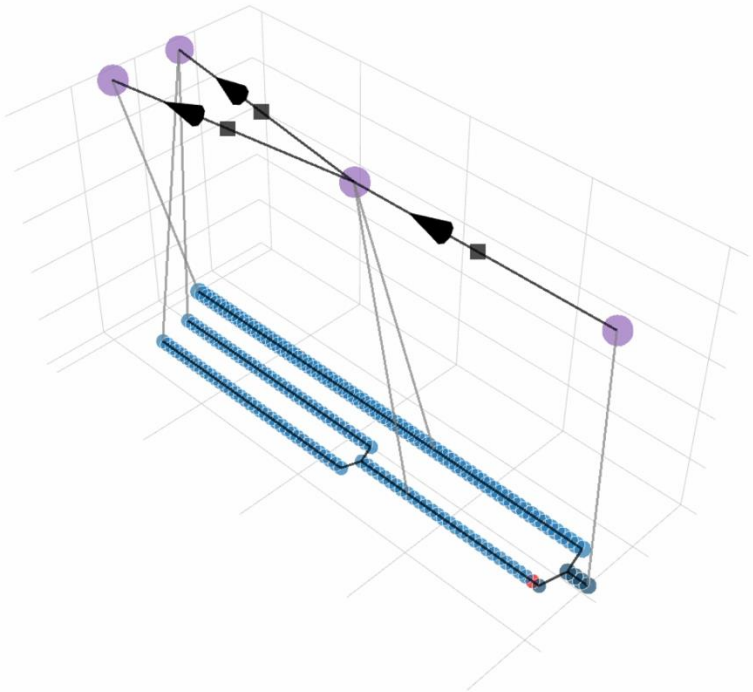


```
Pick (top or side grasp)
Wash
Dry
Paint
Place in box/shelf
```

What's the time complexity of backtracking search?

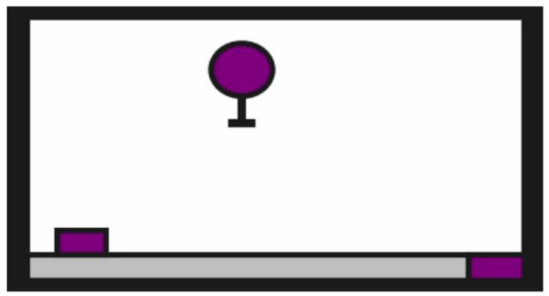
Concrete states
Abstract states
Abstraction

```
num_sampling_attempts_per_step = 2
```



Selected Node: x:5

Visualization:



Timeline: 6 / 178

< Prev | Next > | Play | Pause

Speed (ms): 200

Red = current; darker = visited.

Can hold prev/next to auto-step.

Zoom

\+ | − | Reset view

Buttons zoom past the scroll-wheel limit.

Nodes: 182

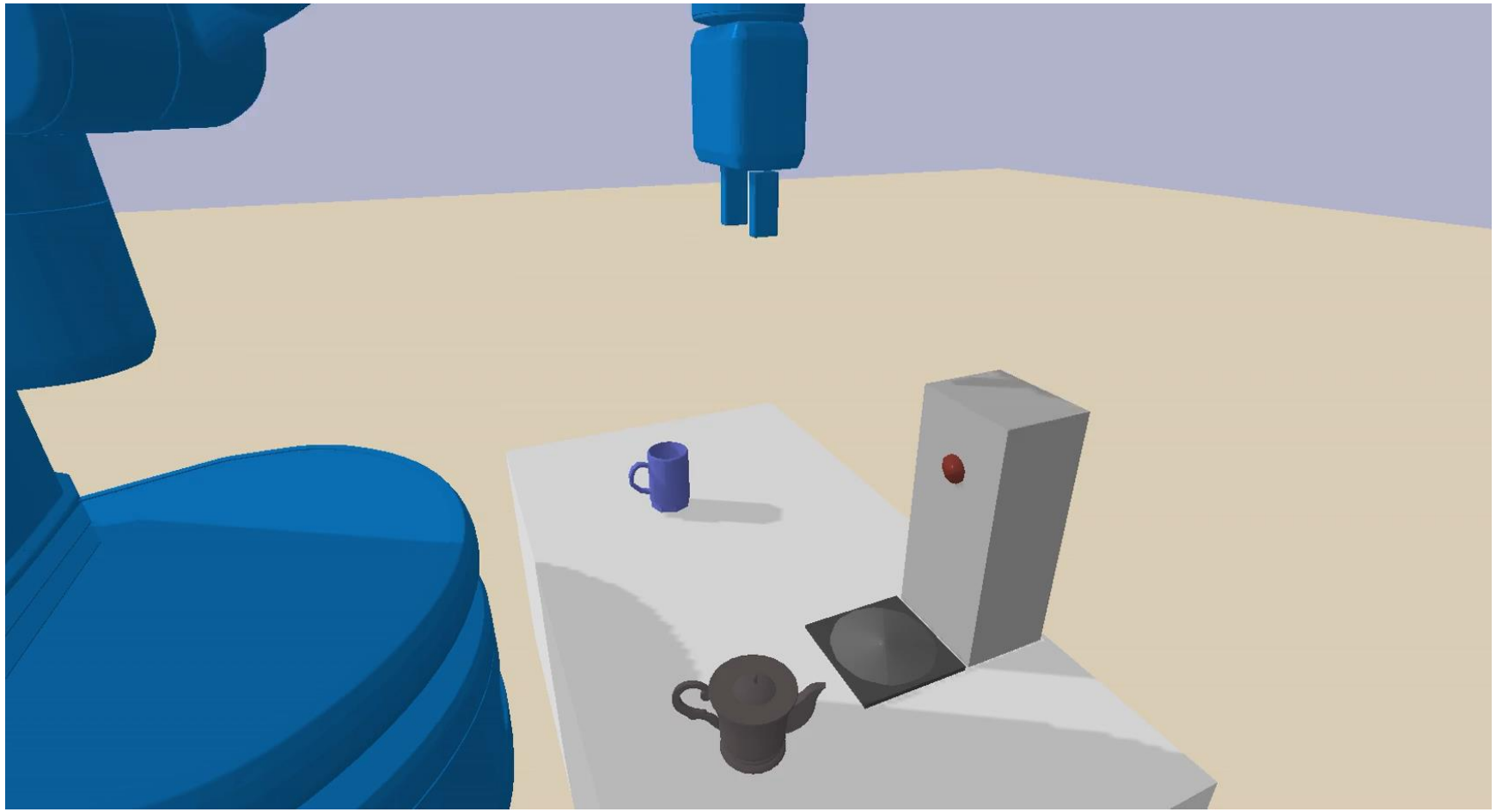
The abstractions may be *pathological* liars…

An abstract plan may not be refinable at all!

Coffee Domain

Abstract Plan

- Grasp handle
- Place on plate
- Turn plate on
- Pick up pot
- Pour into cup

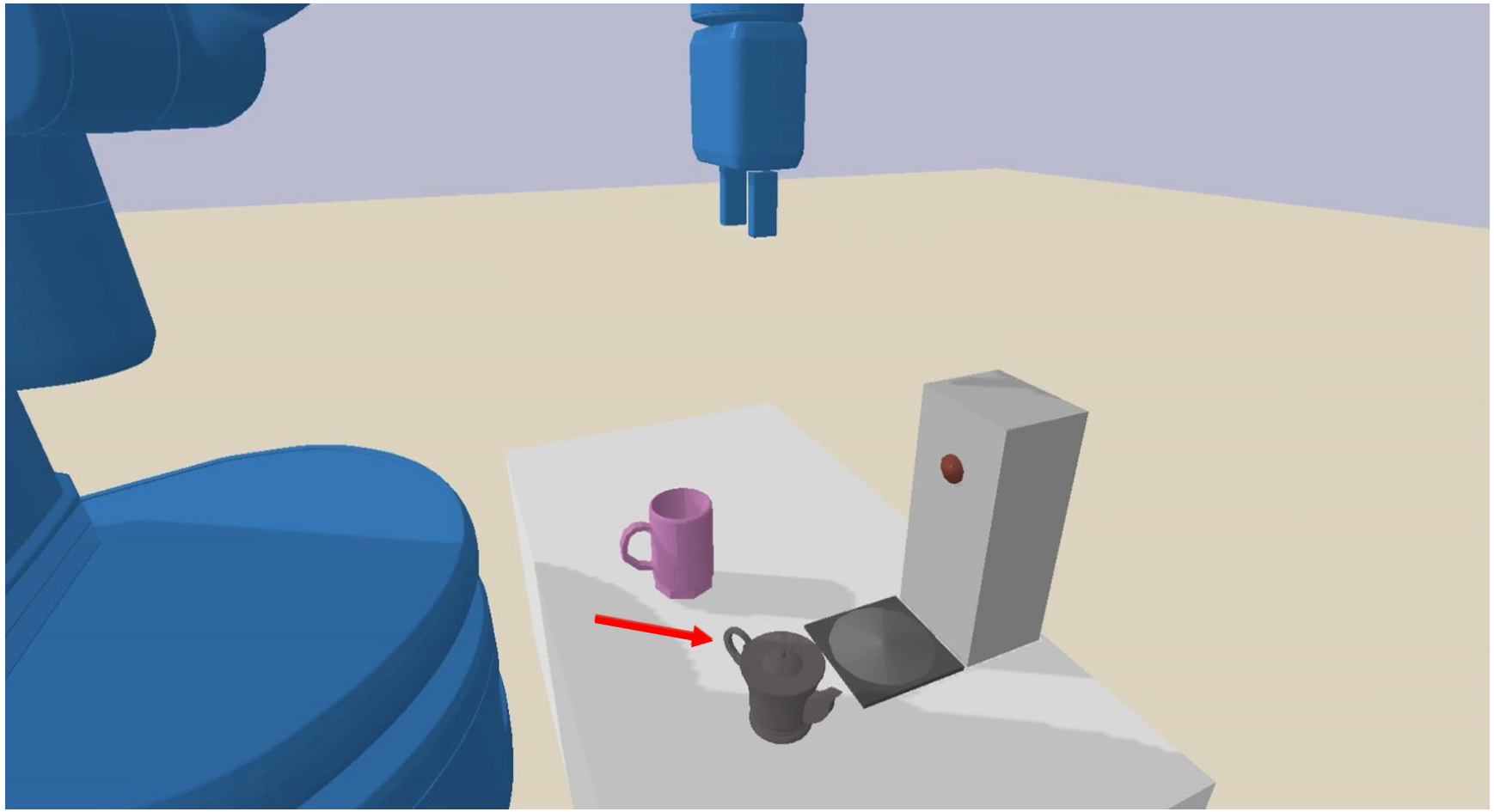


We need a different abstract plan!

Coffee Domain

Abstract Plan

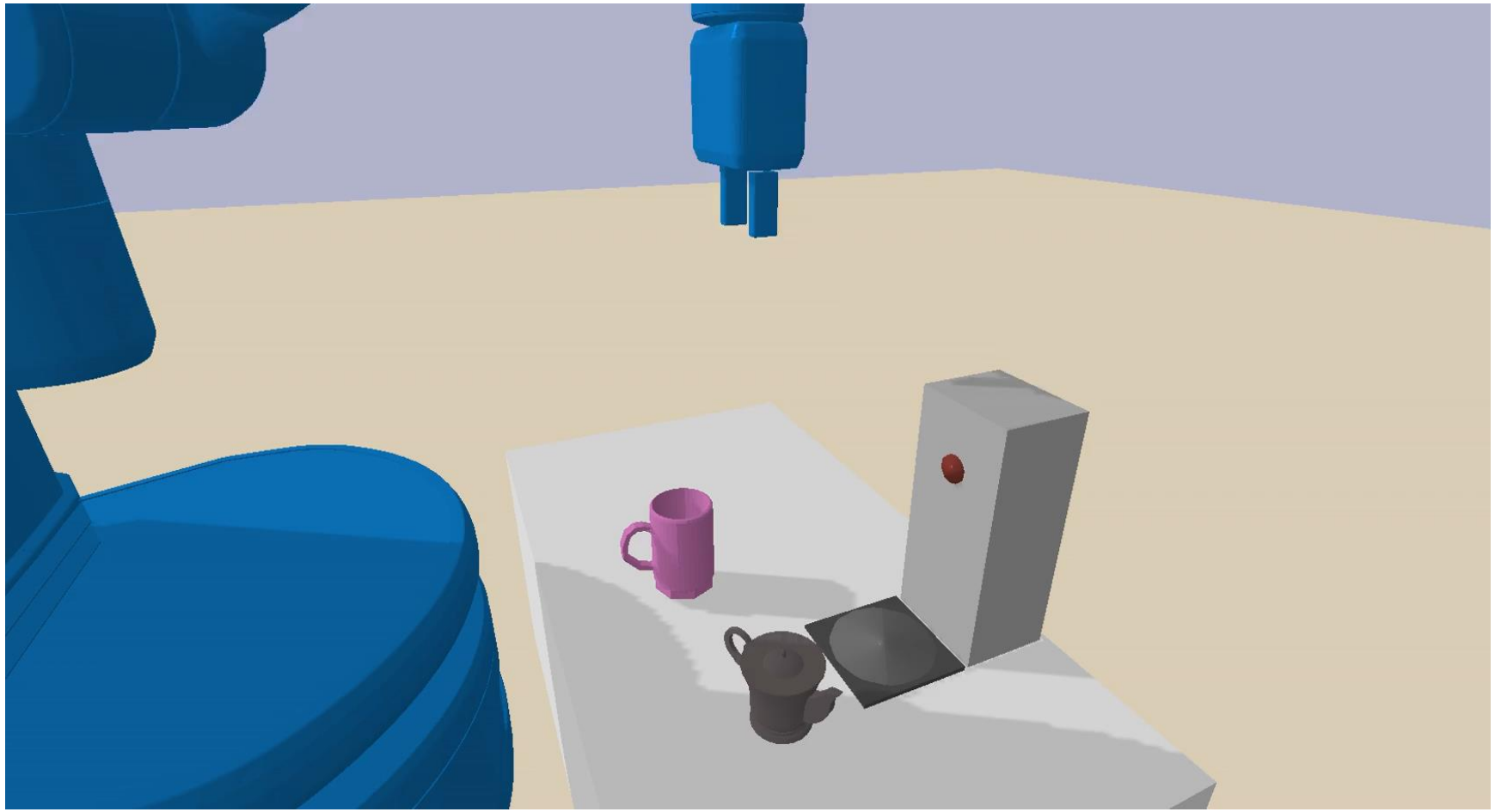
- Grasp handle
- Place on plate
- Turn plate on
- Pick up pot
- Pour into cup



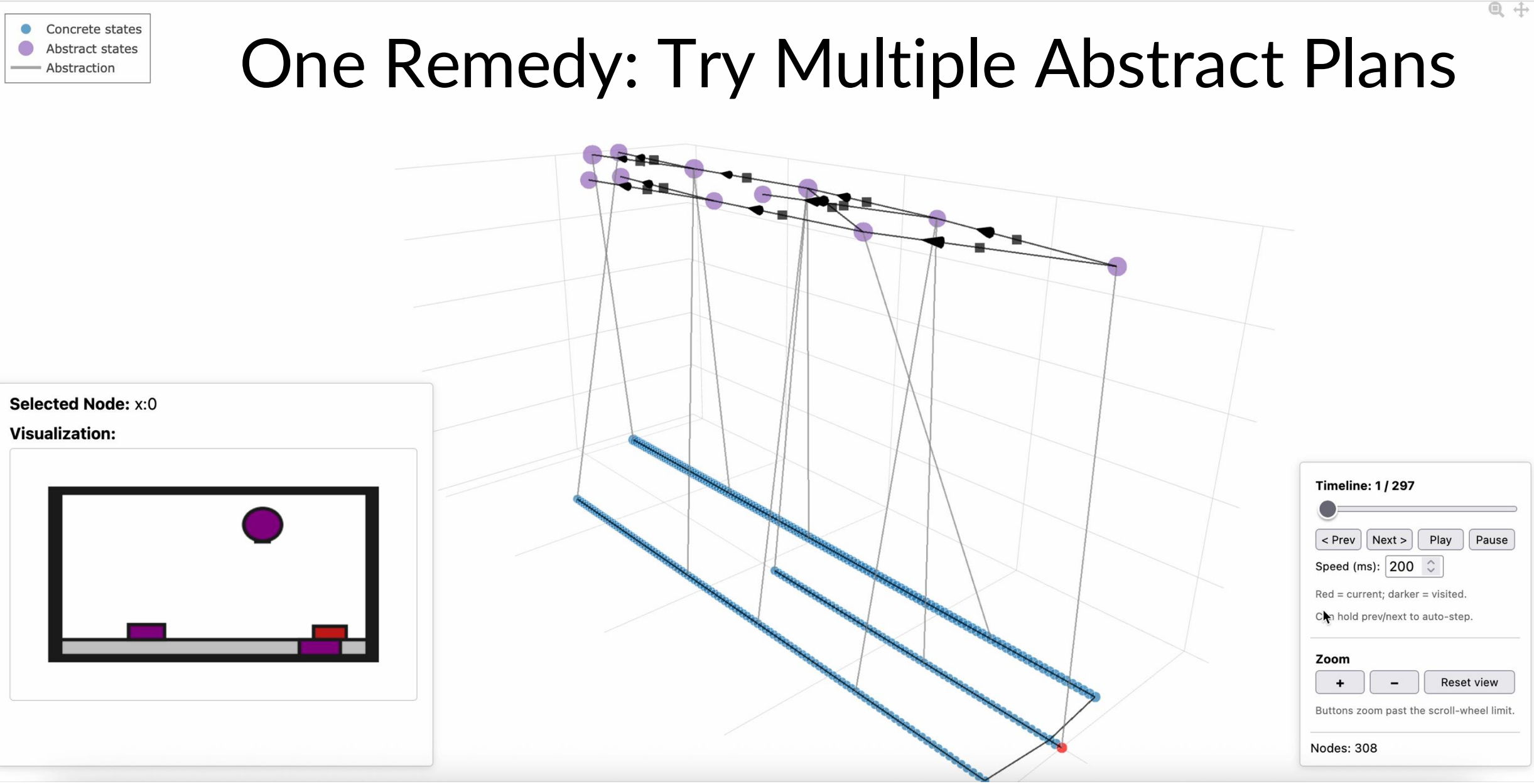
Coffee Domain

Abstract Plan

- Rotate pot
- Grasp handle
- Place on plate
- Turn plate on
- Pick up pot
- Pour into cup



One Remedy: Try Multiple Abstract Plans

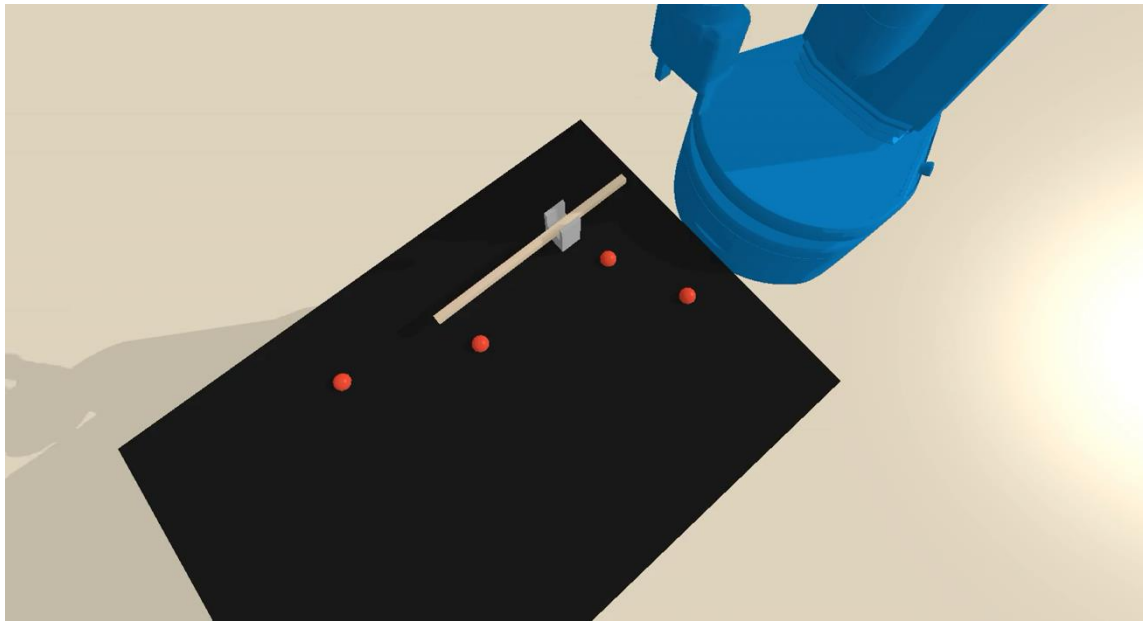


Let's Look at the Code

https://github.com/Princeton-Robot-Planning-and-Learning/prpl-mono/blob/main/bilevel-planning/src/bilevel_planning/bilevel_planners/sesame_planner.py

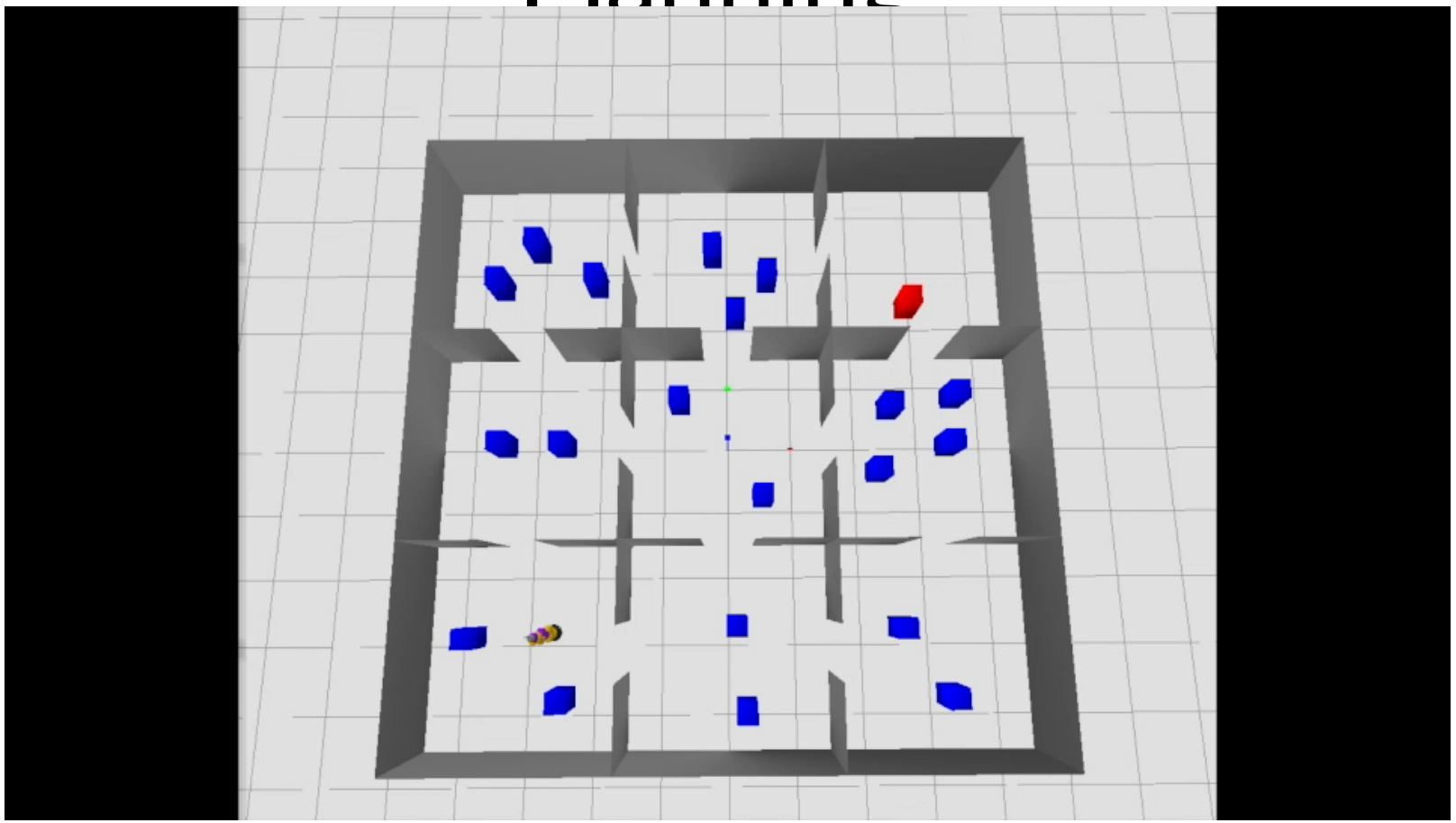
Let's Be Clear About How Terrible Naively Generating Abstract Plans Can Be

`max_abstract_plans`: the maximum number of abstract plans to try before giving up



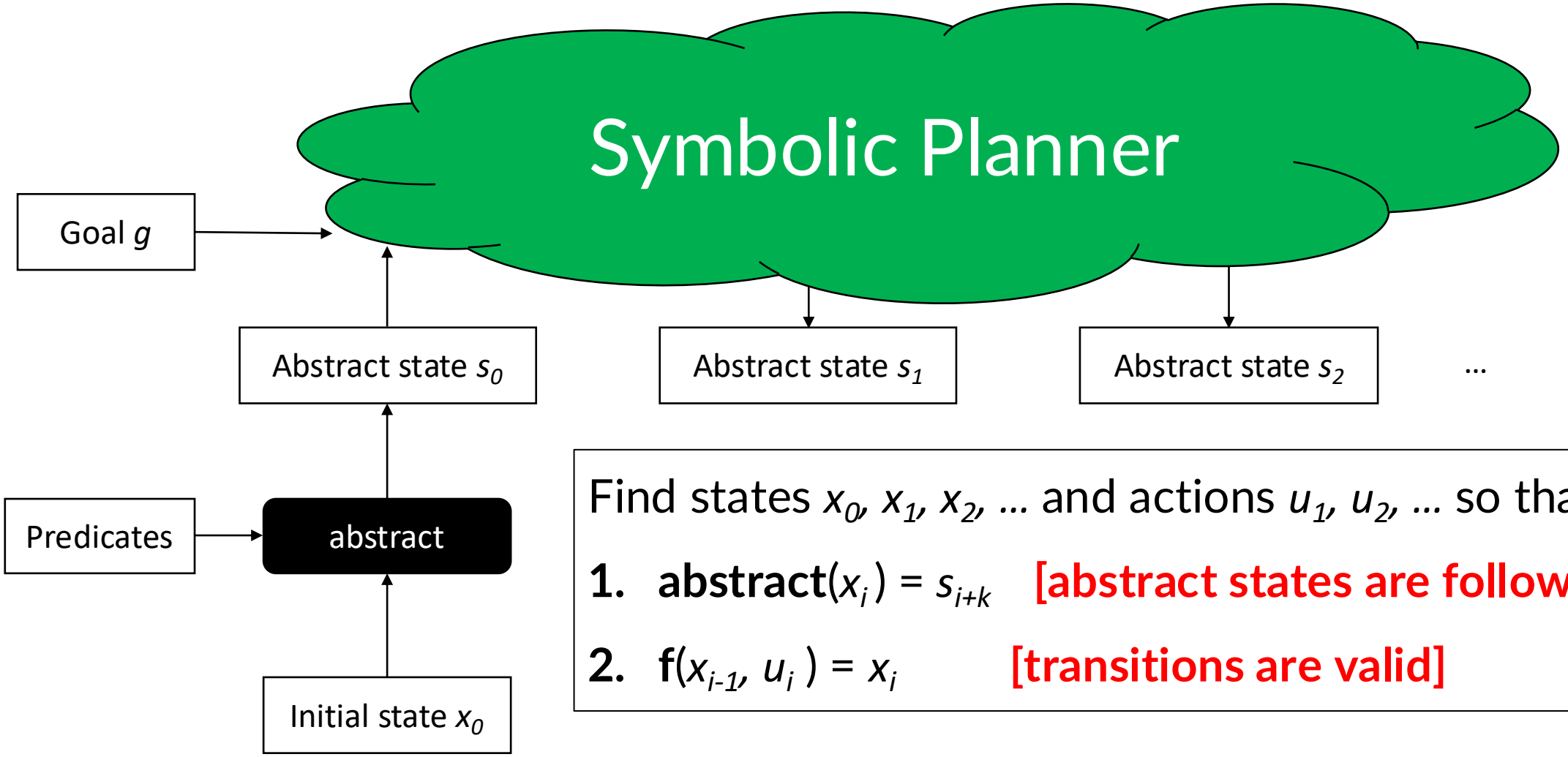
```
Poke(button1)
Poke(button2)
Poke(button3)
Poke(button4)

Well that didn't work…
```

Using Failures to Inform Abstract Planning



See, e.g., “Navigation Among Movable Obstacles” Mike Stilman 2007.

Bilevel Planning: View Abstractions as *Constraints*



Find states x_0, x_1, x_2, ... and actions u_1, u_2, ... so that

1. **abstract**$(x_i) = s_{i+k}$ **[abstract states are followed]**
2. $\mathbf{f}(x_{i-1}, u_i) = x_i$ **[transitions are valid]**

Review: Important Parameters

`max_abstract_plans`: the maximum number of abstract plans to try before giving up

`num_sampling_attempts_per_step`: the number of samples to draw at each step before backtracking

Morning Agenda

- From Task Planning to Bilevel Planning
- **Introduction to Motion Planning**
- Multi-Modal Motion Planning and Beyond
- TAMP in the Real World (TipTop)

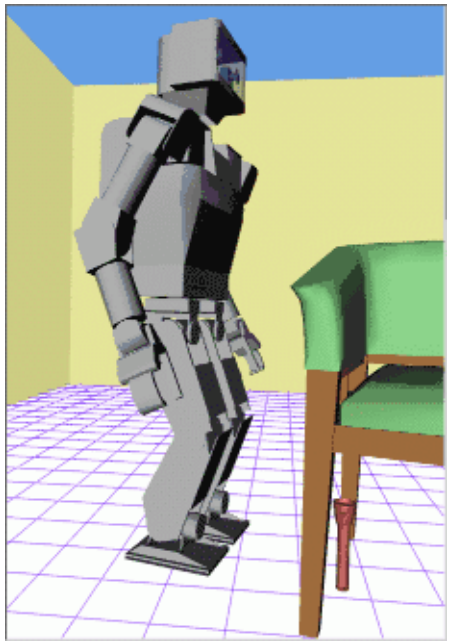
How Should We Choose Abstractions?

To start, let's focus on **motion in free space**

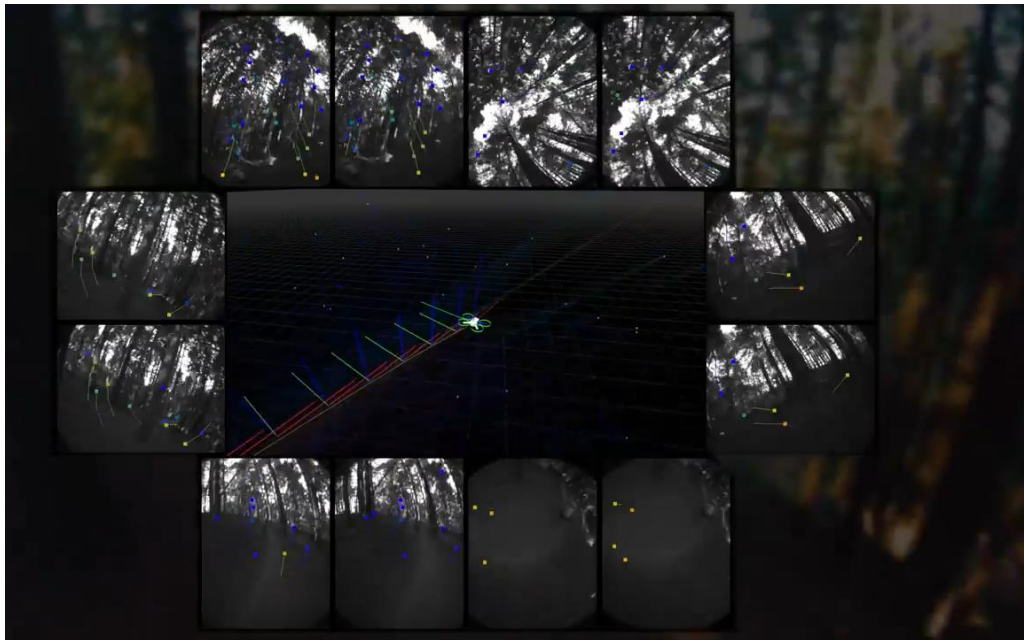
“We have a brain for one reason and one reason only — and that's to produce adaptable and complex movements.”

Daniel Wolpert
TED Talk, 2011

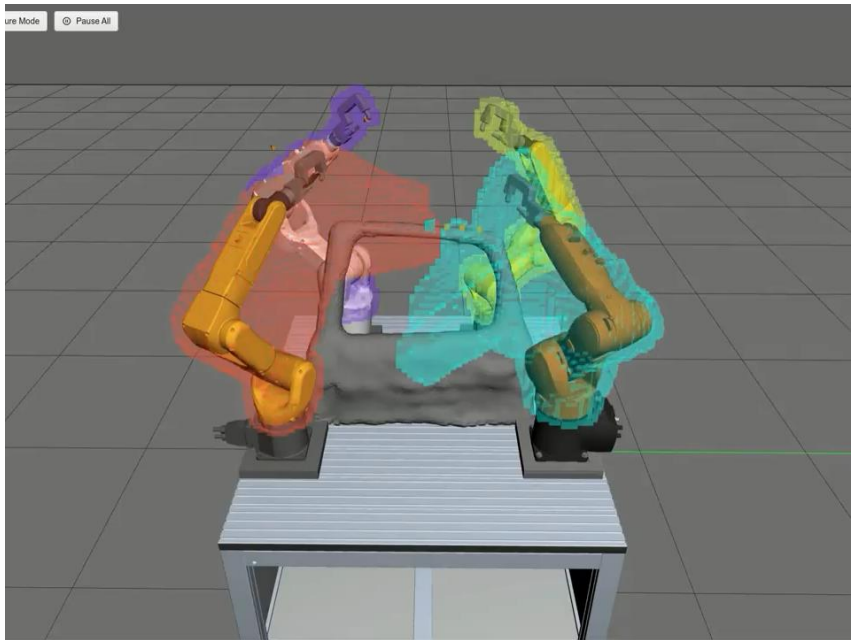
Motion Planning



Kuffner (2002)



Skydio (2019)



Realtime Robotics (2023)

Motion Planning Problem Setting

A motion planning problem includes:

- A *configuration space* $\mathcal{X}$

Must be bounded & have distance metric & have other nice properties…

- An initial configuration $x_0 \in \mathcal{X}$
- A goal configuration $x_g \in \mathcal{X}$

Alternative definition: *set* of configurations

- A *feasibility check* $f : \mathcal{X} \rightarrow \{ \mathrm{T}, \mathrm{F} \}$

Usually: feasible (T) if robot not in collision

Motion Planning Solution

A *solution* to a motion planning problem is a trajectory

$$\alpha: [0, H] \to \mathcal{X}$$

where

1. $\alpha(0) = x_0$
2. $\alpha(H) = x_g$
3. The robot can follow the trajectory

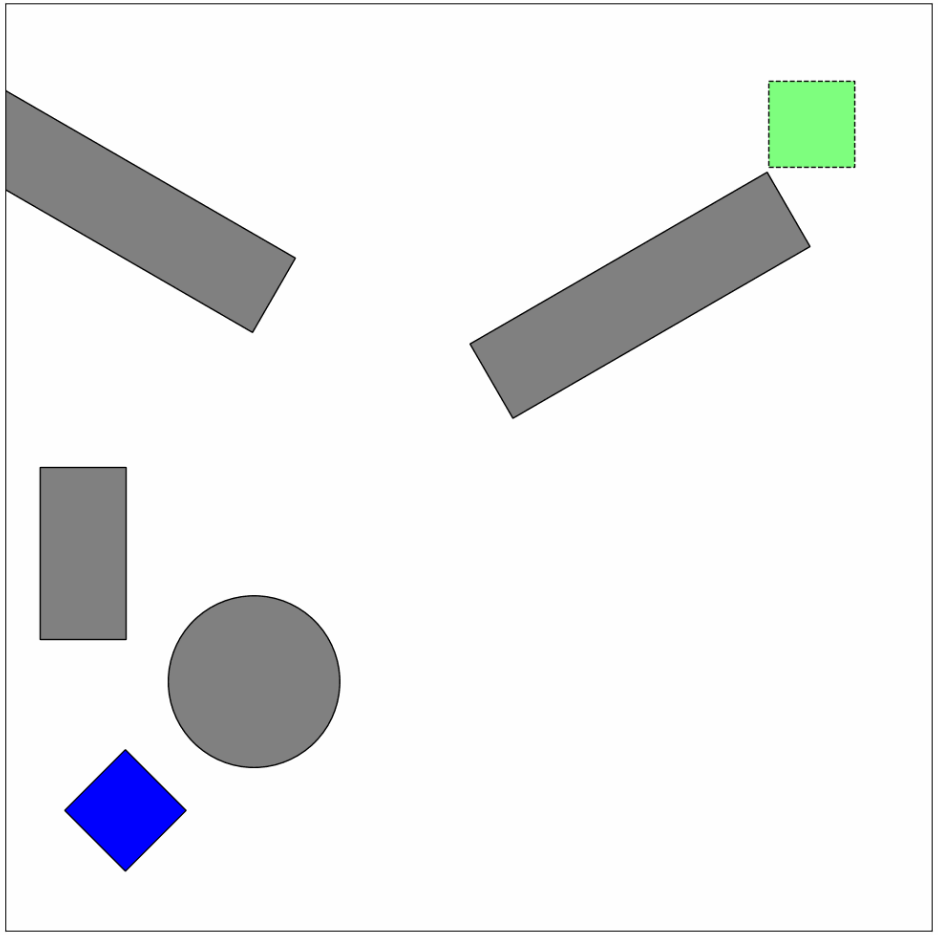
Continuous states *and* continuous time!

Many different ways to specify this

Actions are implicit

Example: Motion Planning with 2D Shapes

- Configuration space:
  - (x position, y position, rotation)
  - x and y position are bounded
  - Subset of SE(2)
- Initial configuration: see image
- Goal configuration: see image
- Feasibility check:
  - Configuration is feasible if robot is not in collision with obstacles



A Stupidest Possible Algorithm

1. Discretize the configuration space

2. Run path planning (e.g., A* with distance-to-goal heuristic)

When will this go badly?

Is this sound, complete, optimal?

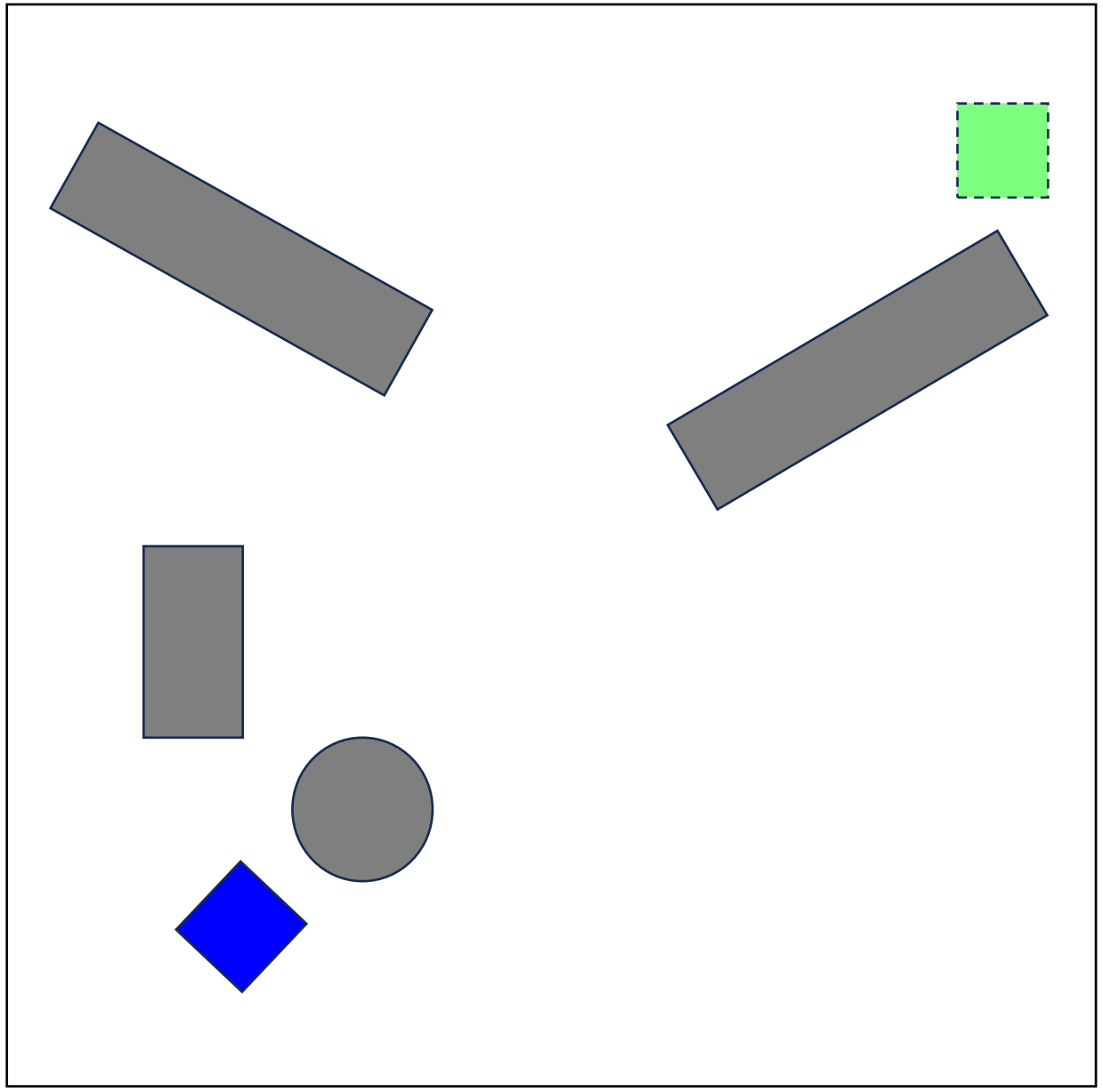
Sampling-Based Motion Planning

Let's assume we can sample from the configuration space

Rapidly Exploring Random Trees (RRT)

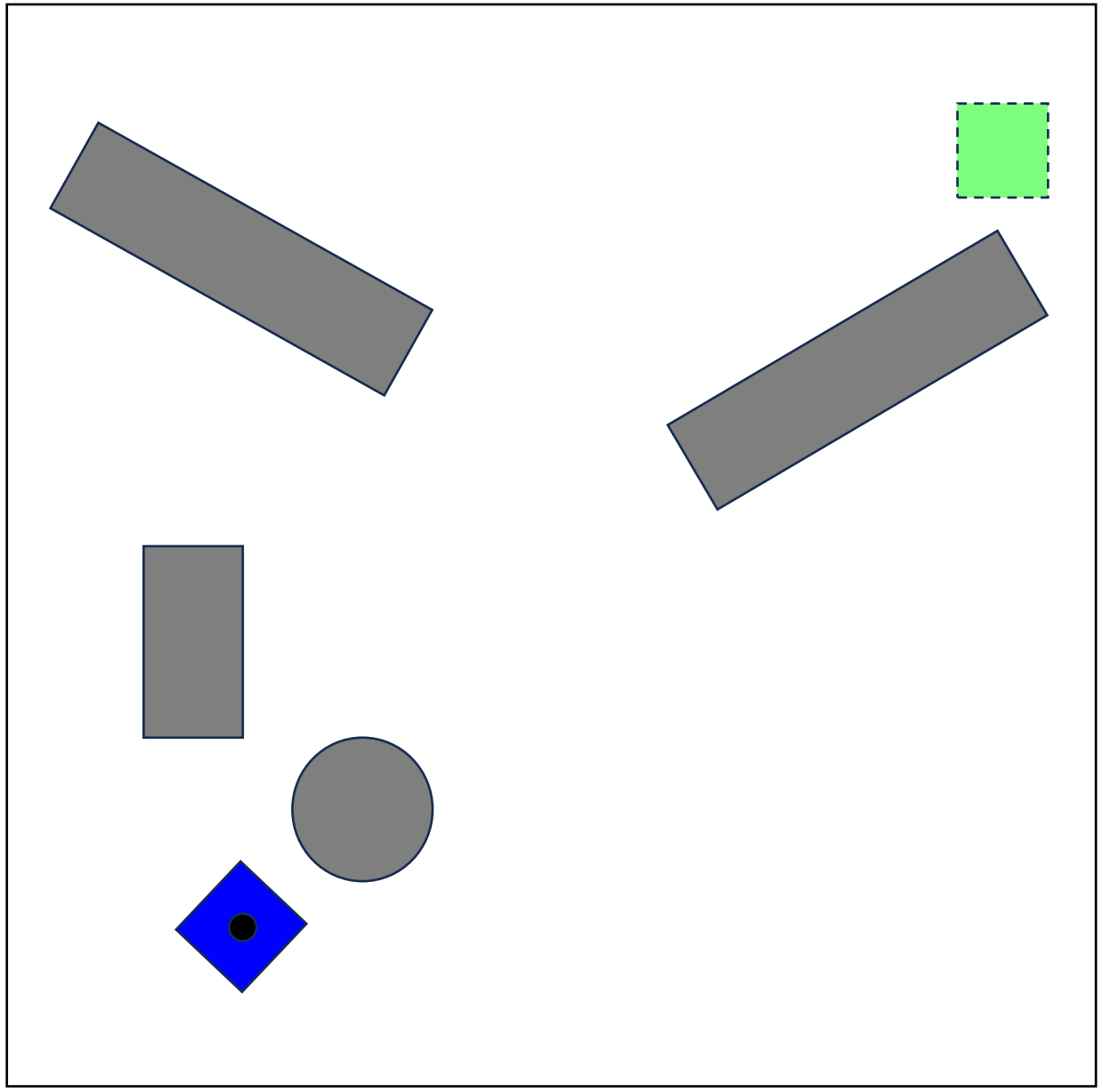
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



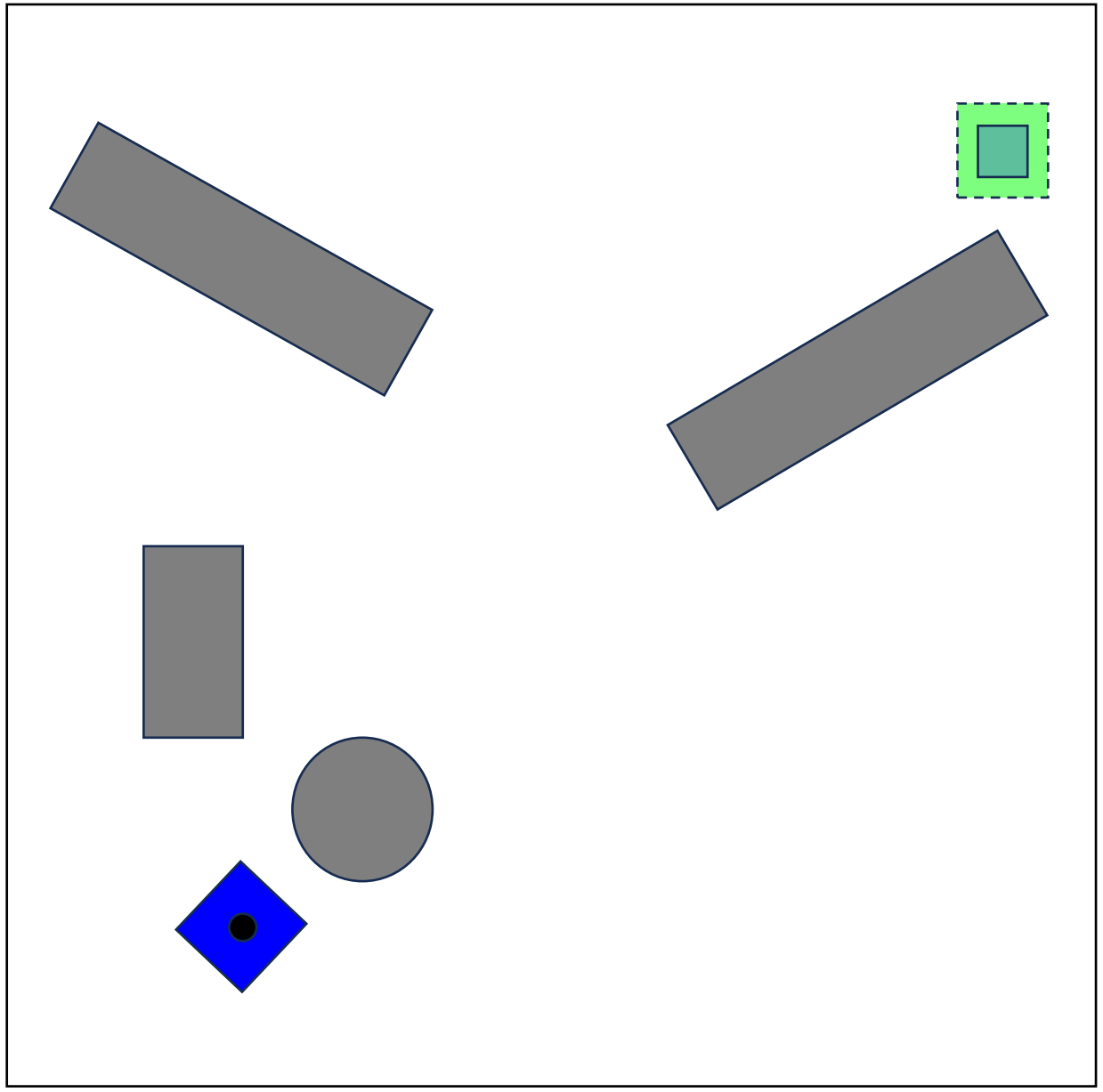
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



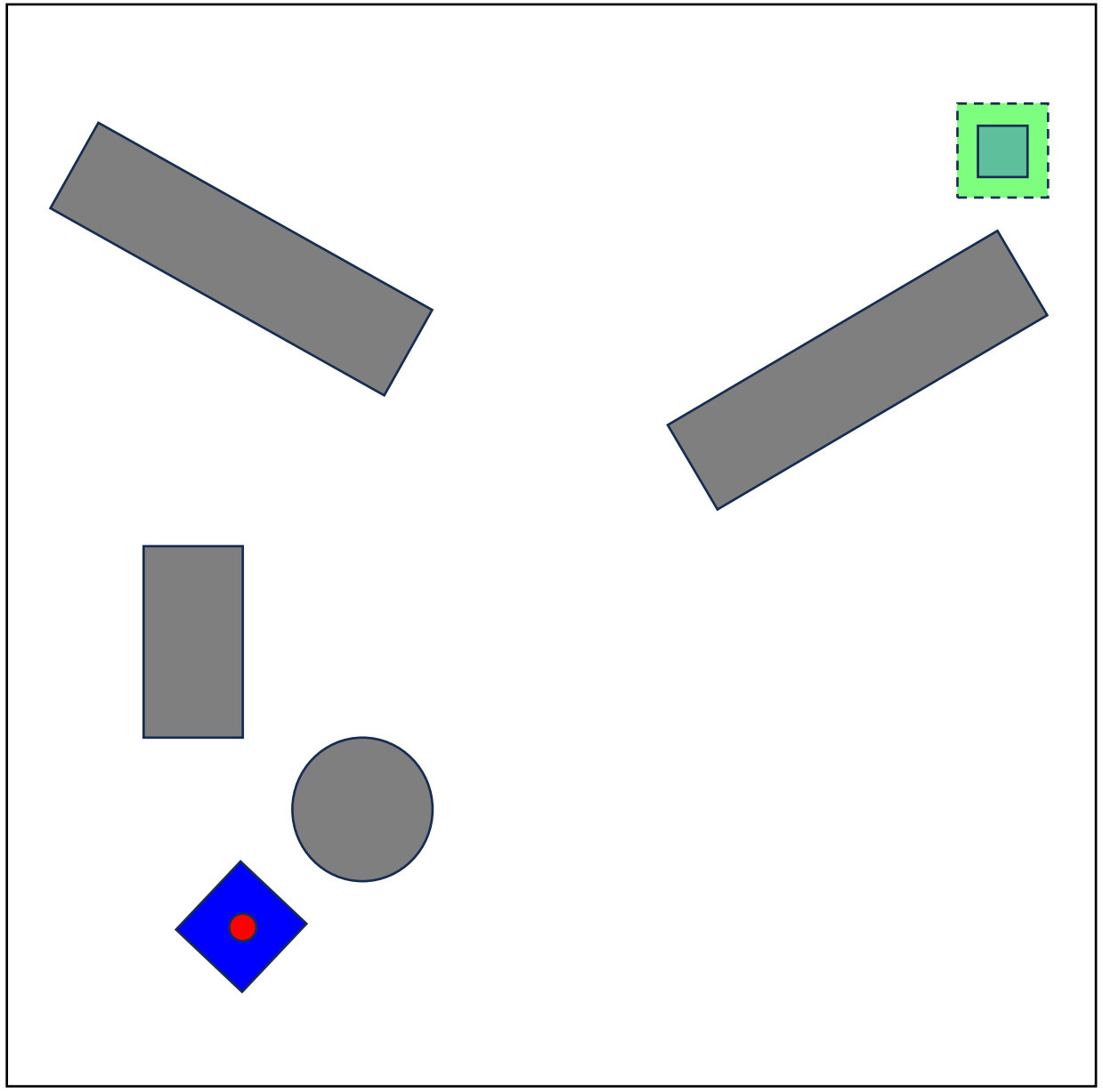
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



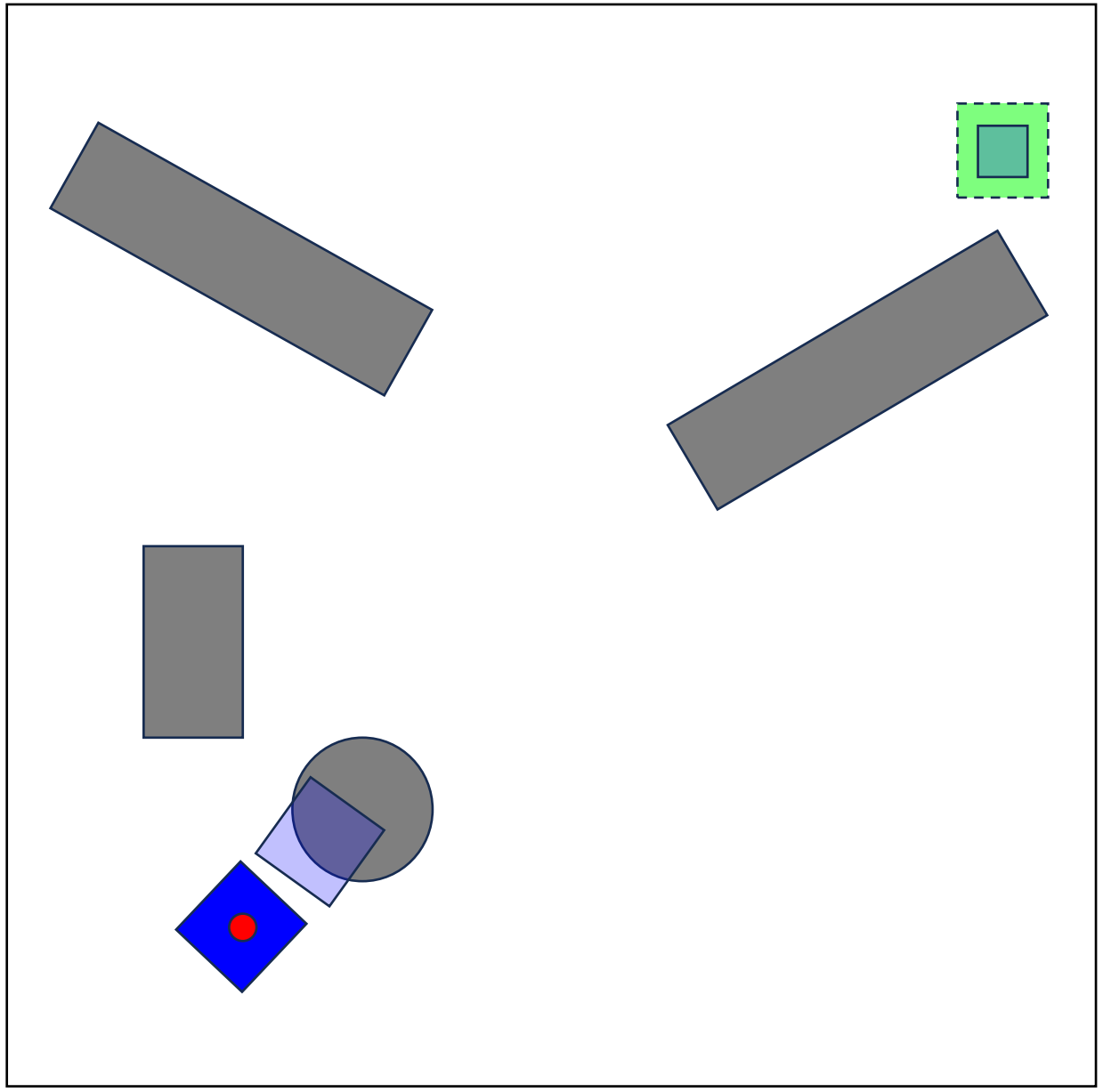
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



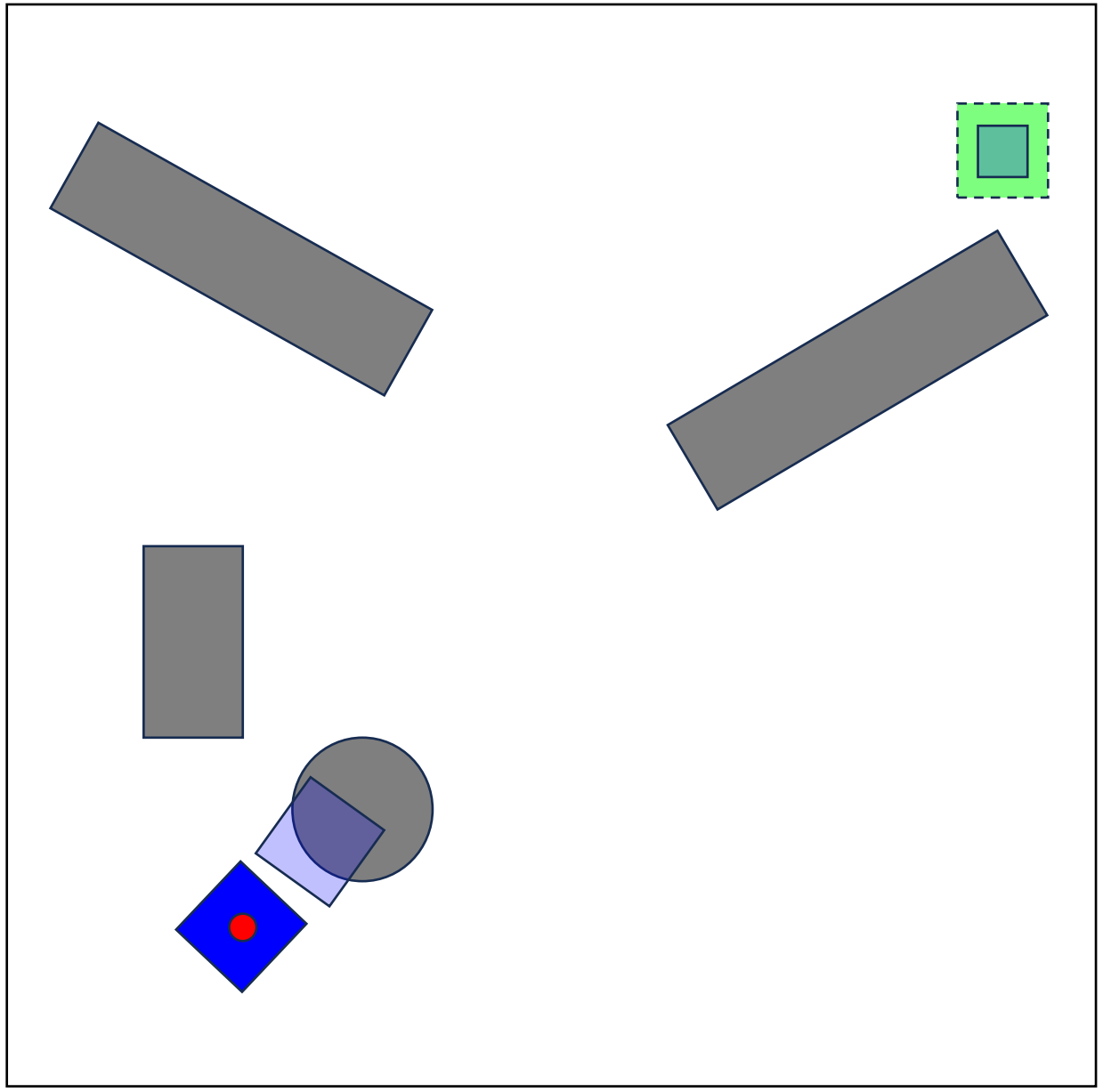
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



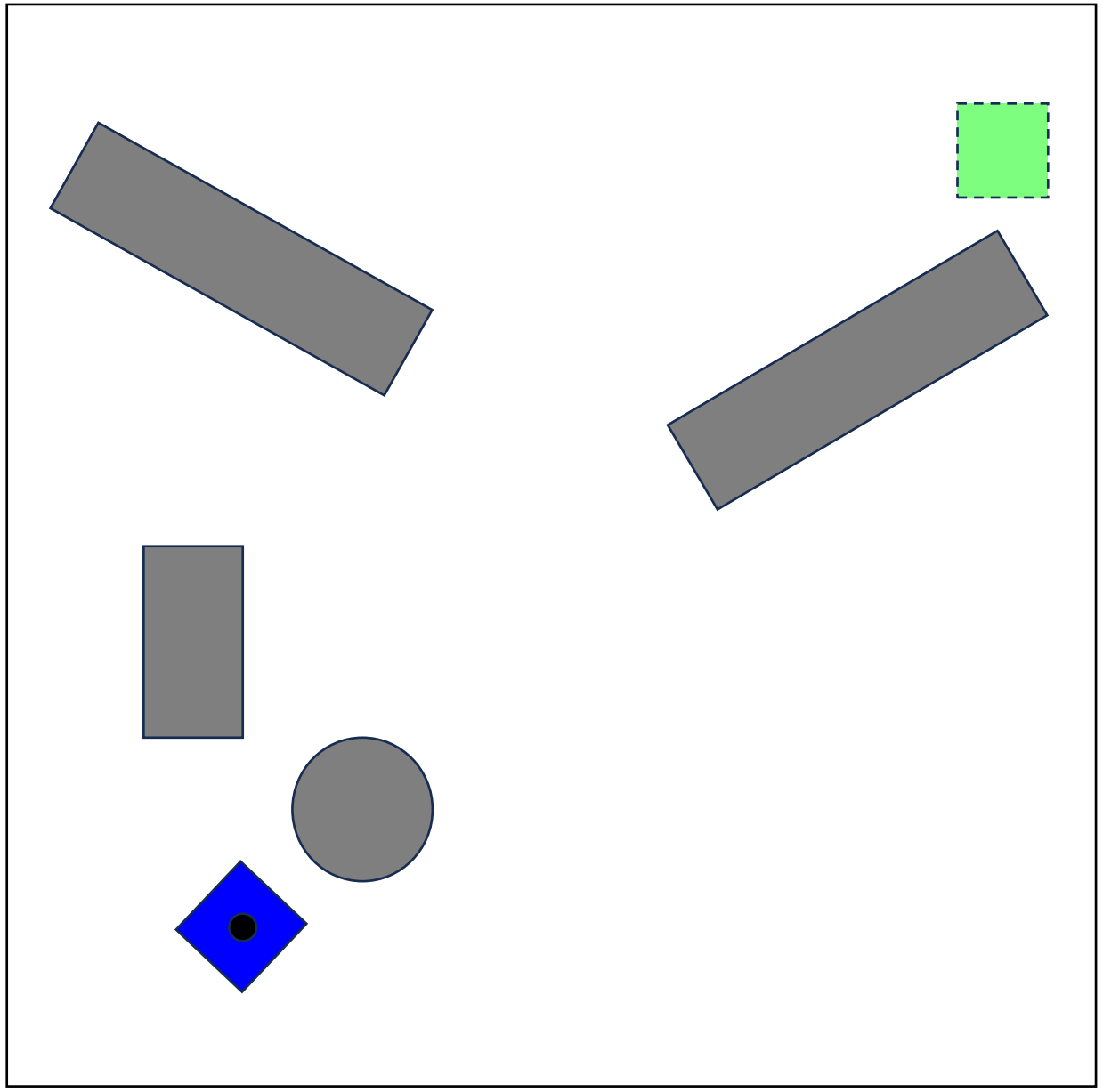
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



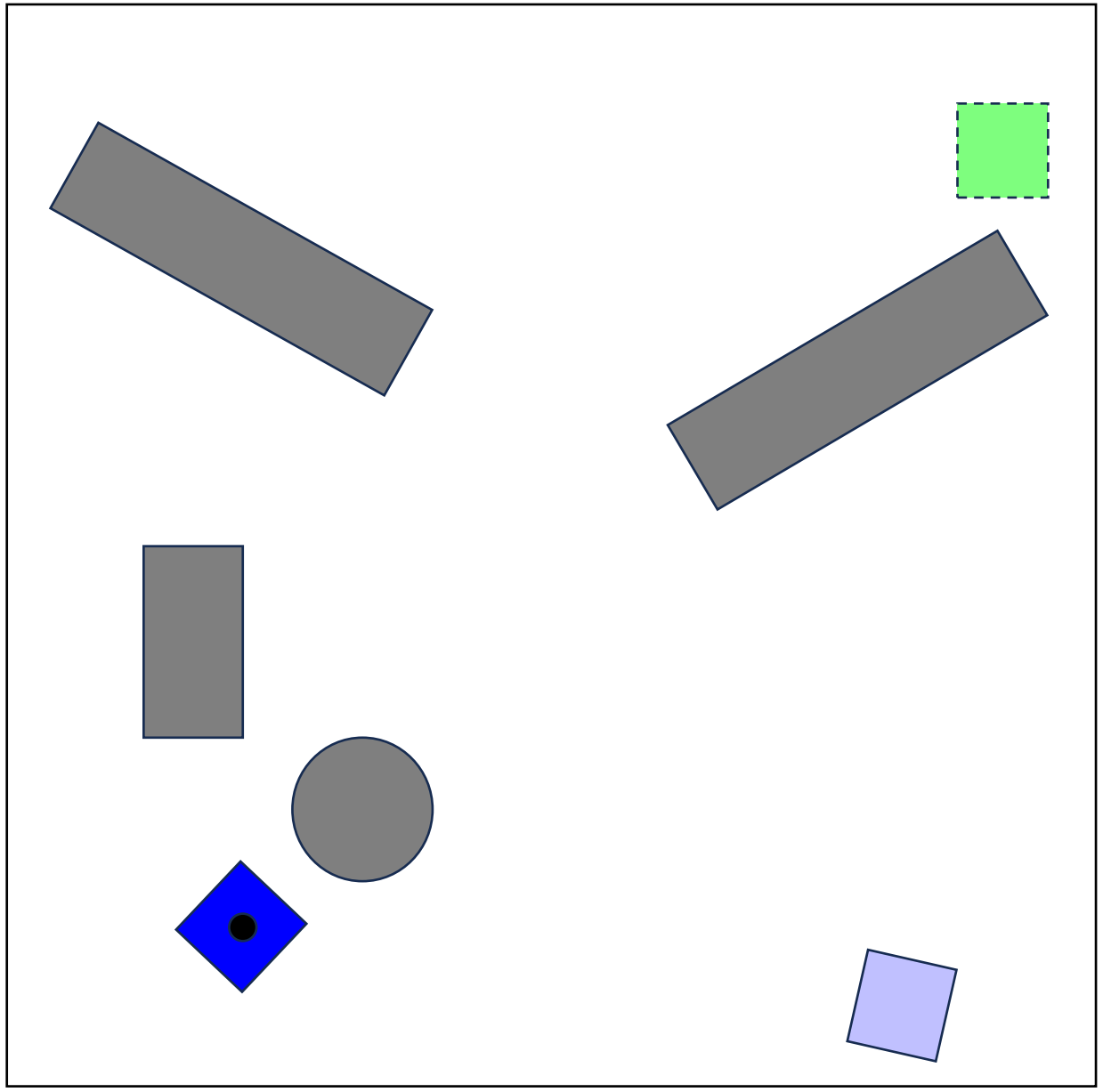
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



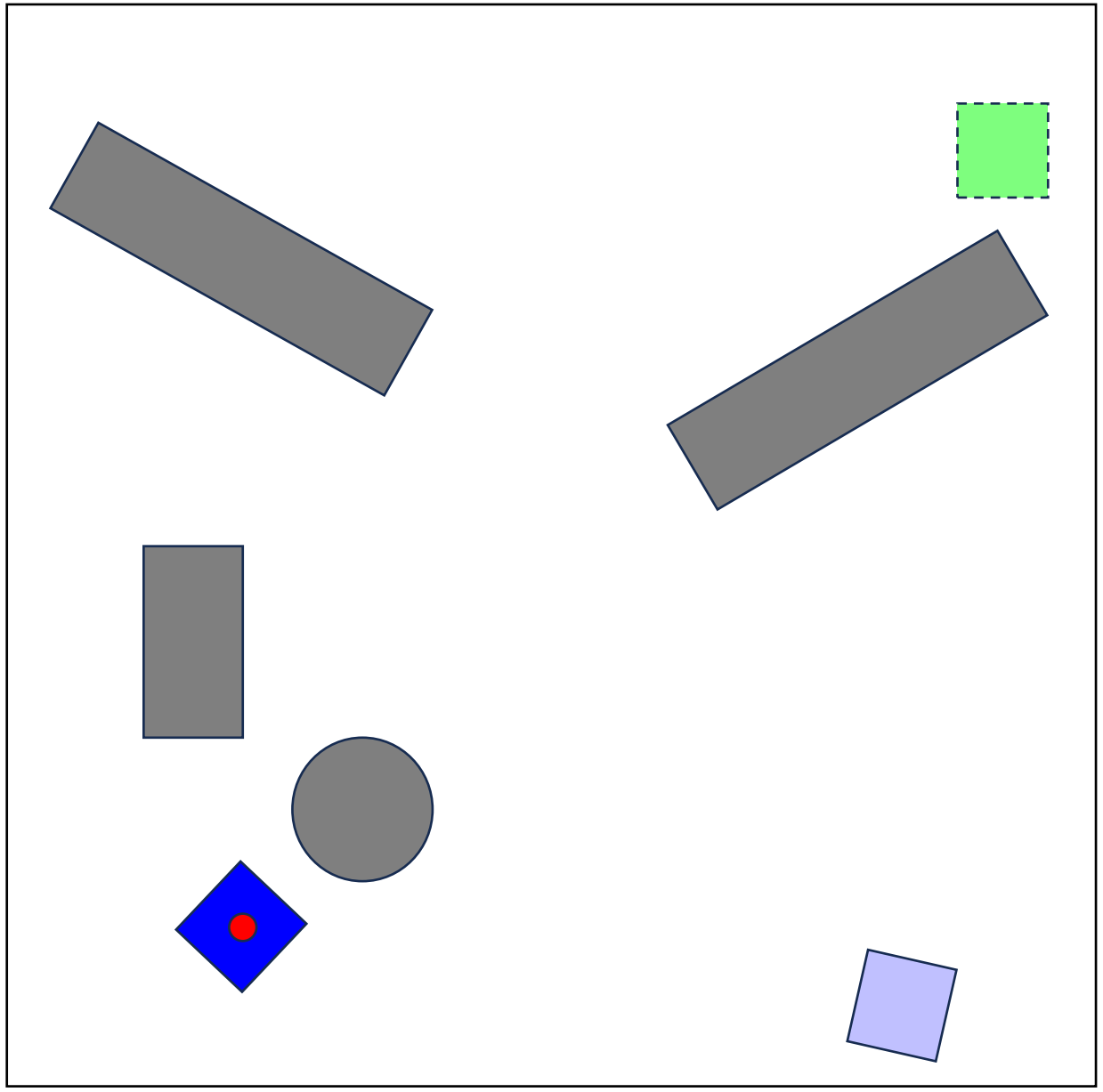
RRT(x_0, x_g, $\mathcal{X}$, f)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



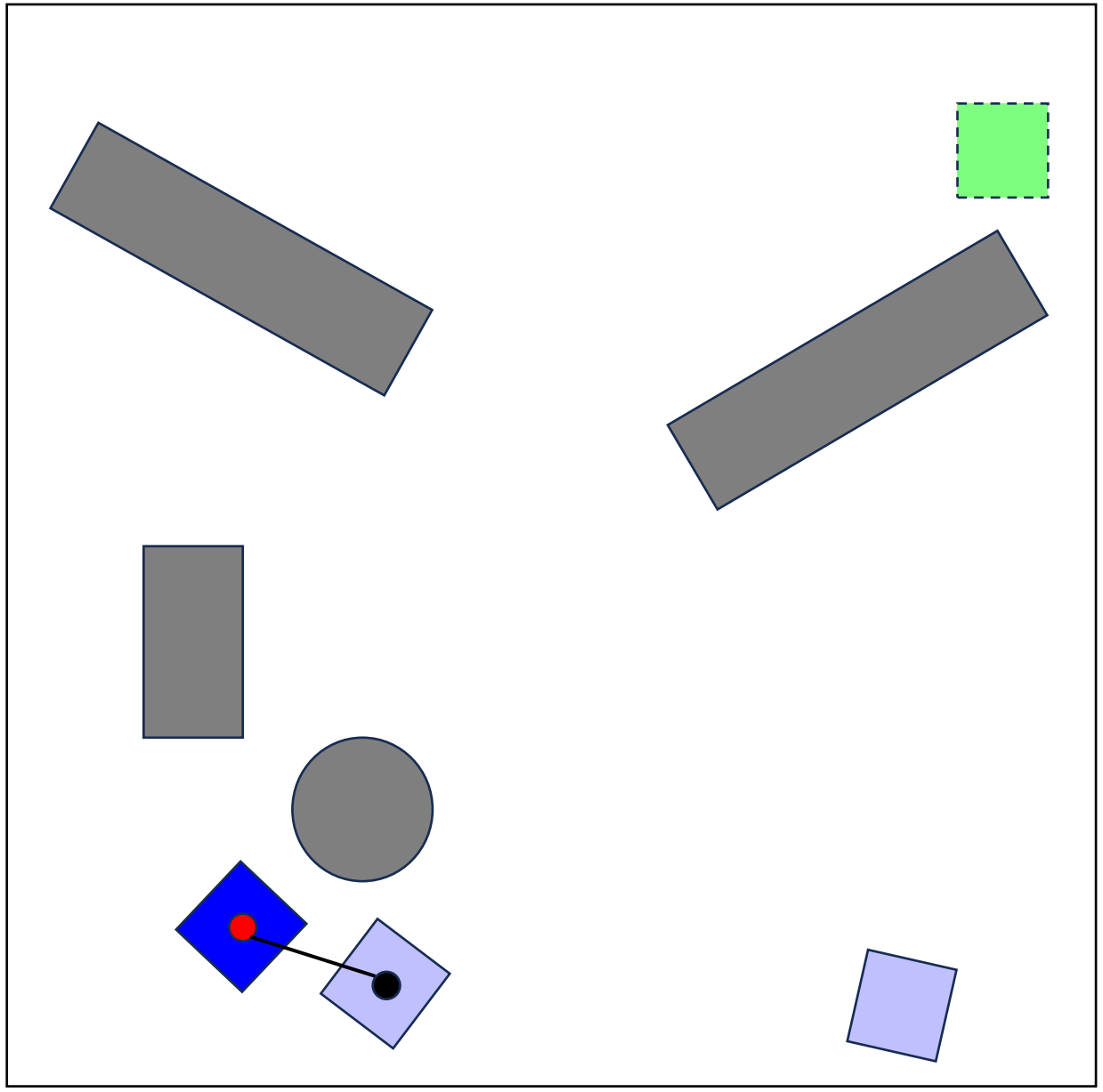
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



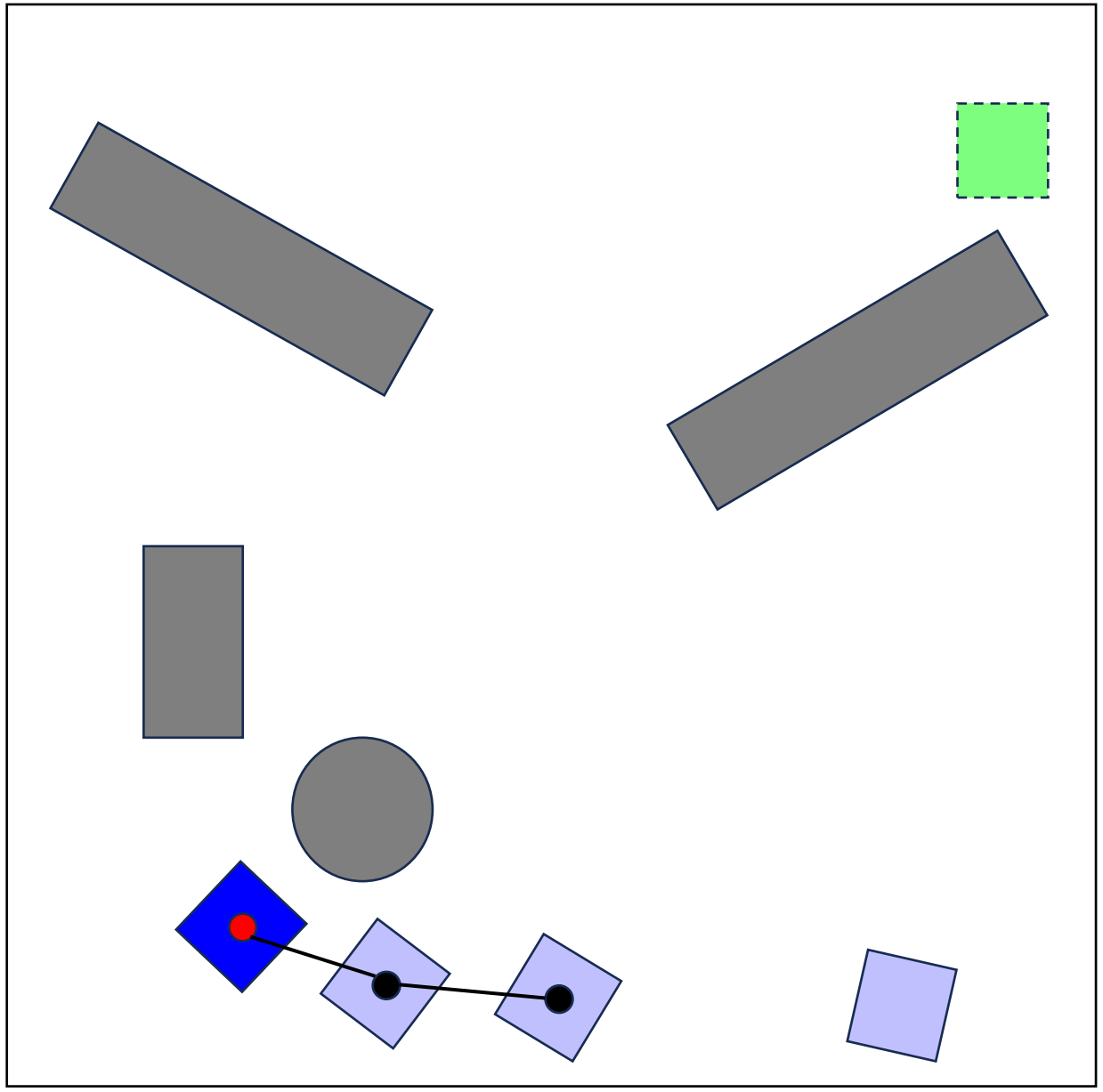
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



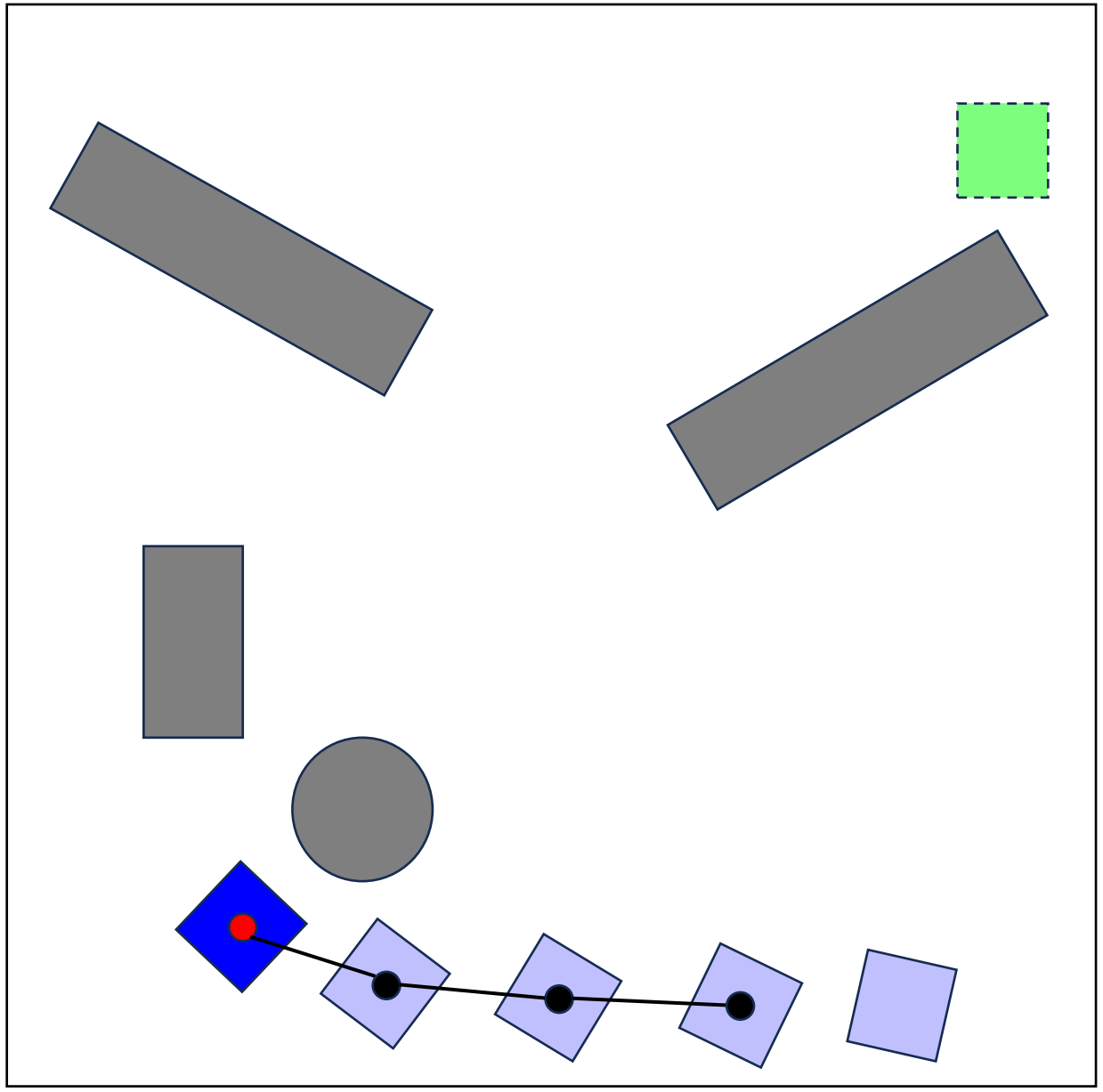
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



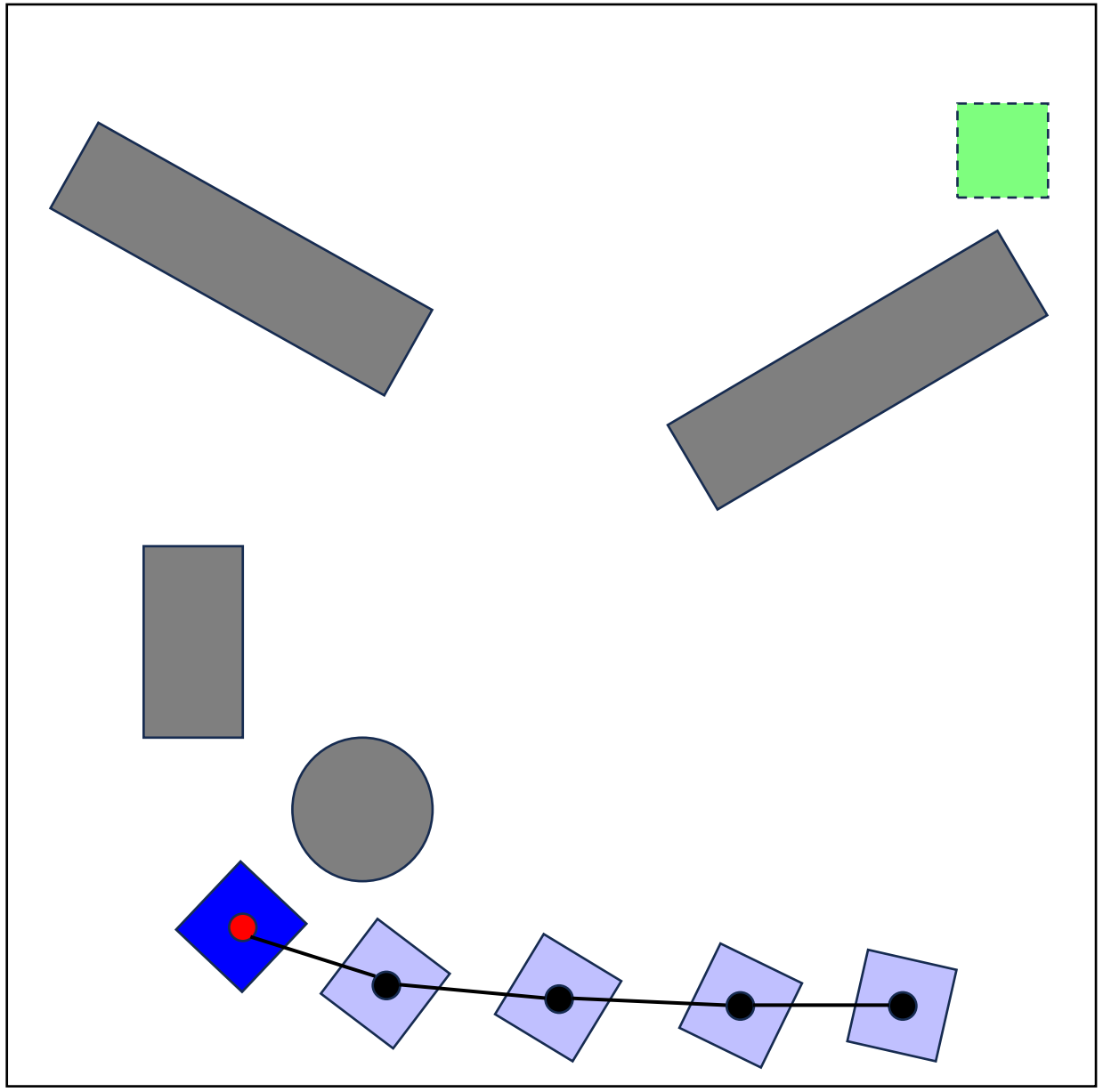
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



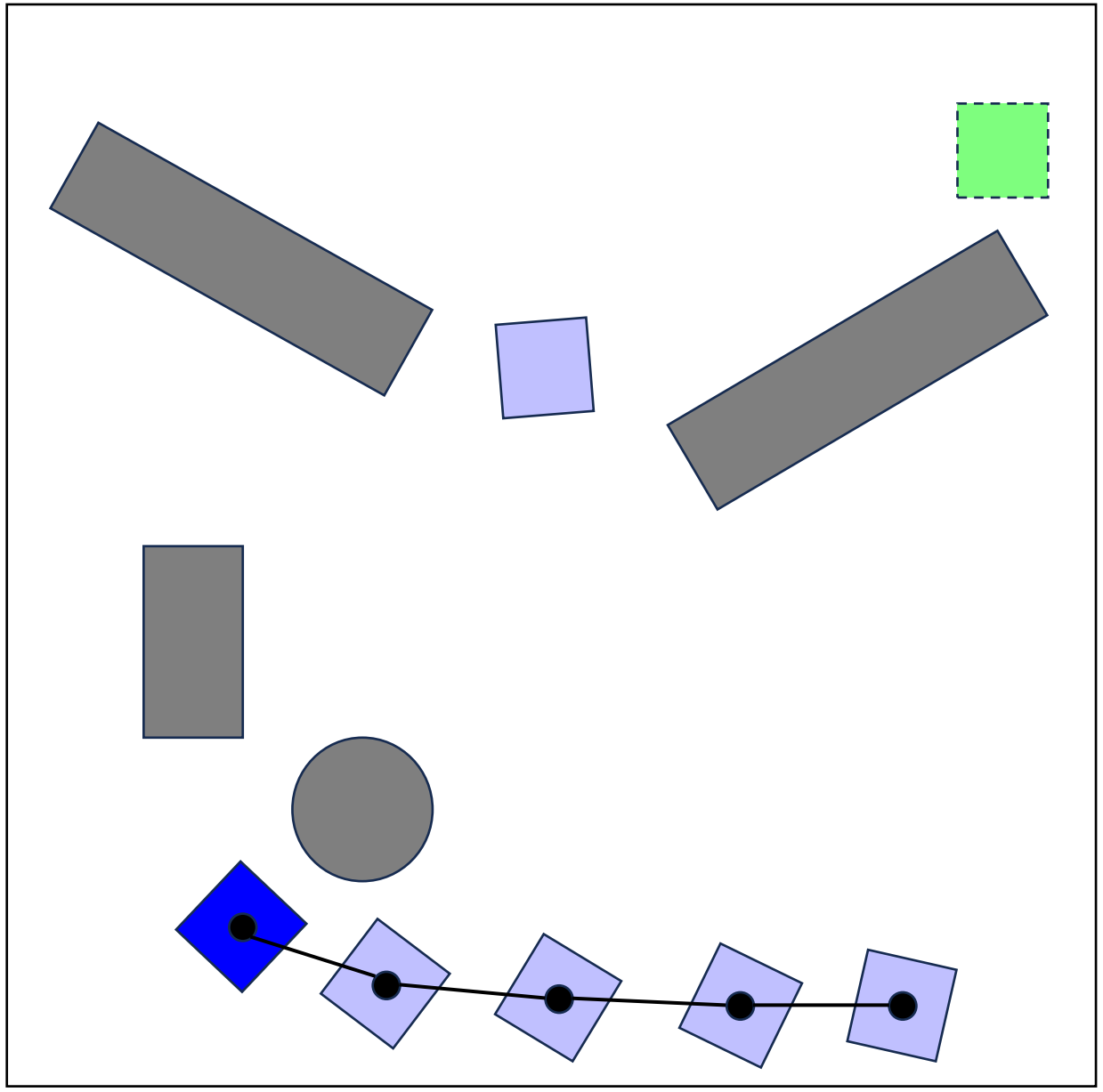
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



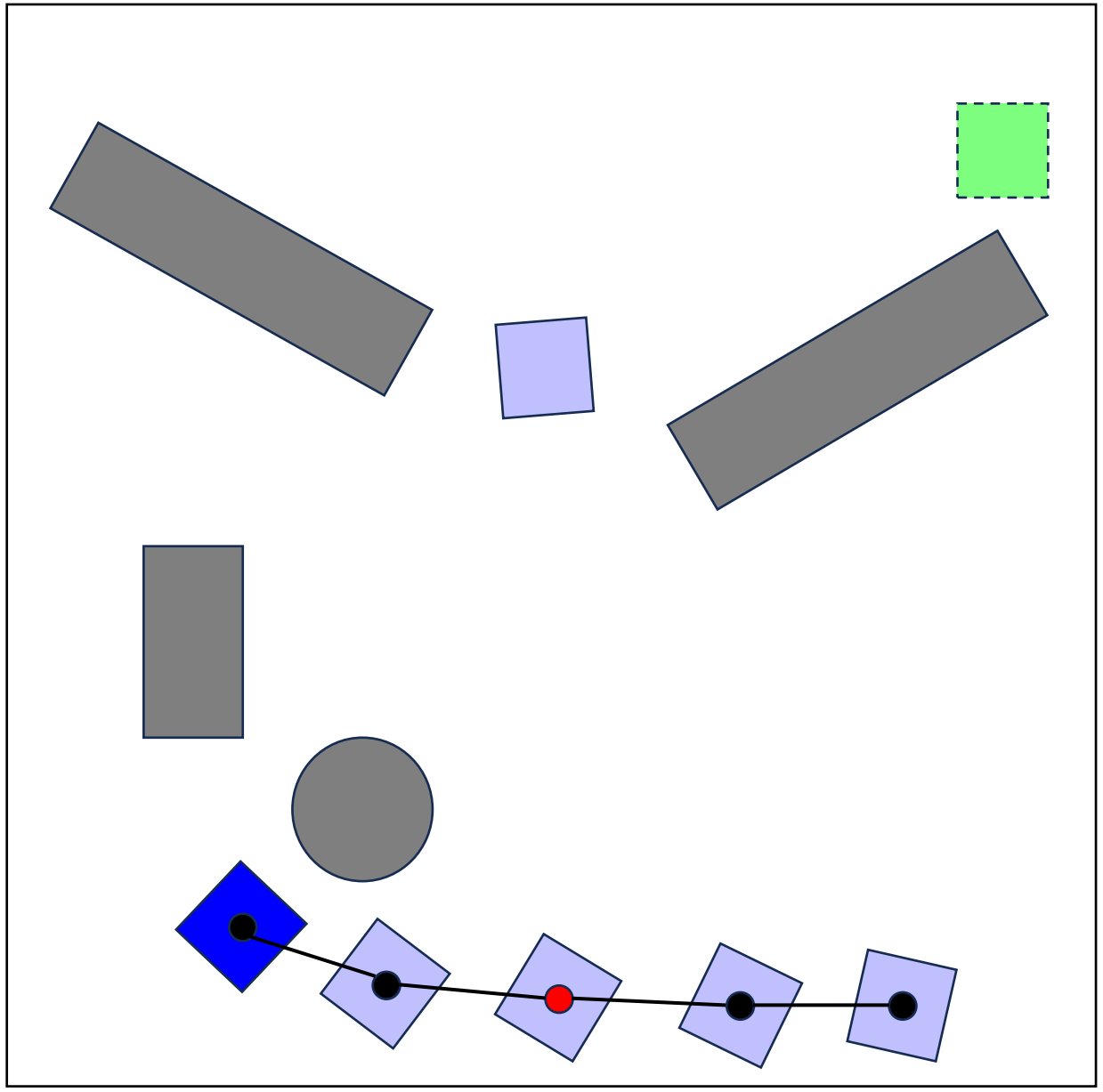
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



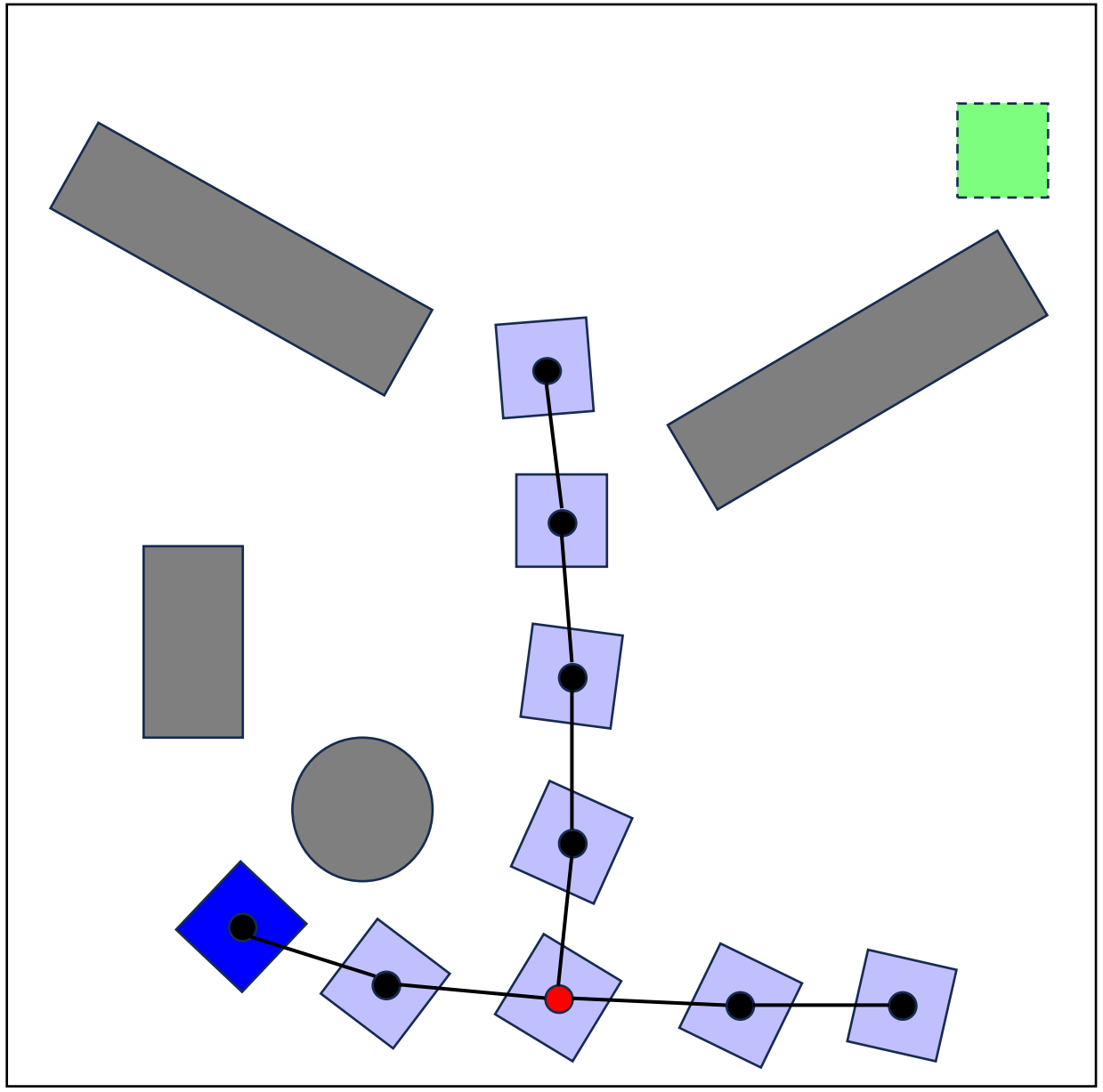
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



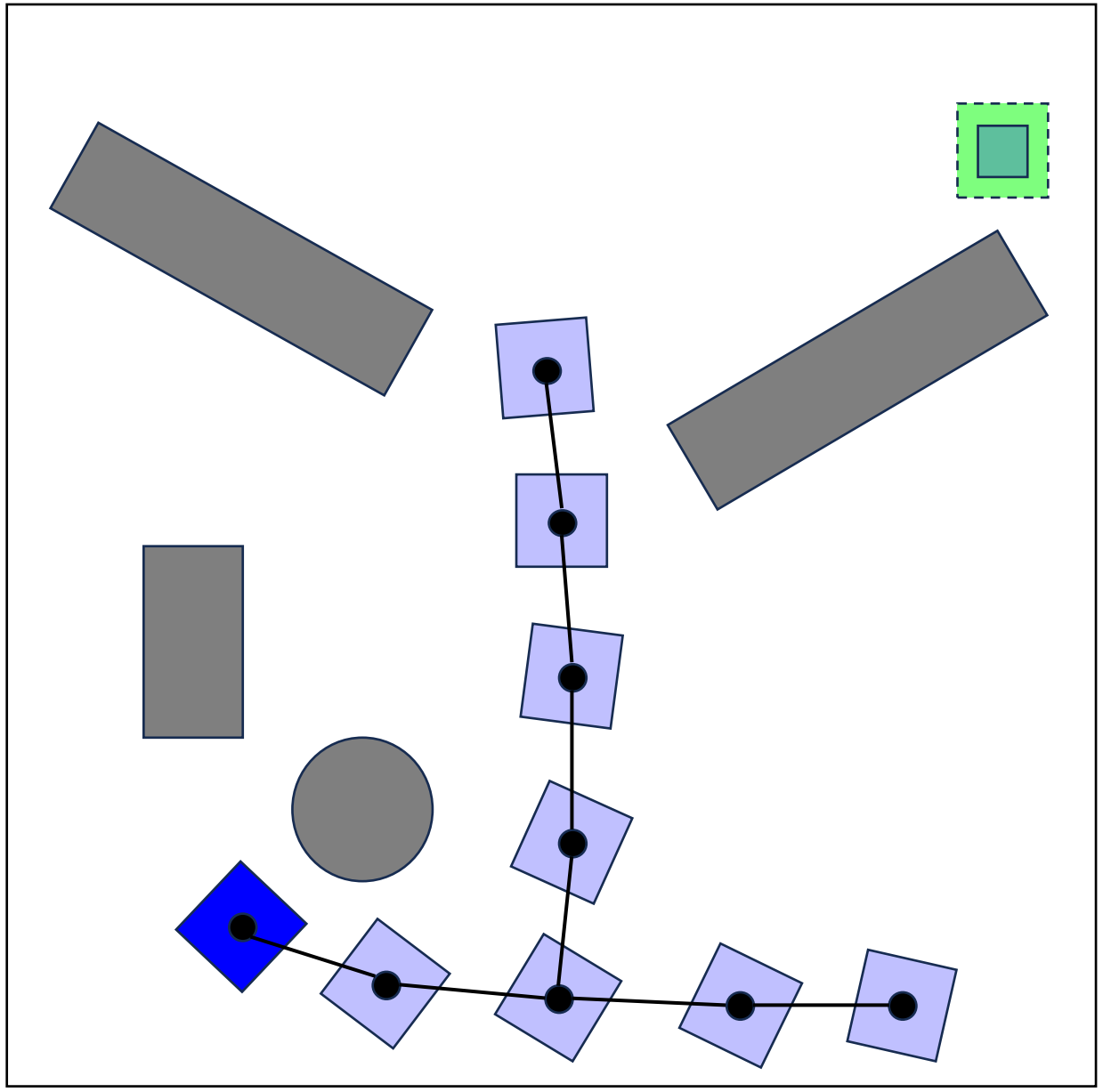
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



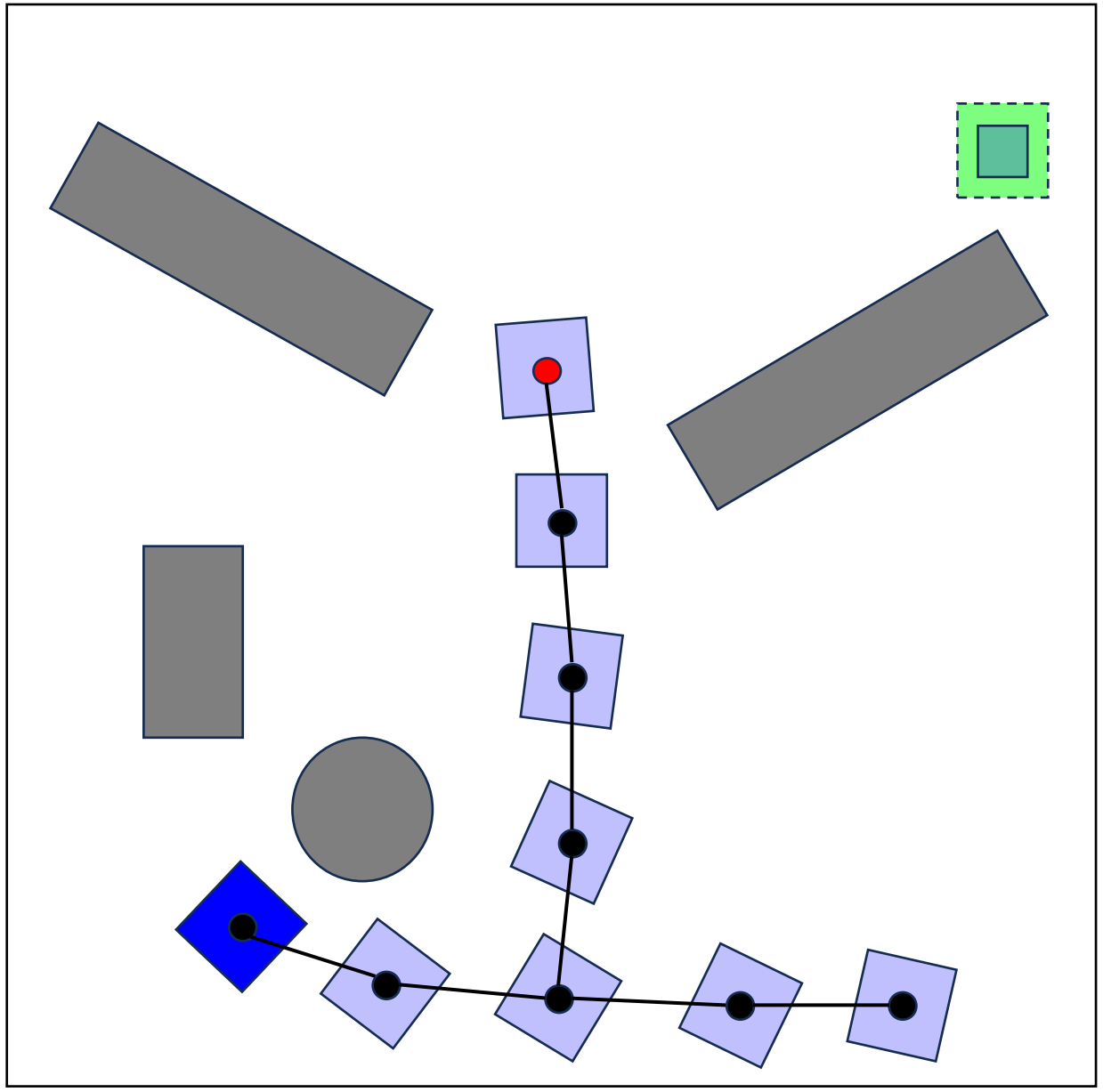
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



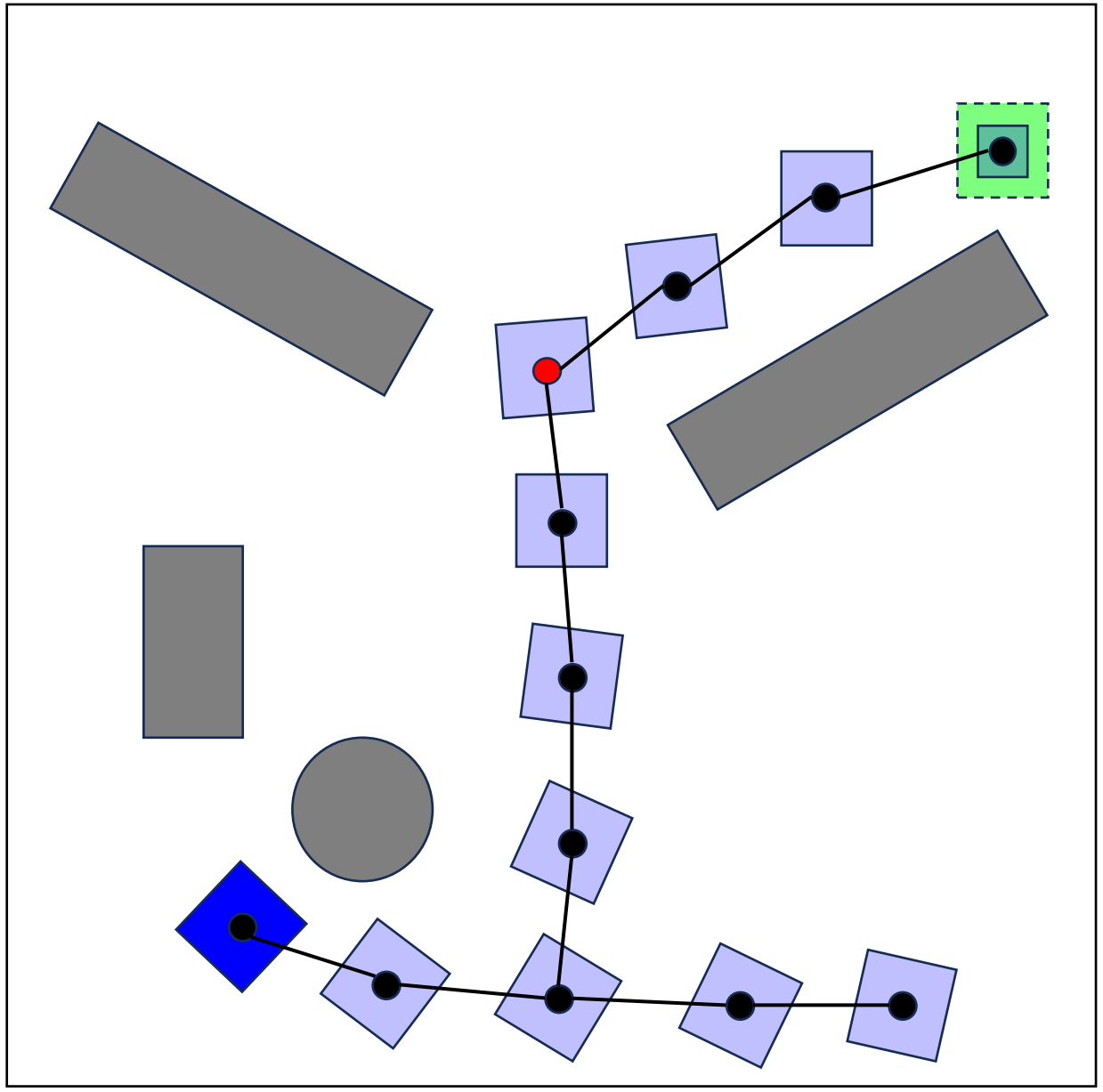
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



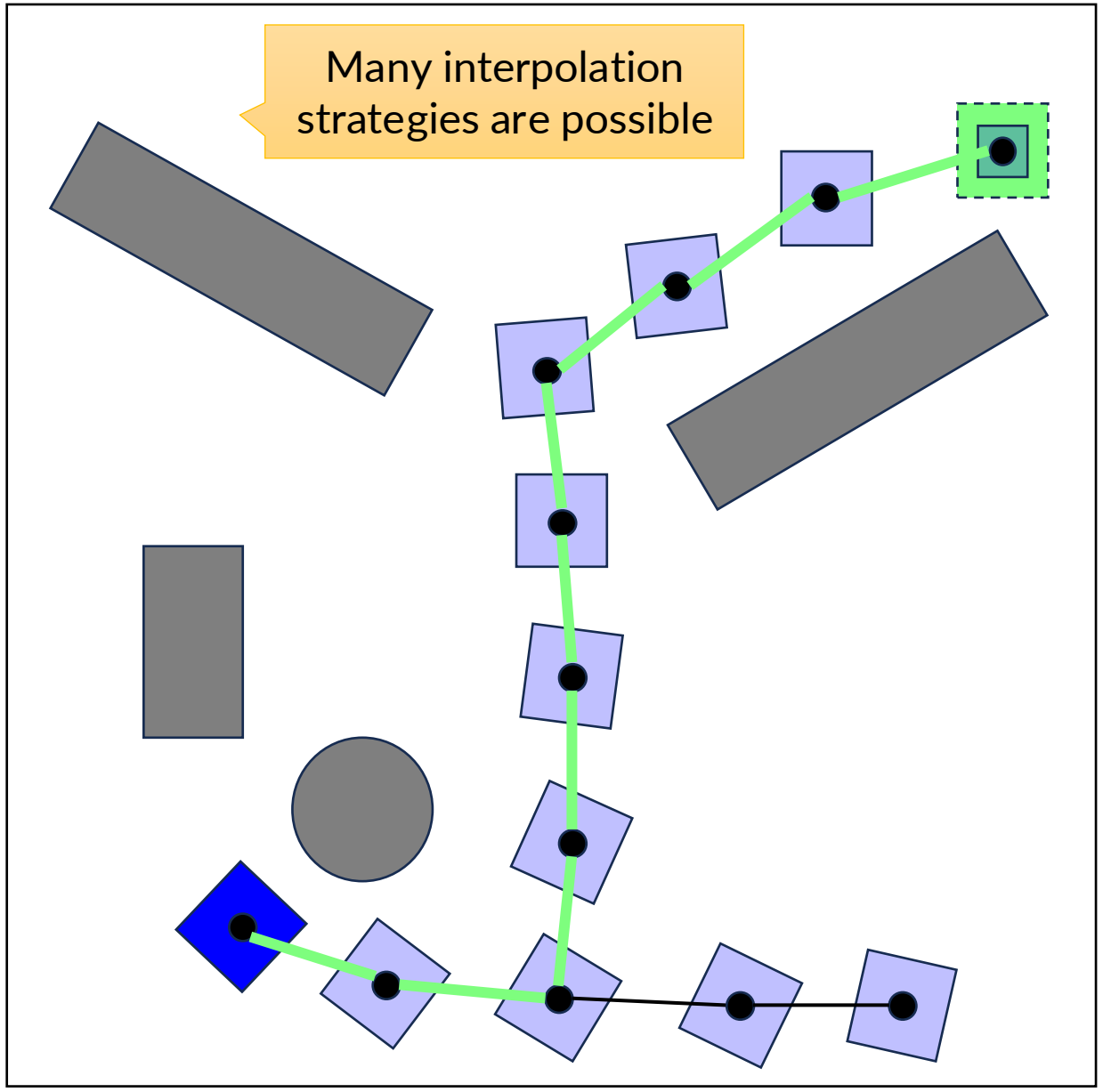
RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```



RRT($x_0, x_g, \mathcal{X}, f$)

```
1   // Initialize tree at x_0
2   nodes = [Node(x_0)]
3   repeat:
4       if uniform() < goalSampleProb
5           // Try to go directly to the goal
6           x_target = x_g
7       else
8           // Sample a target configuration
9           x_target = sample(X)
10      // Extend the tree towards the target
11      node = getClosest(nodes, x_target)
12      x_node = node.conf
13      for x in extend(x_node, x_target)
14          if f(x):
15              nodes.add(Node(x))
16              if x = x_g:
17                  return finish(nodes)
18          else
19              break
```

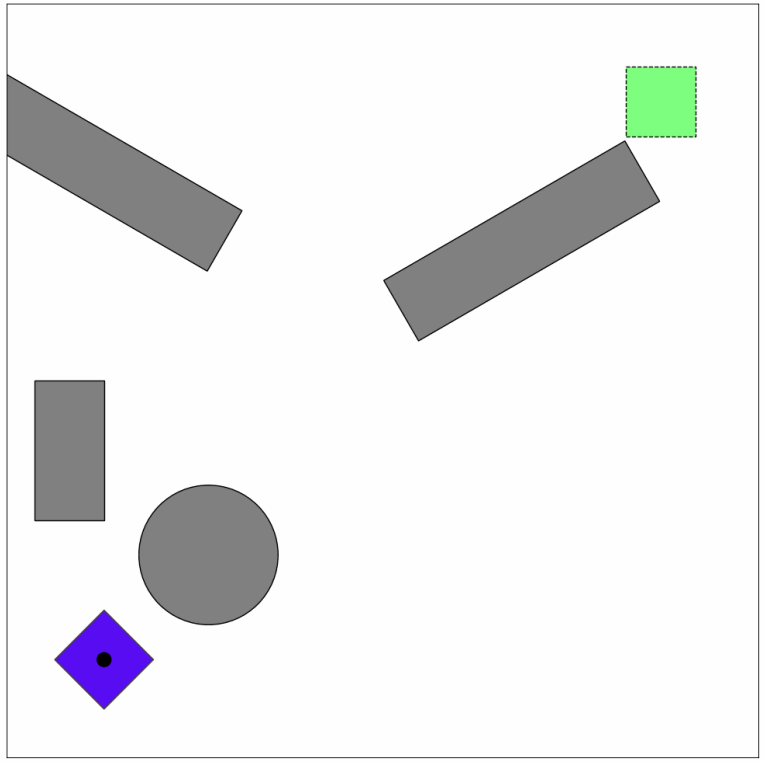


RRT(x_0, x_g, $\mathcal{X}$, f)

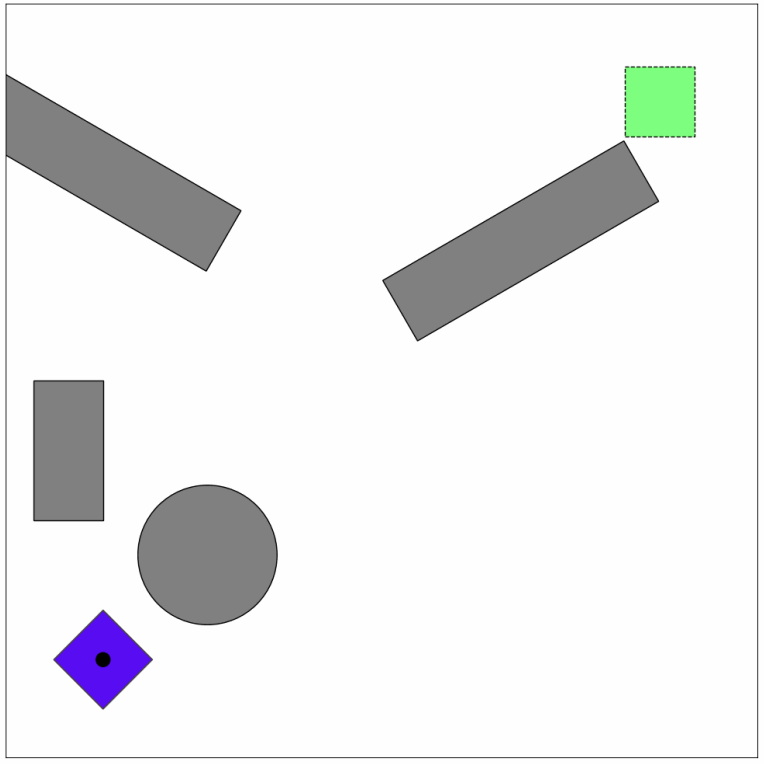
```
1  // Initialize tree at x_0
2  nodes = [Node(x_0)]
3  repeat:
4      if uniform() < goalSampleProb
5          // Try to go directly to the goal
6          x_target = x_g
7      else
8          // Sample a target configuration
9          x_target = sample(X)
10     // Extend the tree towards the target
11     node = getClosest(nodes, x_target)
12     x_node = node.conf
13     for x in extend(x_node, x_target)
14         if f(x):
15             nodes.add(Node(x))
16             if x = x_g:
17                 return finish(nodes)
18         else
19             break
```

Important Hyperparameter: Feasibility Check Distance

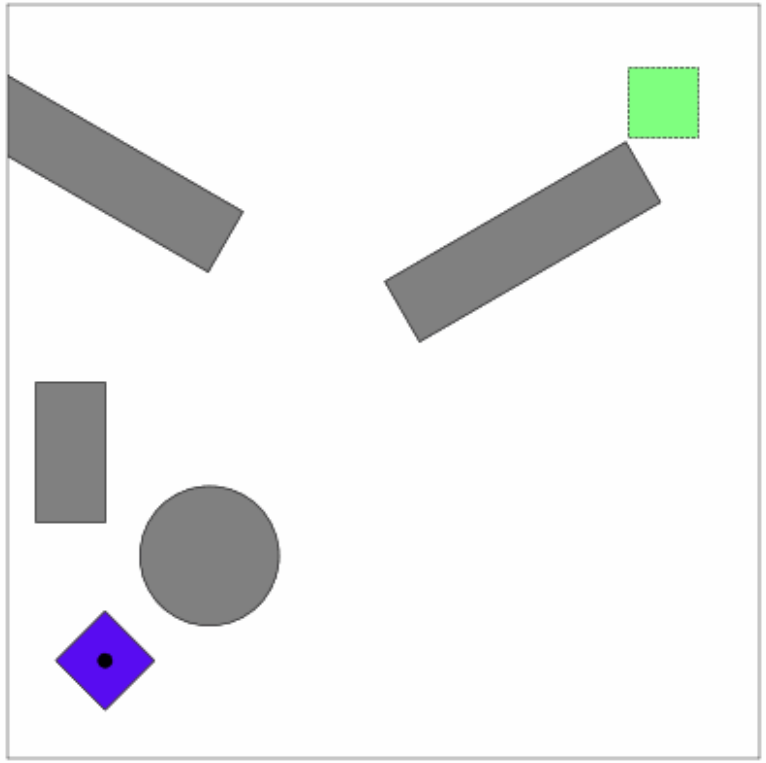
Check Distance = 5.0



Check Distance = 1.0

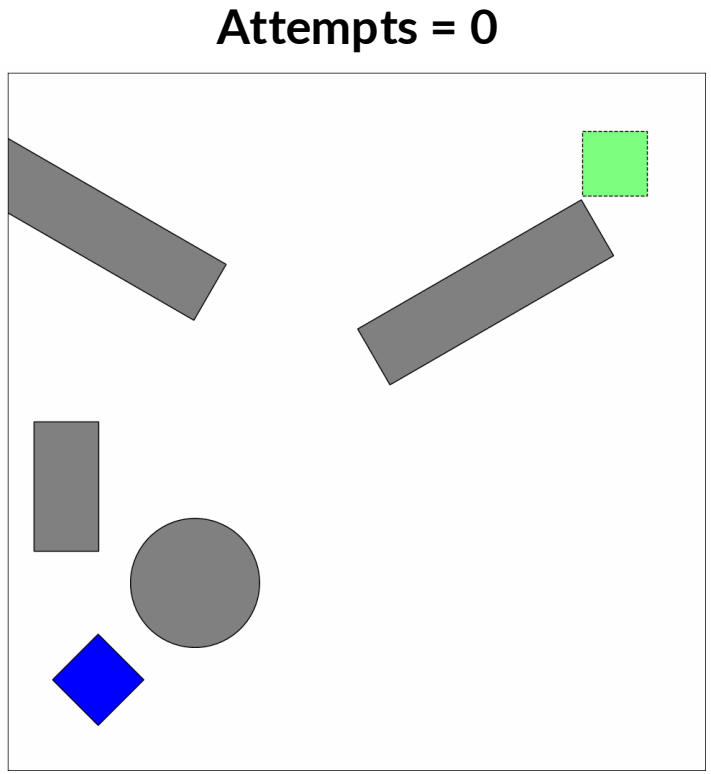


Check Distance = 0.2



Post-Processing with Shortcuts

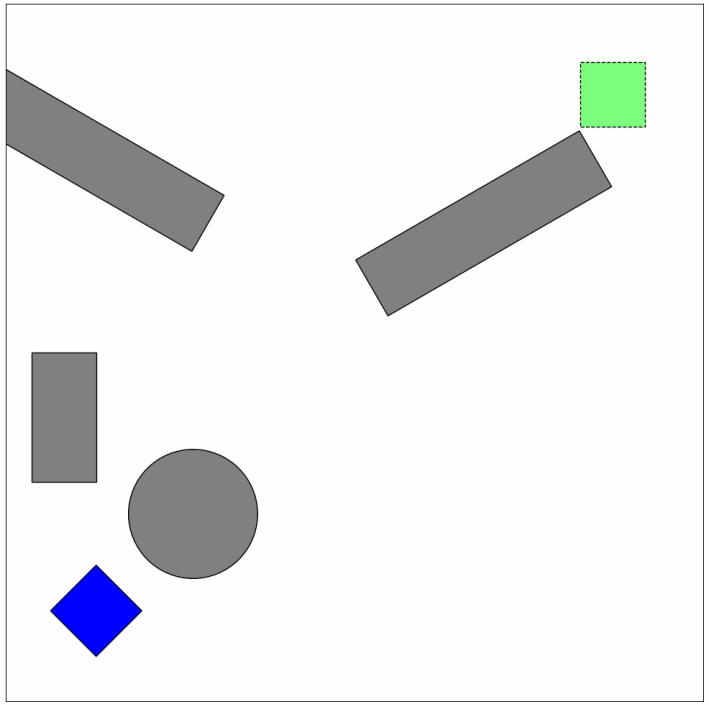
Repeatedly sample two points on the trajectory and check if a direct line between them is feasible (rewire if so)



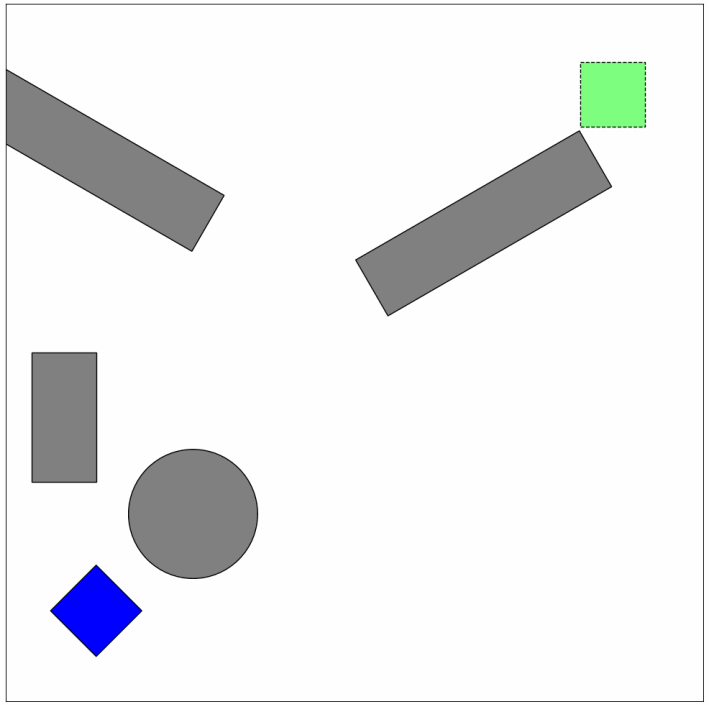
Post-Processing with Shortcuts

Repeatedly sample two points on the trajectory and check if a direct line between them is feasible (rewire if so)

Attempts = 0



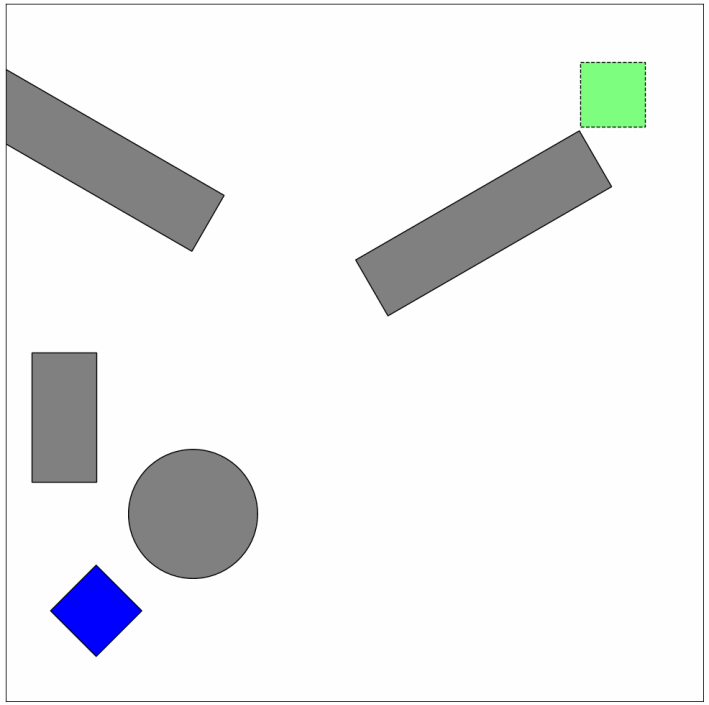
Attempts = 100



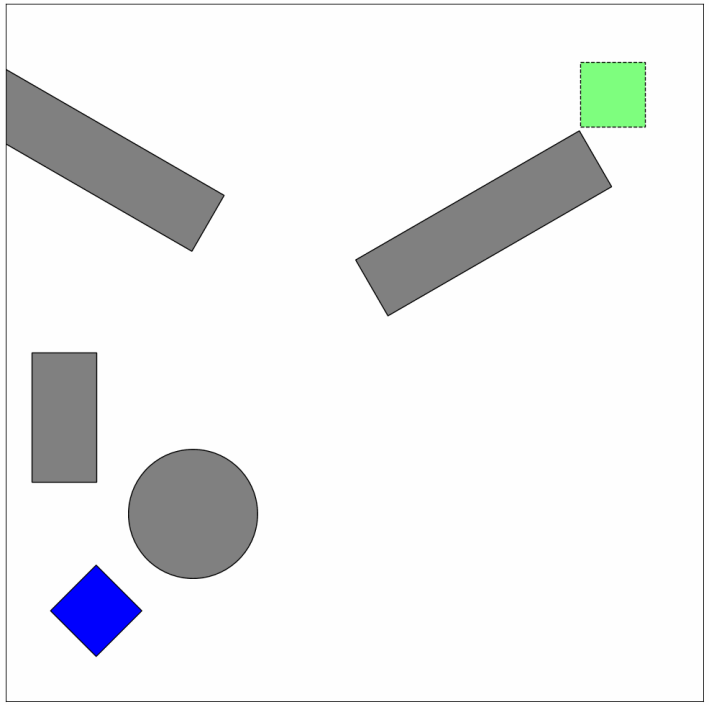
Post-Processing with Shortcuts

Repeatedly sample two points on the trajectory and check if a direct line between them is feasible (rewire if so)

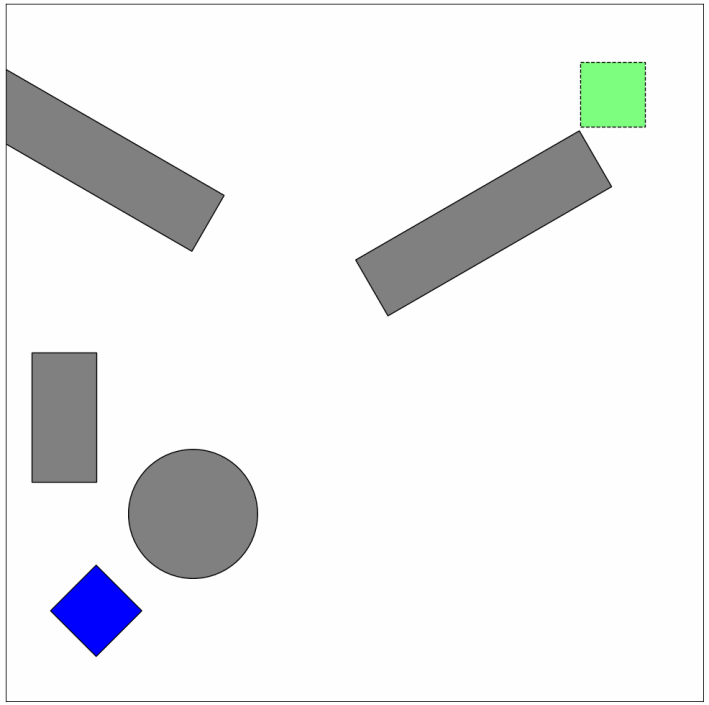
Attempts = 0



Attempts = 100



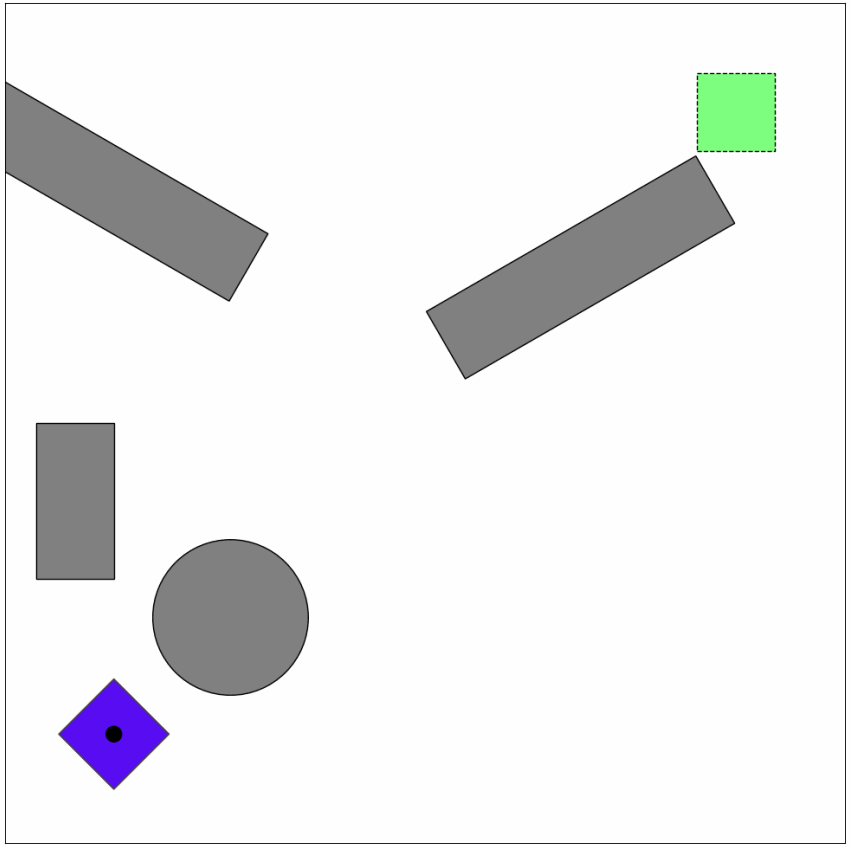
Attempts = 10000

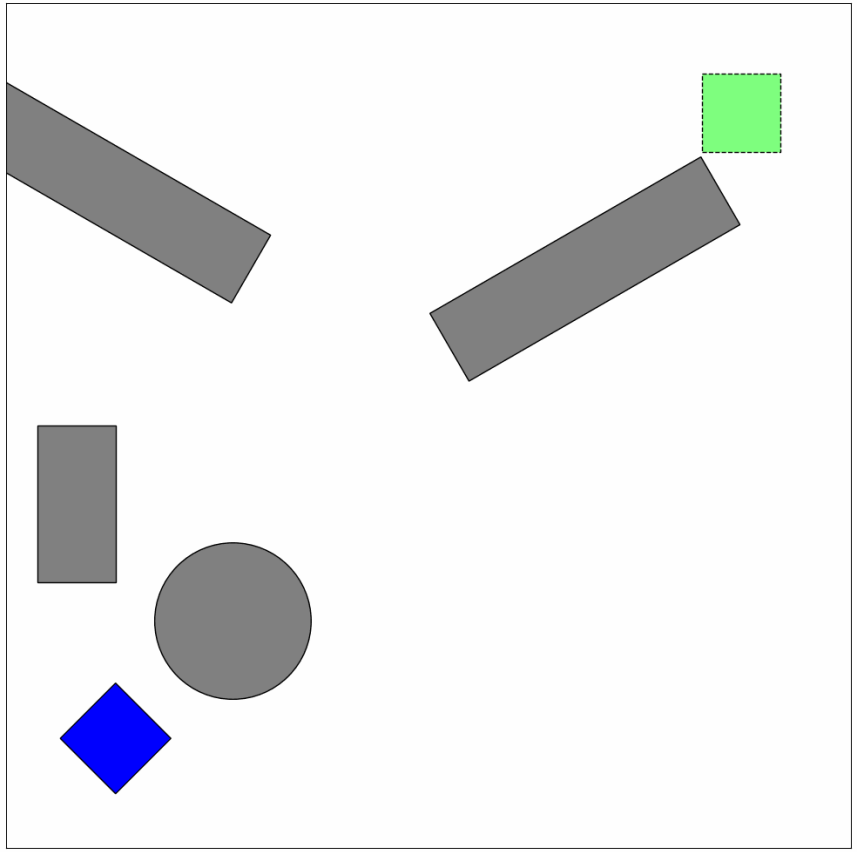


Bidirectional RRT

Does not work for underactuated systems

Grow two trees: one from start, the other from goal





Multi-Query Motion Planning

- RRT grows tree once, from initial to goal
- After the robot moves, need to start from scratch
- Can we build a representation once, up front, instead?
- Need a *graph* instead of a *tree*

Probabilistic Roadmaps (PRMs)

BuildPRM($\mathcal{X}$, f)

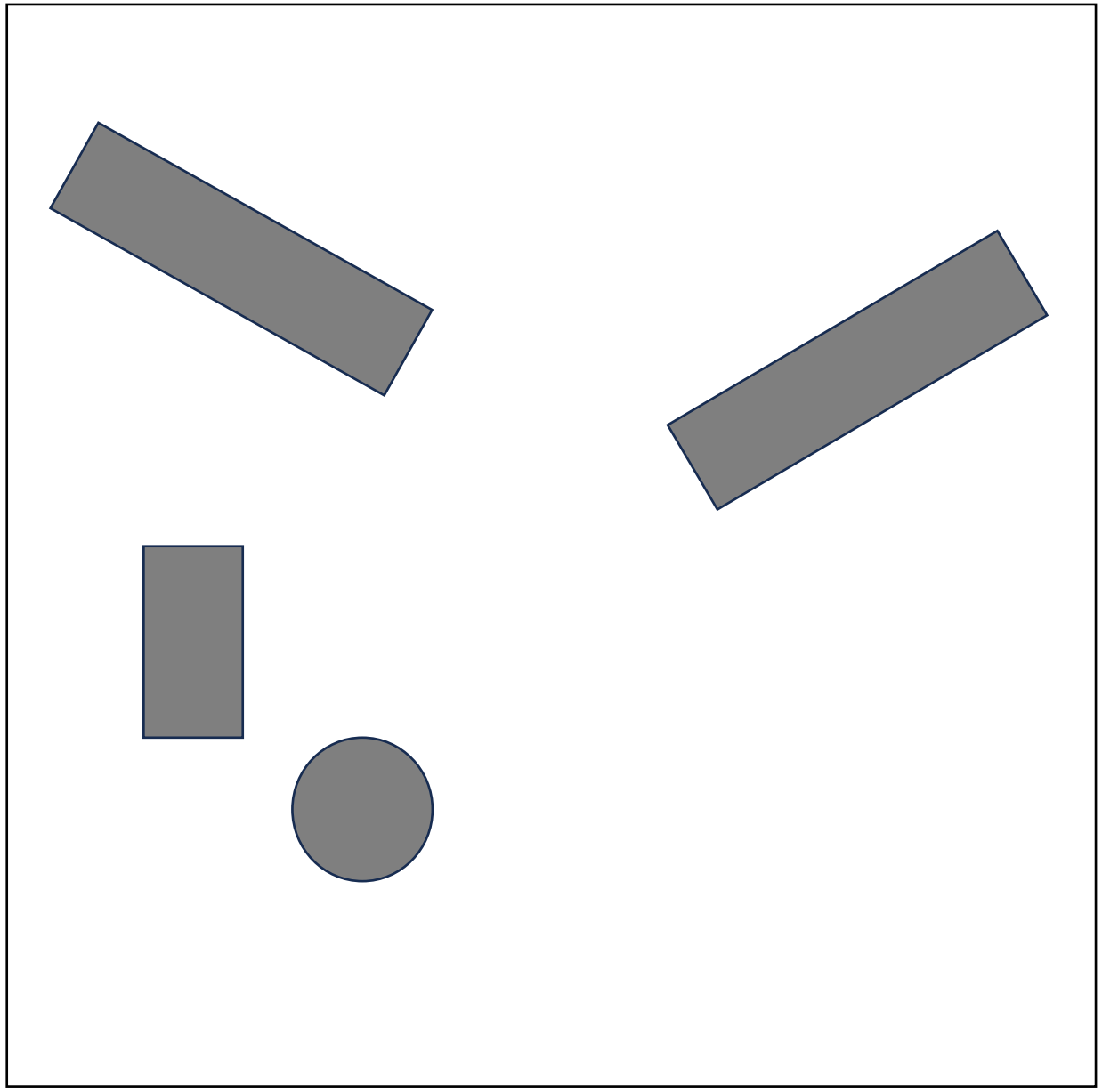
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

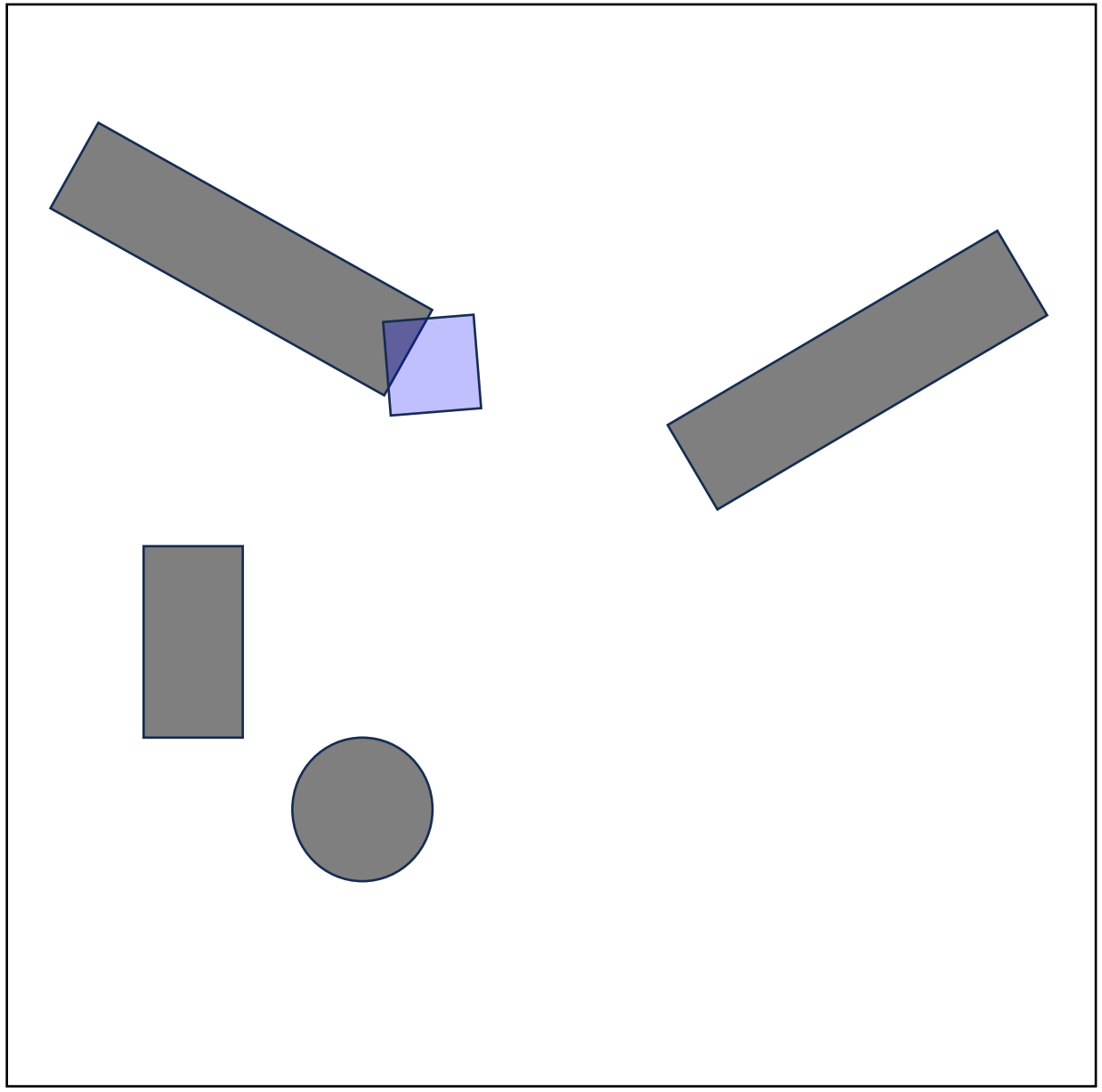
```
1  graph = UndirectedGraph()
2  repeat:
3      // Sample from configuration space
4      x = sample(𝒳)
5      // Skip if not feasible
6      if not f(x): continue
7      // Update the graph
8      UpdatePRM(x, graph, f)
9  return graph
```

UpdatePRM(x, graph, f)

```
1  newNode = addNode(graph, x)
2  for node ∈ getNeighbors(x)
3      if pathFeasible(node.conf, x, f):
4          addEdge(graph, node, newNode)
5  return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
1  initNode = UpdatePRM(x_0, graph, f)
2  goalNode = UpdatePRM(x_g, graph, f)
3  nodePath = graphSearch(initNode, goalNode)
4  return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

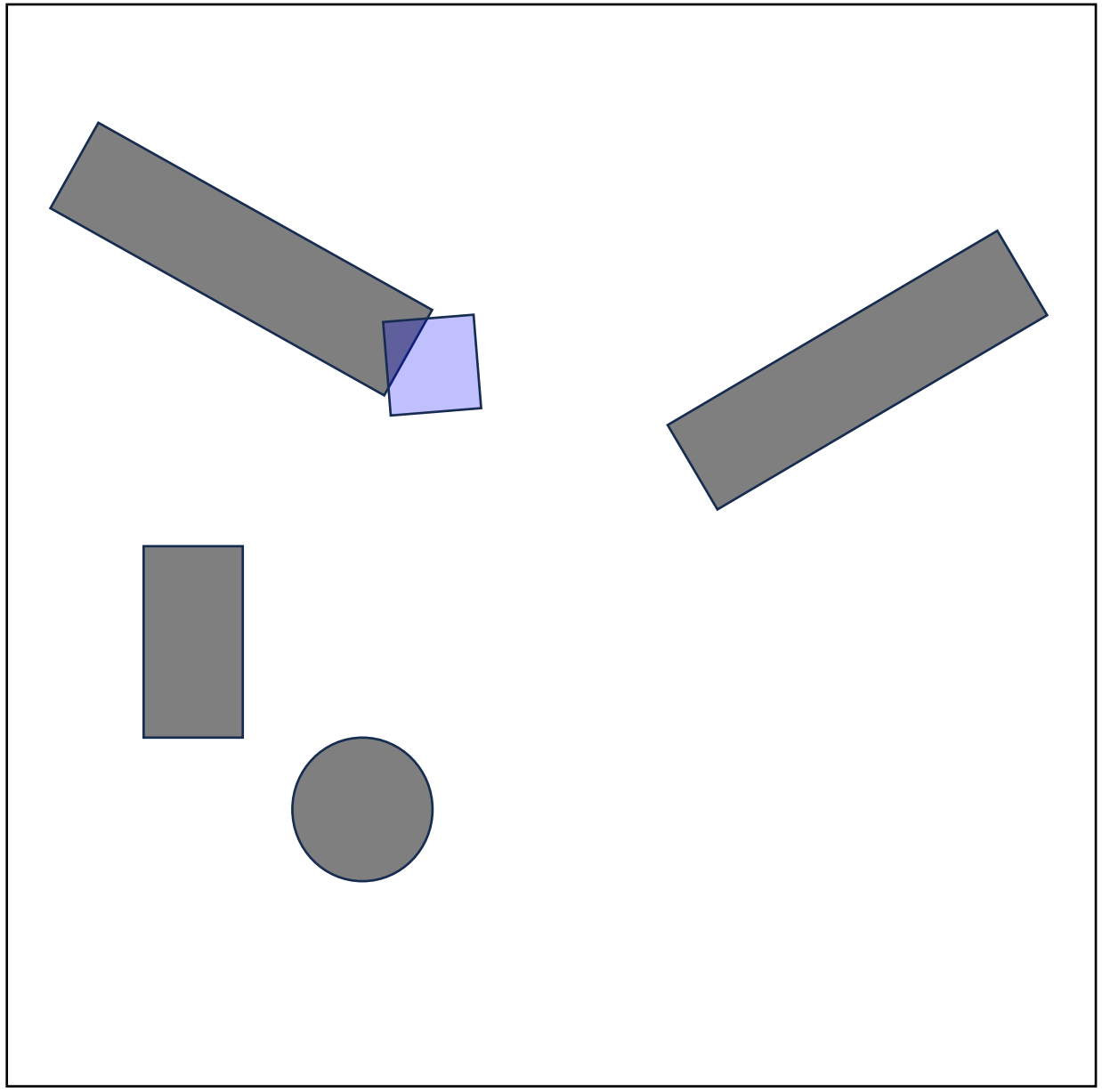
```
1  graph = UndirectedGraph()
2  repeat:
3      // Sample from configuration space
4      x = sample(𝒳)
5      // Skip if not feasible
6      if not f(x): continue
7      // Update the graph
8      UpdatePRM(x, graph, f)
9  return graph
```

UpdatePRM(x, graph, f)

```
1  newNode = addNode(graph, x)
2  for node ∈ getNeighbors(x)
3      if pathFeasible(node.conf, x, f):
4          addEdge(graph, node, newNode)
5  return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
1  initNode = UpdatePRM(x_0, graph, f)
2  goalNode = UpdatePRM(x_g, graph, f)
3  nodePath = graphSearch(initNode, goalNode)
4  return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

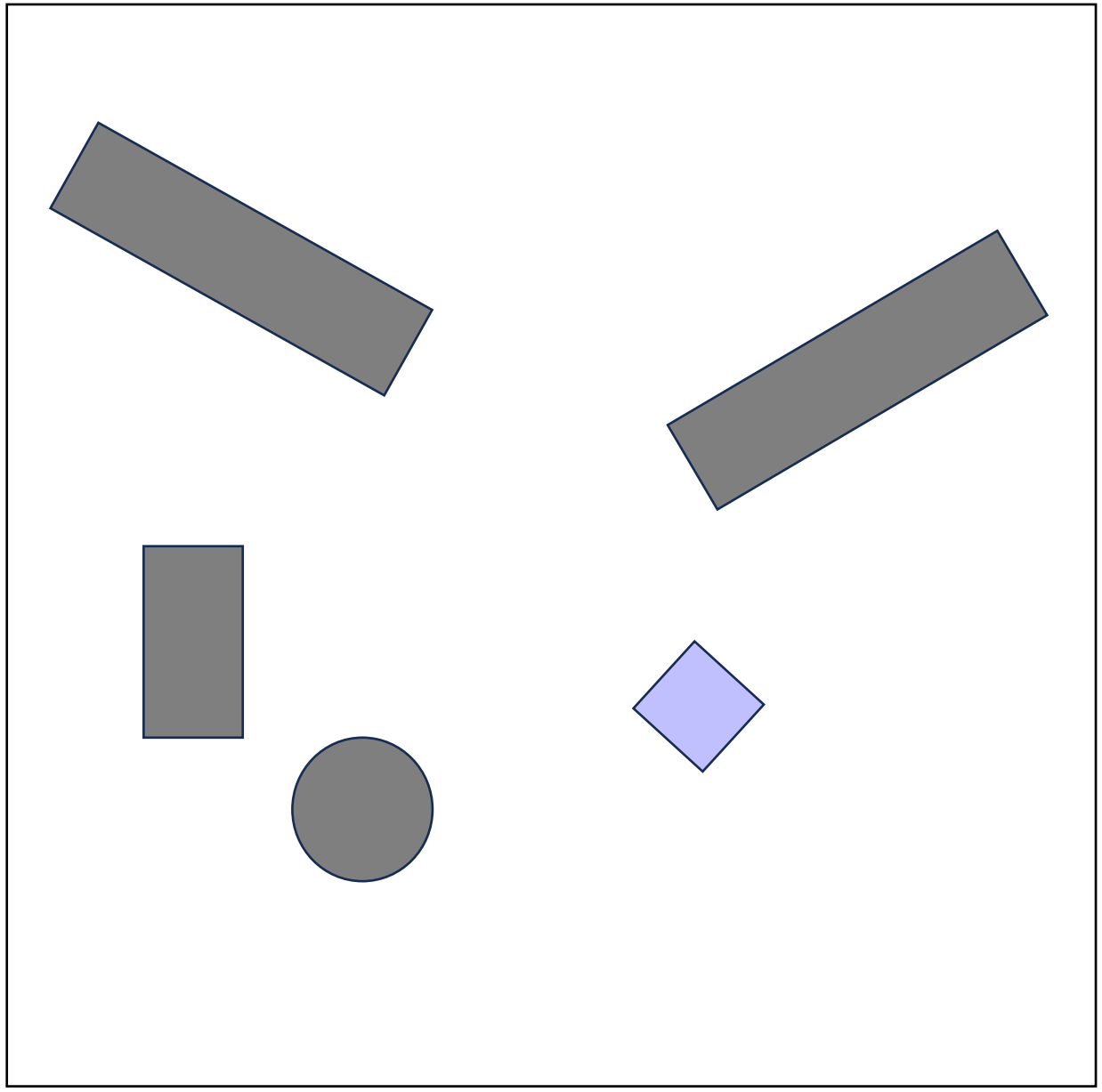
```
1  graph = UndirectedGraph()
2  repeat:
3      // Sample from configuration space
4      x = sample(𝒳)
5      // Skip if not feasible
6      if not f(x): continue
7      // Update the graph
8      UpdatePRM(x, graph, f)
9  return graph
```

UpdatePRM(x, graph, f)

```
1  newNode = addNode(graph, x)
2  for node ∈ getNeighbors(x)
3      if pathFeasible(node.conf, x, f):
4          addEdge(graph, node, newNode)
5  return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
1  initNode = UpdatePRM(x_0, graph, f)
2  goalNode = UpdatePRM(x_g, graph, f)
3  nodePath = graphSearch(initNode, goalNode)
4  return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

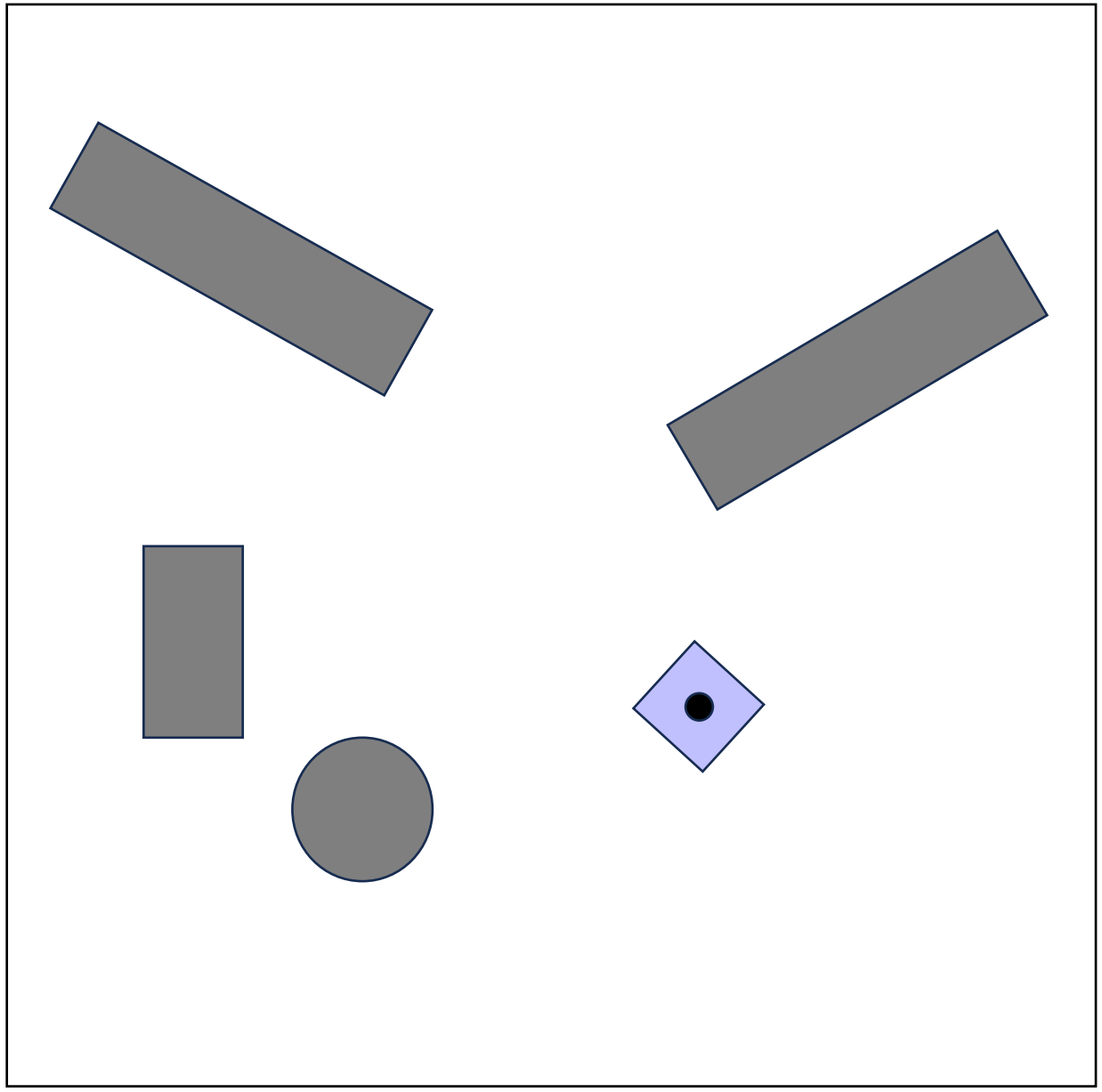
```
1  graph = UndirectedGraph()
2  repeat:
3      // Sample from configuration space
4      x = sample(𝒳)
5      // Skip if not feasible
6      if not f(x): continue
7      // Update the graph
8      UpdatePRM(x, graph, f)
9  return graph
```

UpdatePRM(x, graph, f)

```
1  newNode = addNode(graph, x)
2  for node ∈ getNeighbors(x)
3      if pathFeasible(node.conf, x, f):
4          addEdge(graph, node, newNode)
5  return newNode
```

QueryPRM(x_0, x_g, graph, f)

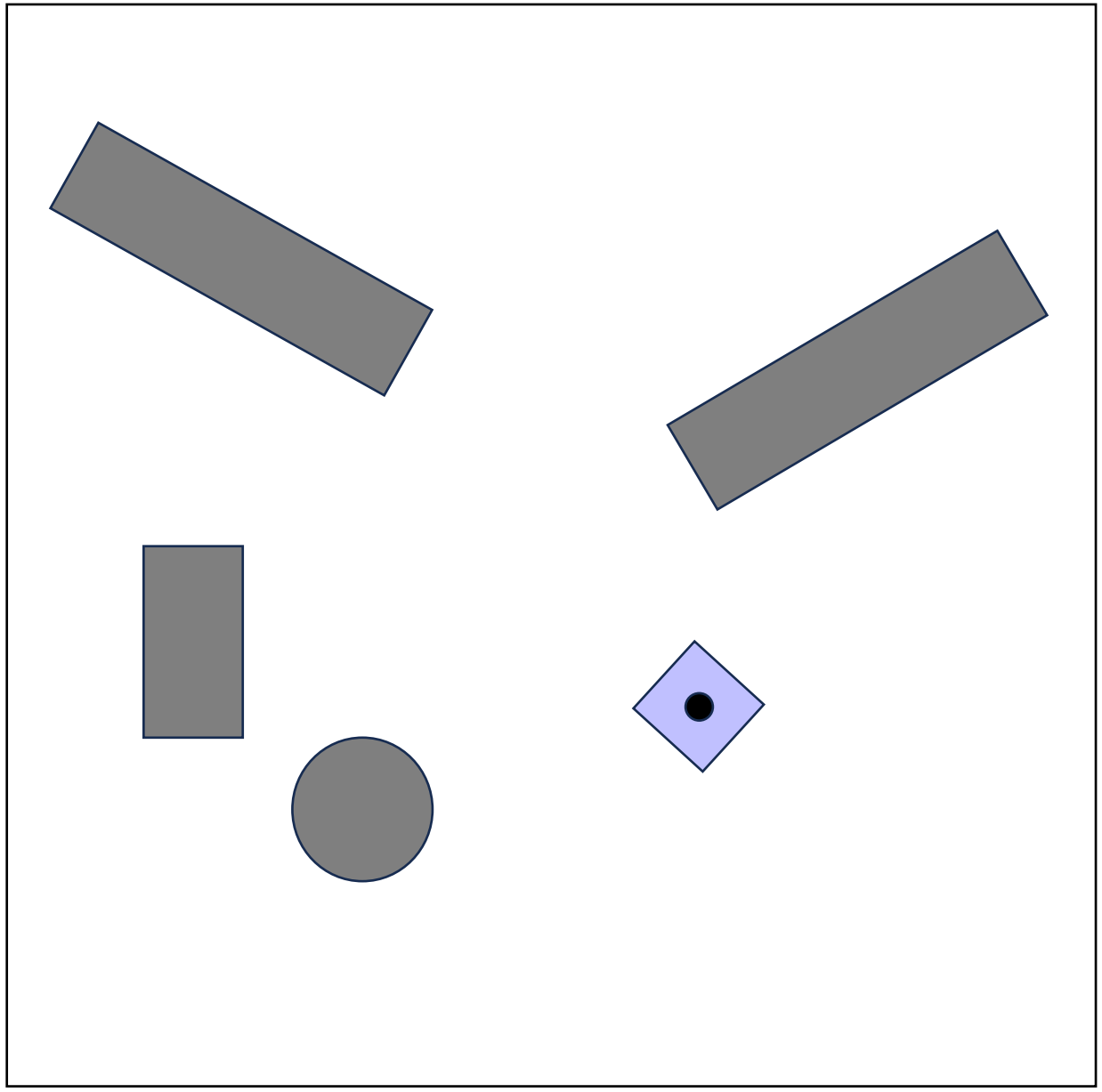
```
1  initNode = UpdatePRM(x_0, graph, f)
2  goalNode = UpdatePRM(x_g, graph, f)
3  nodePath = graphSearch(initNode, goalNode)
4  return finish(nodePath)
```



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

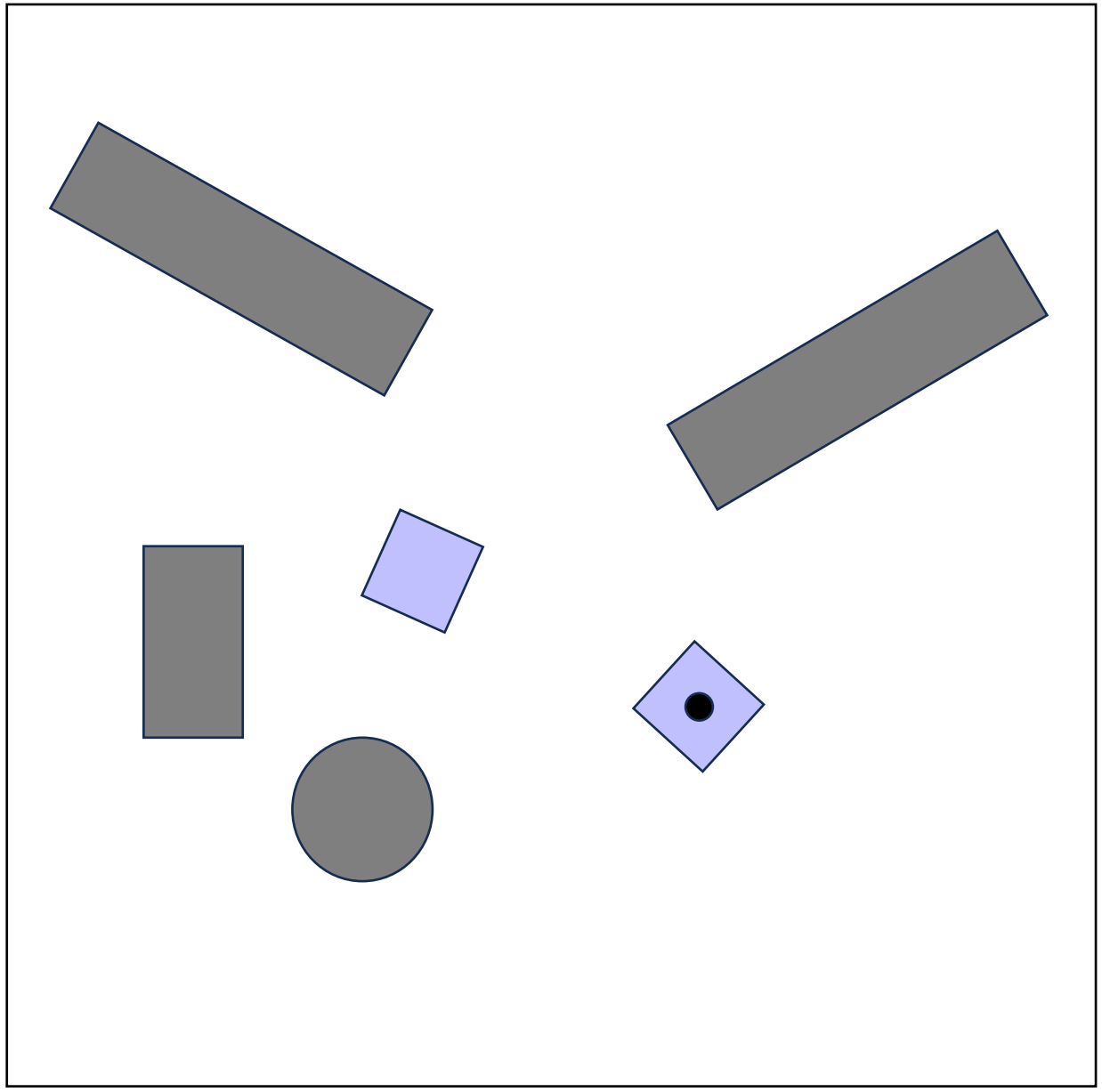
```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

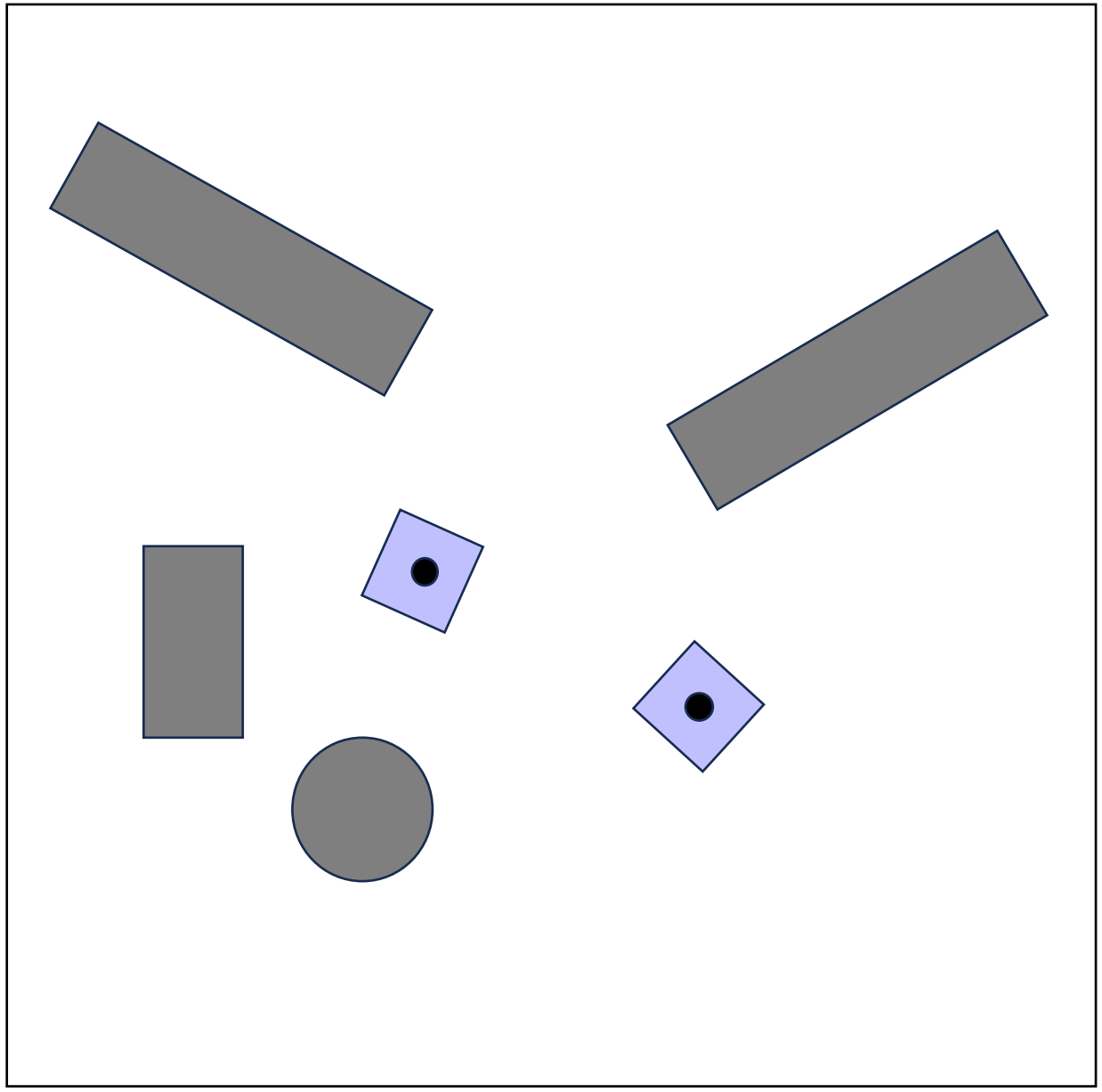
```
1  graph = UndirectedGraph()
2  repeat:
3      // Sample from configuration space
4      x = sample(𝒳)
5      // Skip if not feasible
6      if not f(x): continue
7      // Update the graph
8      UpdatePRM(x, graph, f)
9  return graph
```

UpdatePRM(x, graph, f)

```
1  newNode = addNode(graph, x)
2  for node ∈ getNeighbors(x)
3      if pathFeasible(node.conf, x, f):
4          addEdge(graph, node, newNode)
5  return newNode
```

QueryPRM(x_0, x_g, graph, f)

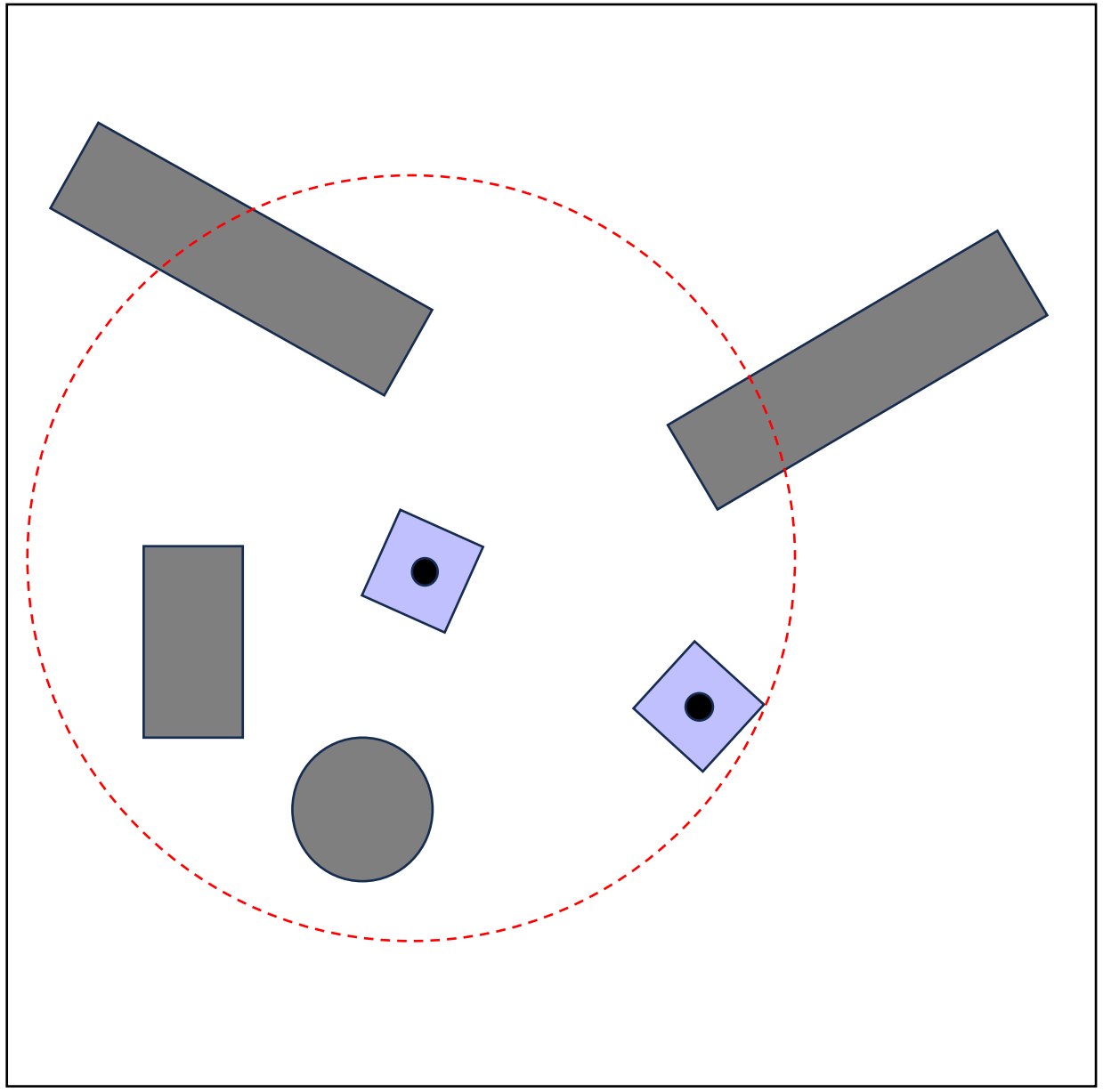
```
1  initNode = UpdatePRM(x_0, graph, f)
2  goalNode = UpdatePRM(x_g, graph, f)
3  nodePath = graphSearch(initNode, goalNode)
4  return finish(nodePath)
```



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

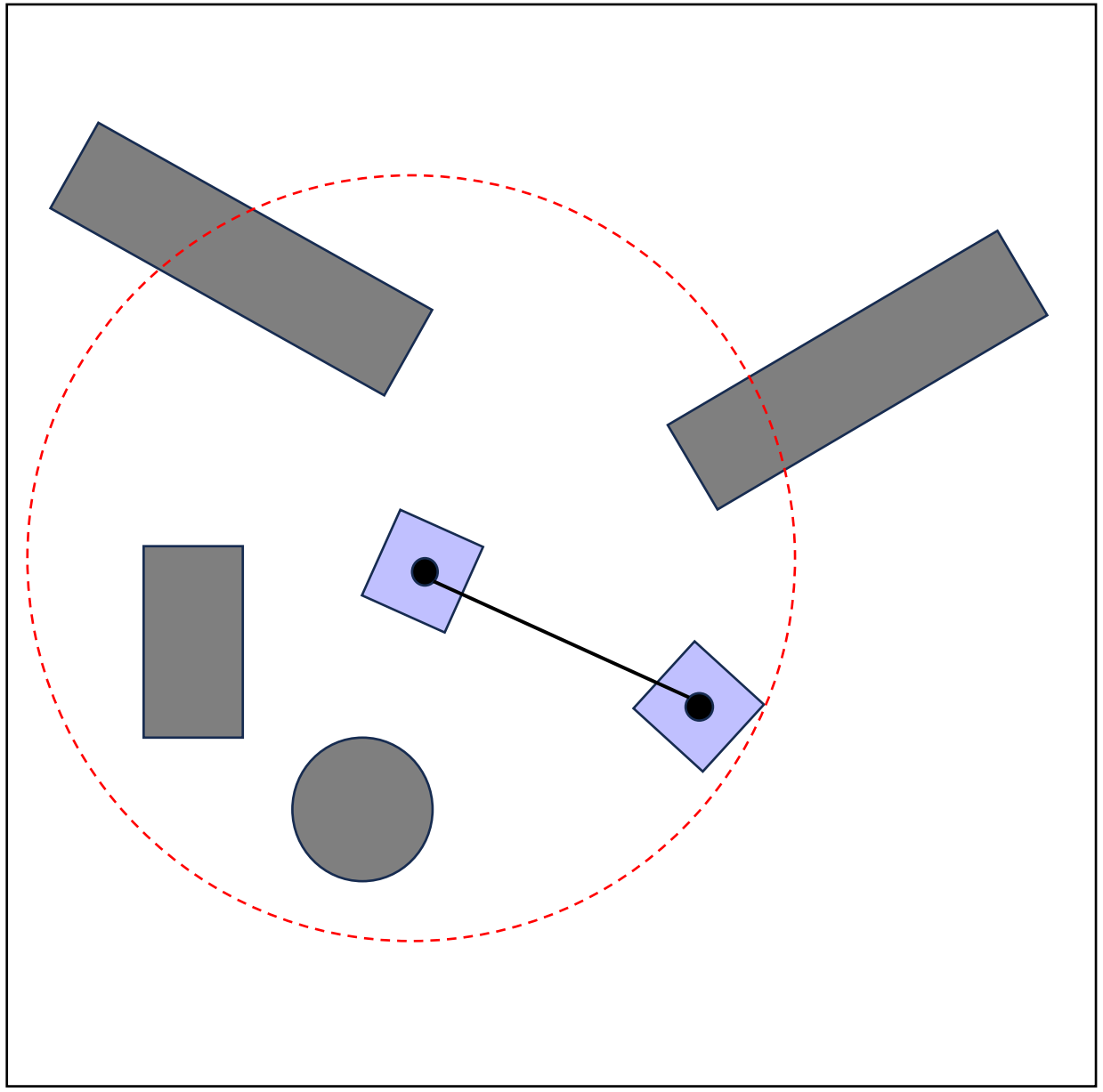
```
1  graph = UndirectedGraph()
2  repeat:
3      // Sample from configuration space
4      x = sample(𝒳)
5      // Skip if not feasible
6      if not f(x): continue
7      // Update the graph
8      UpdatePRM(x, graph, f)
9  return graph
```

UpdatePRM(x, graph, f)

```
1  newNode = addNode(graph, x)
2  for node ∈ getNeighbors(x)
3      if pathFeasible(node.conf, x, f):
4          addEdge(graph, node, newNode)
5  return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
1  initNode = UpdatePRM(x_0, graph, f)
2  goalNode = UpdatePRM(x_g, graph, f)
3  nodePath = graphSearch(initNode, goalNode)
4  return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

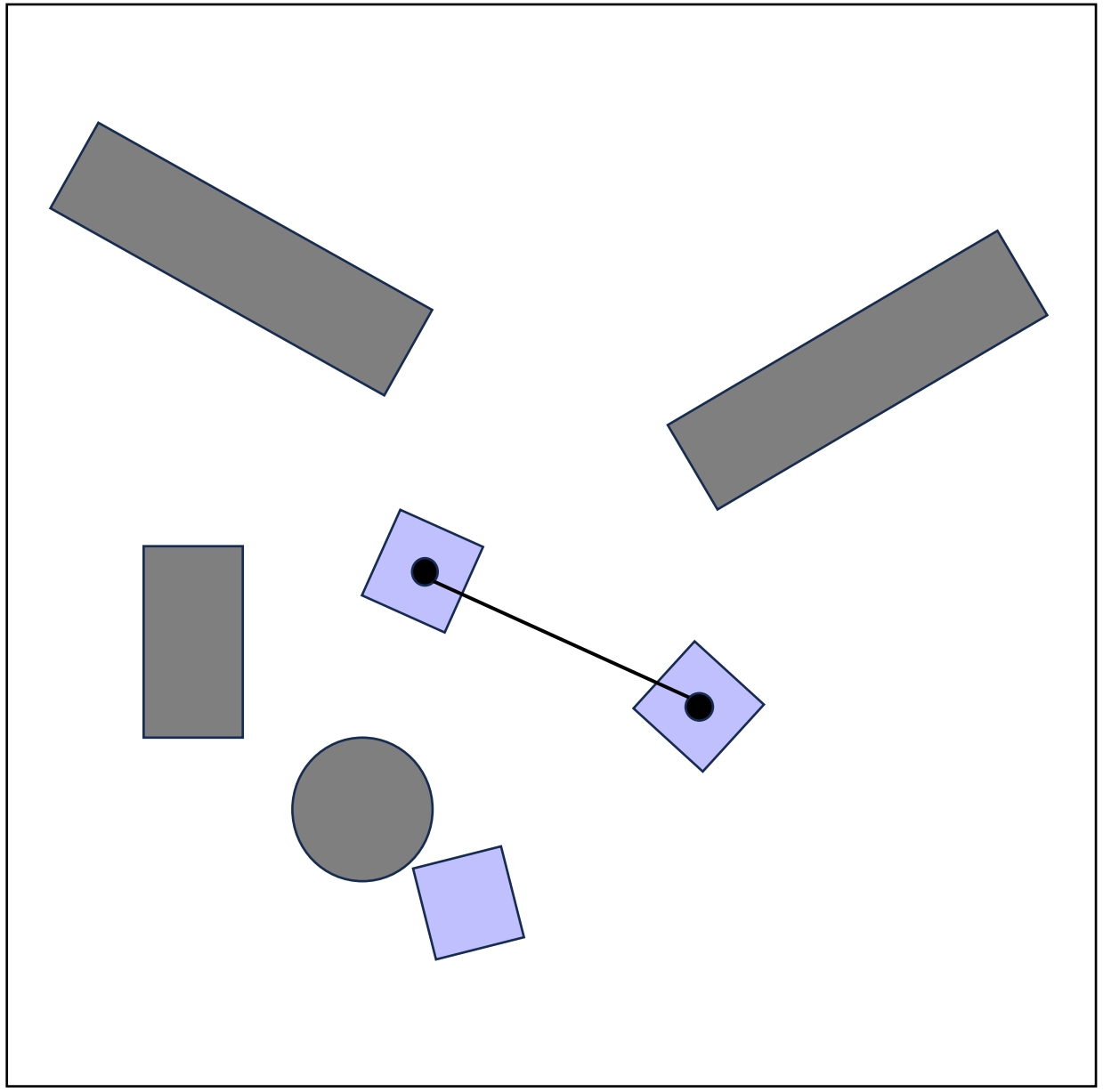
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

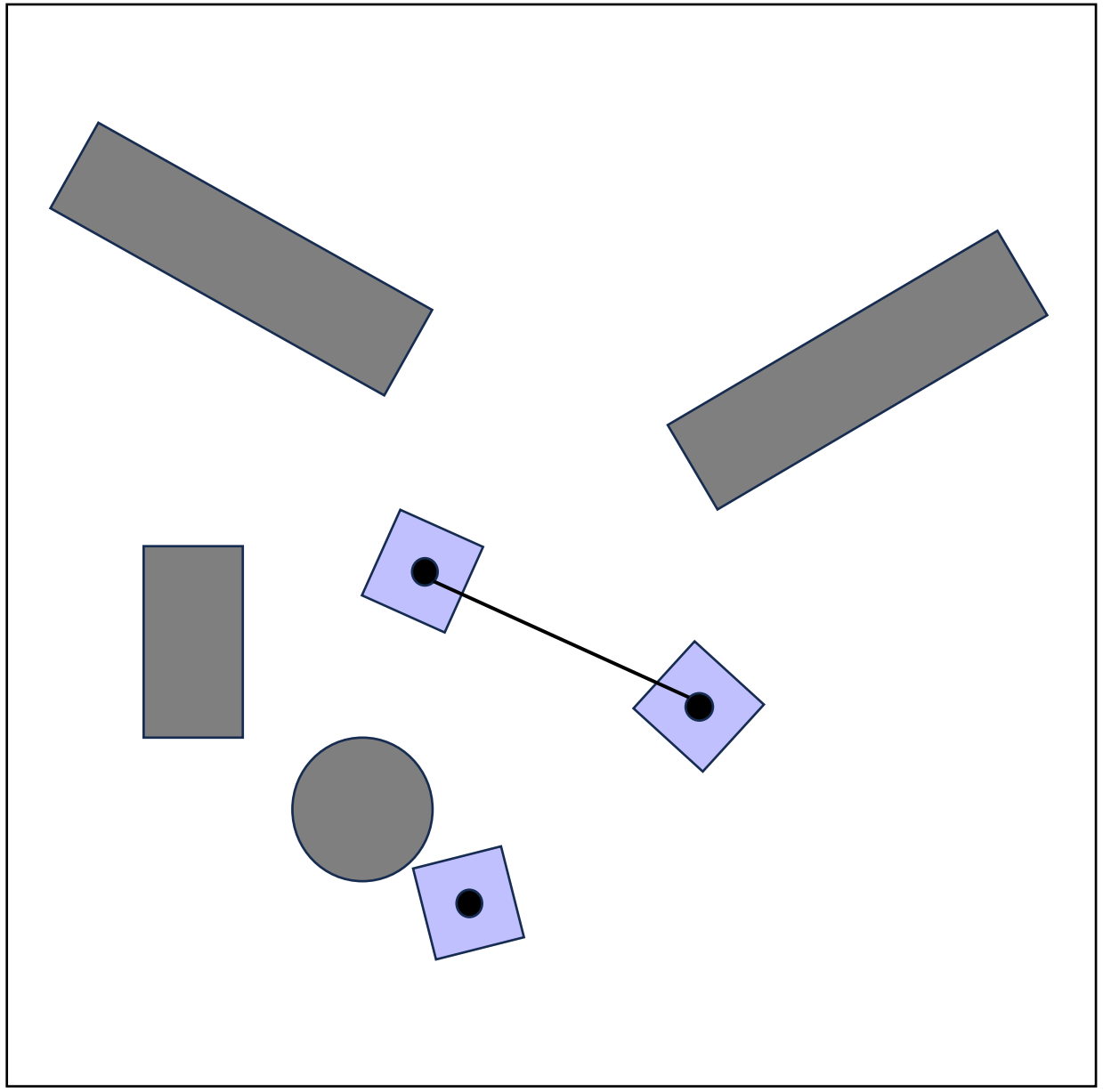
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

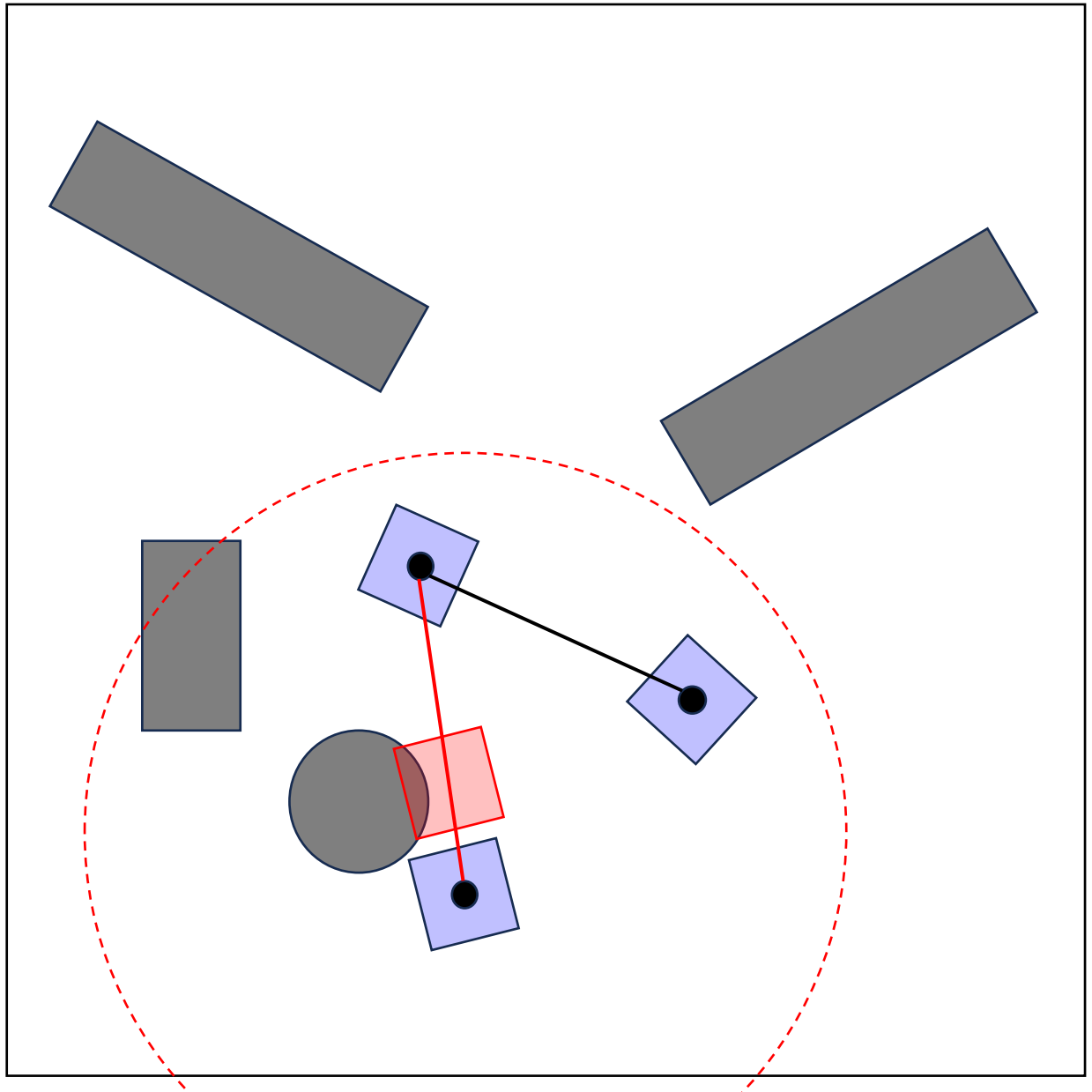
```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



```
BuildPRM(𝒳, f)
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

```
UpdatePRM(x, graph, f)
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

```
QueryPRM(x_0, x_g, graph, f)
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

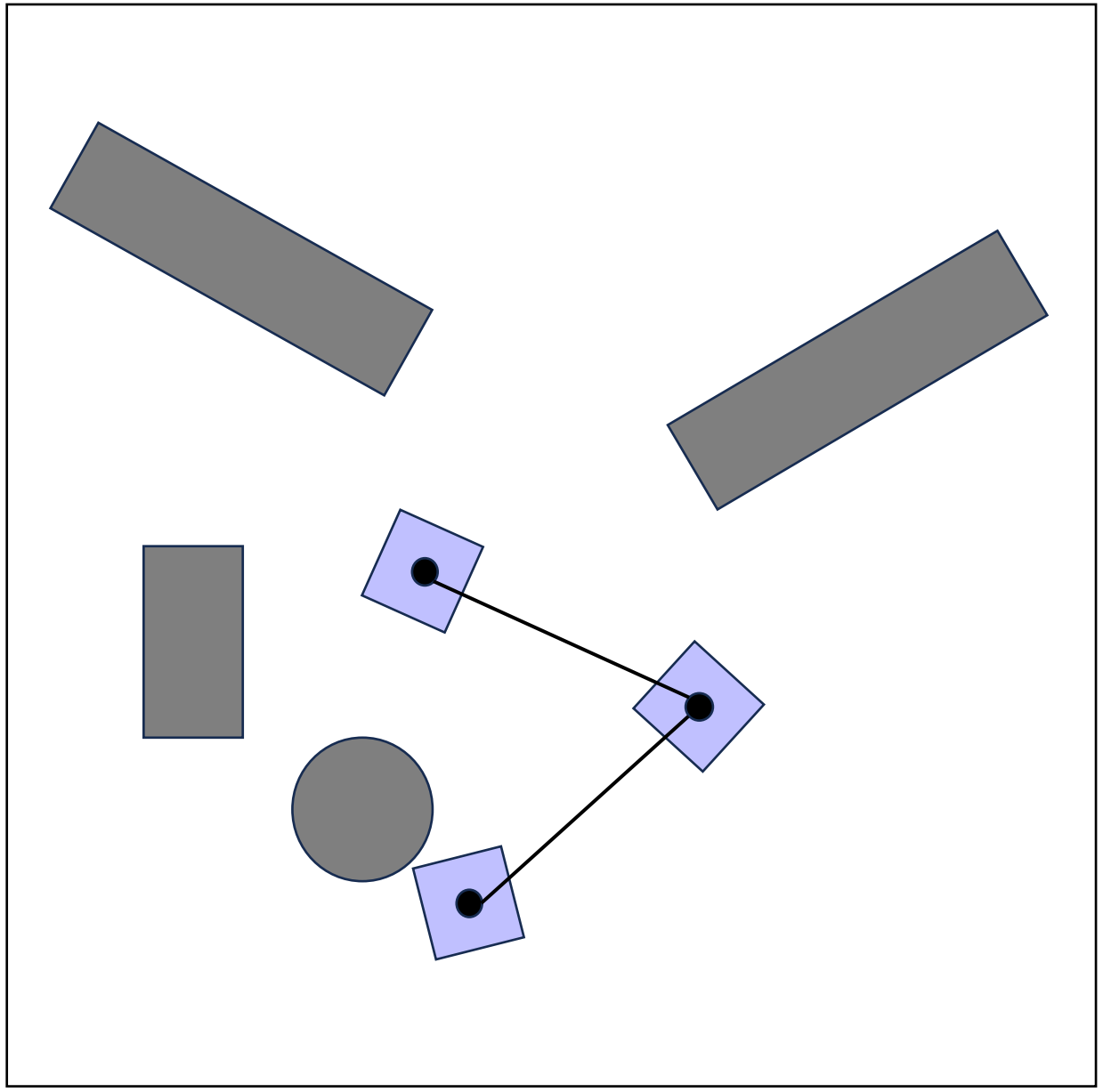
```
1  graph = UndirectedGraph()
2  repeat:
3      // Sample from configuration space
4      x = sample(𝒳)
5      // Skip if not feasible
6      if not f(x): continue
7      // Update the graph
8      UpdatePRM(x, graph, f)
9  return graph
```

UpdatePRM(x, graph, f)

```
1  newNode = addNode(graph, x)
2  for node ∈ getNeighbors(x)
3      if pathFeasible(node.conf, x, f):
4          addEdge(graph, node, newNode)
5  return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
1  initNode = UpdatePRM(x_0, graph, f)
2  goalNode = UpdatePRM(x_g, graph, f)
3  nodePath = graphSearch(initNode, goalNode)
4  return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

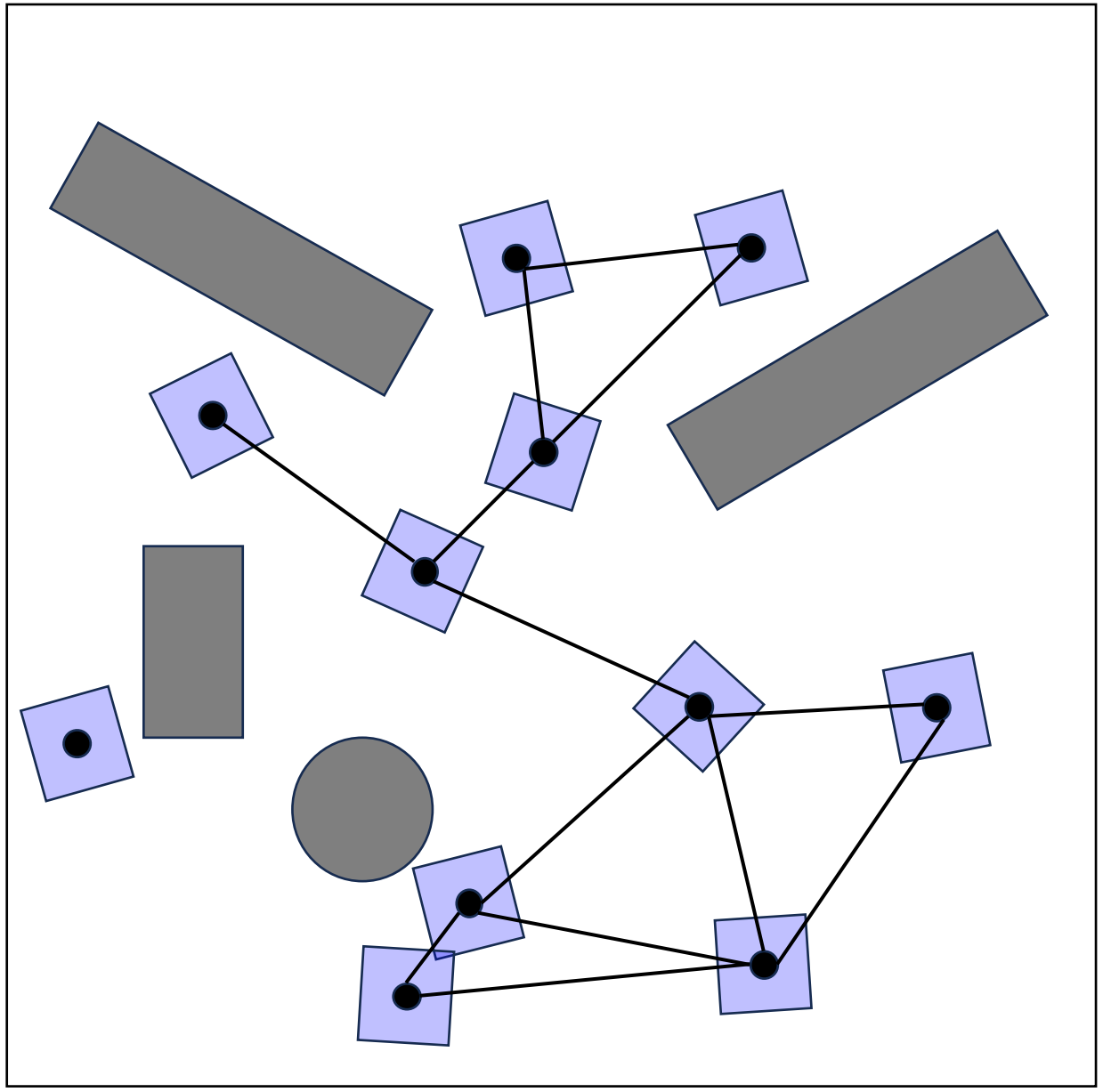
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

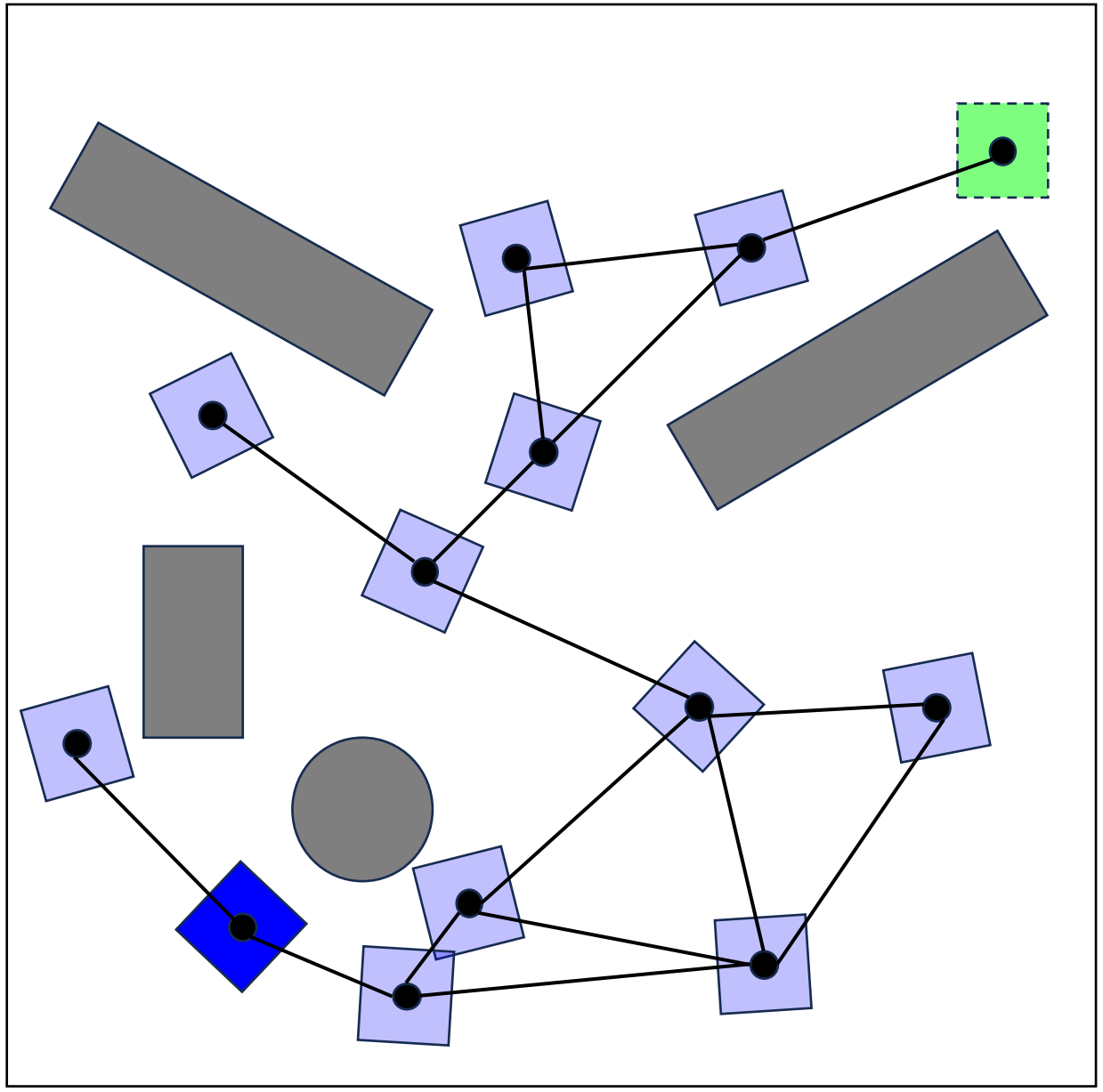
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

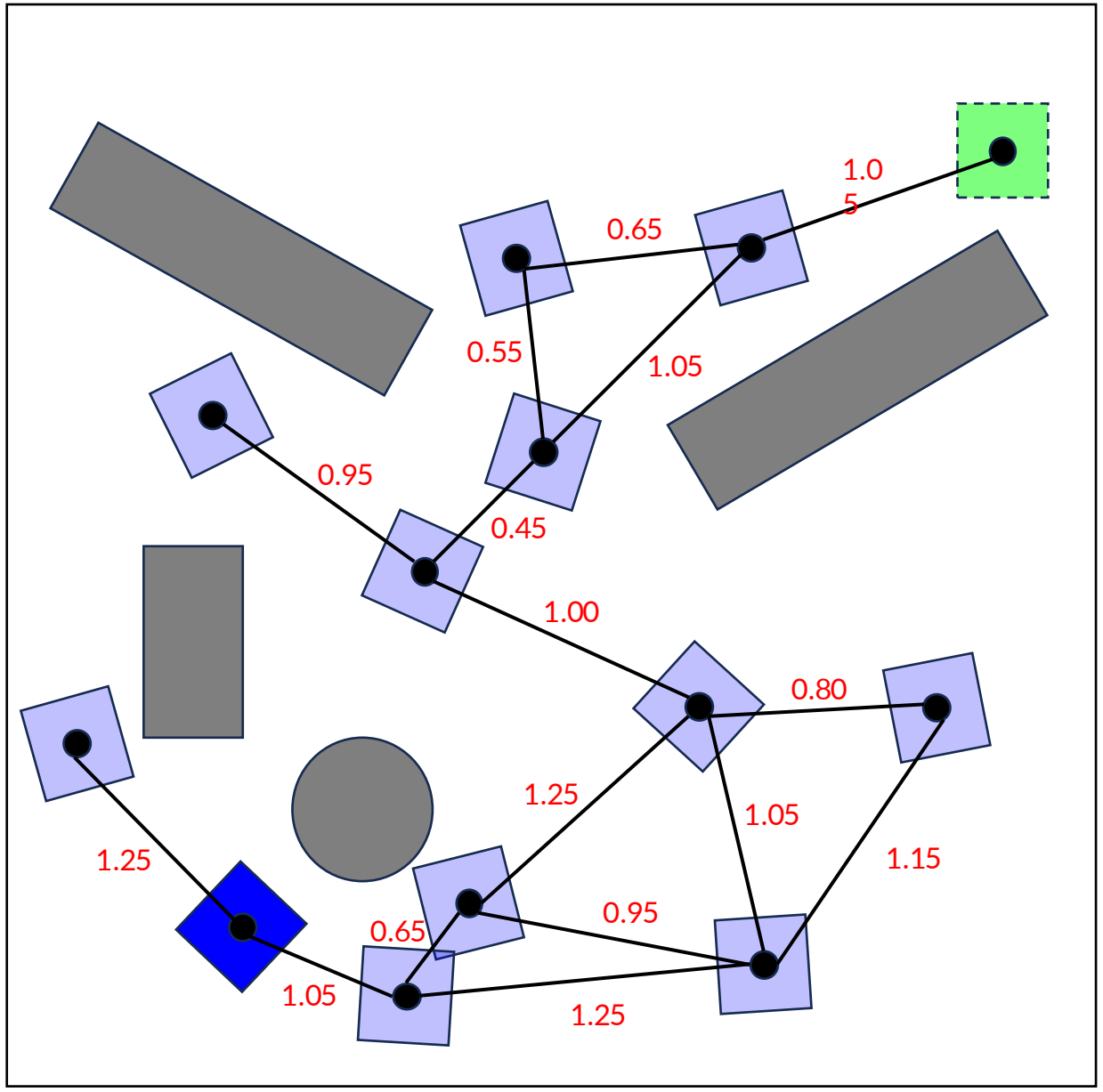
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

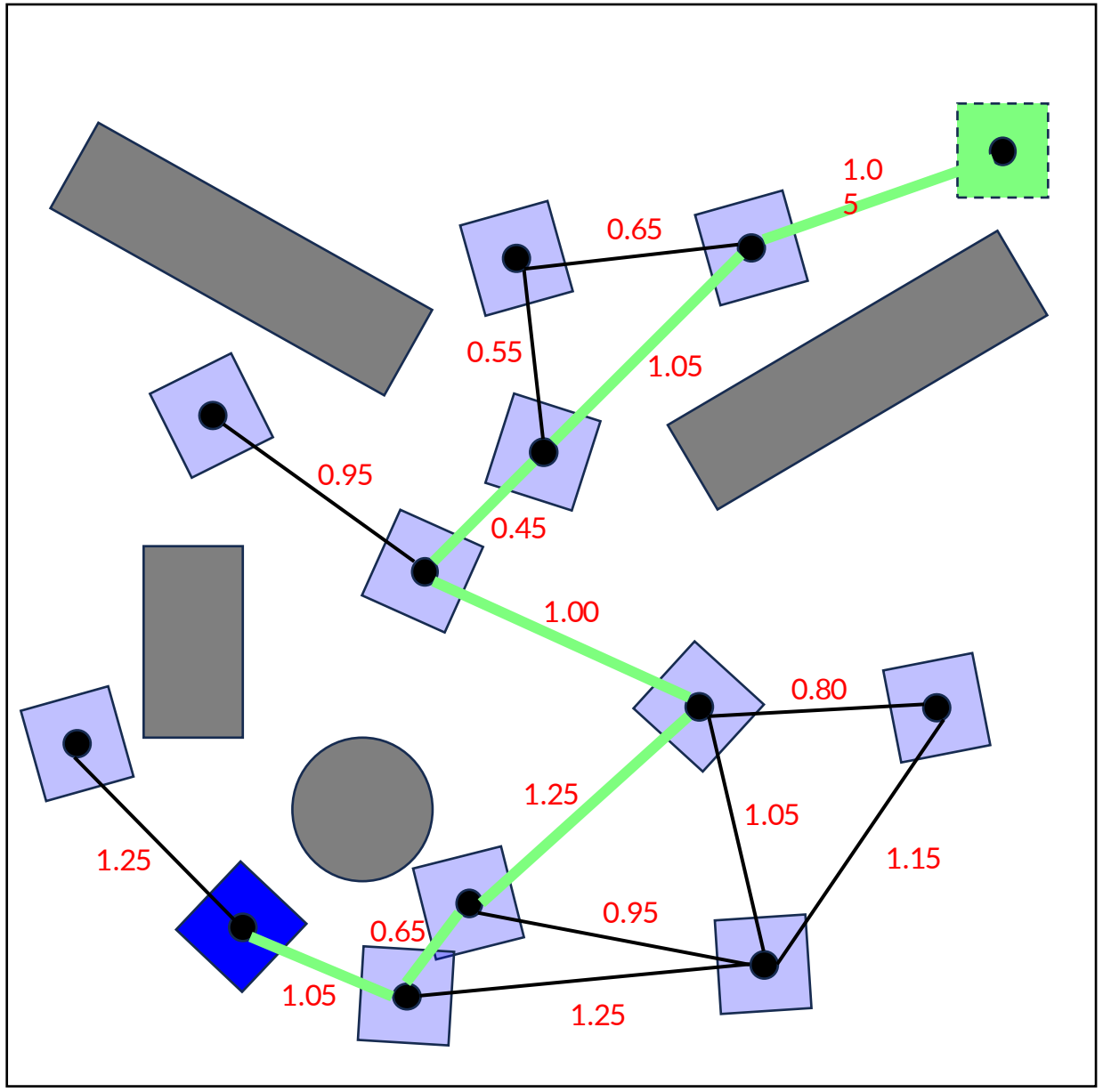
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

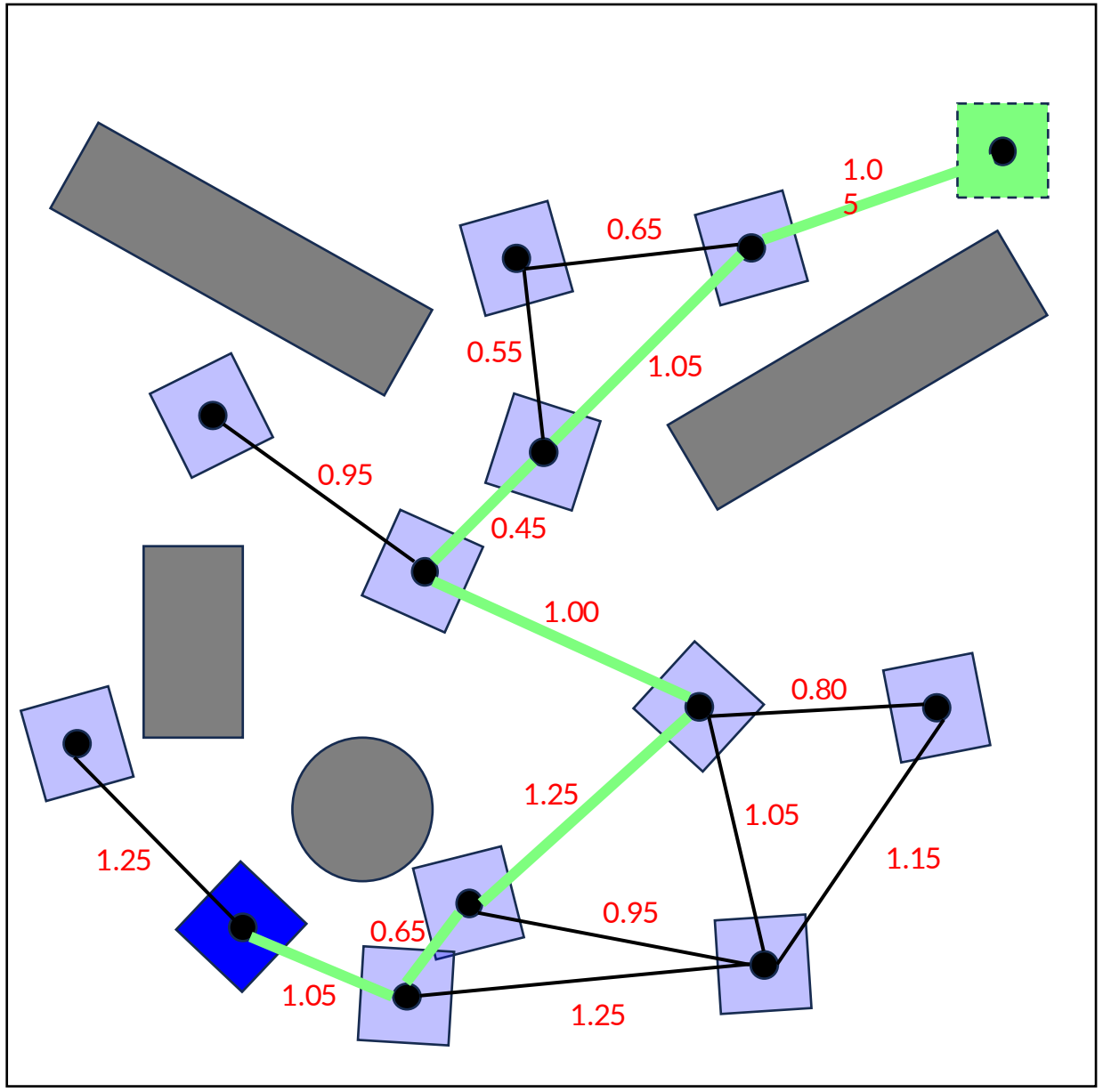
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

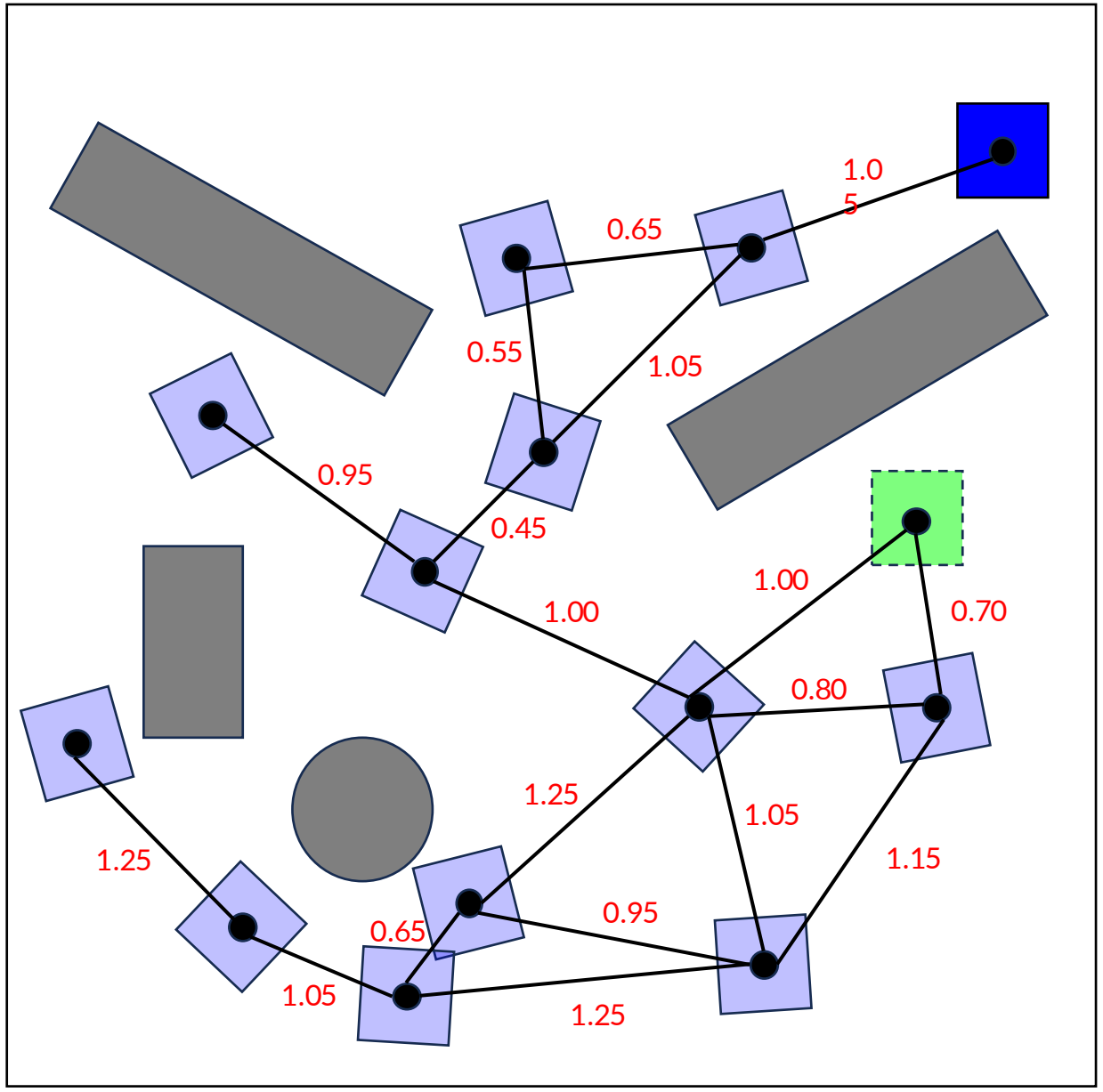
```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, node, newNode)
return newNode
```

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

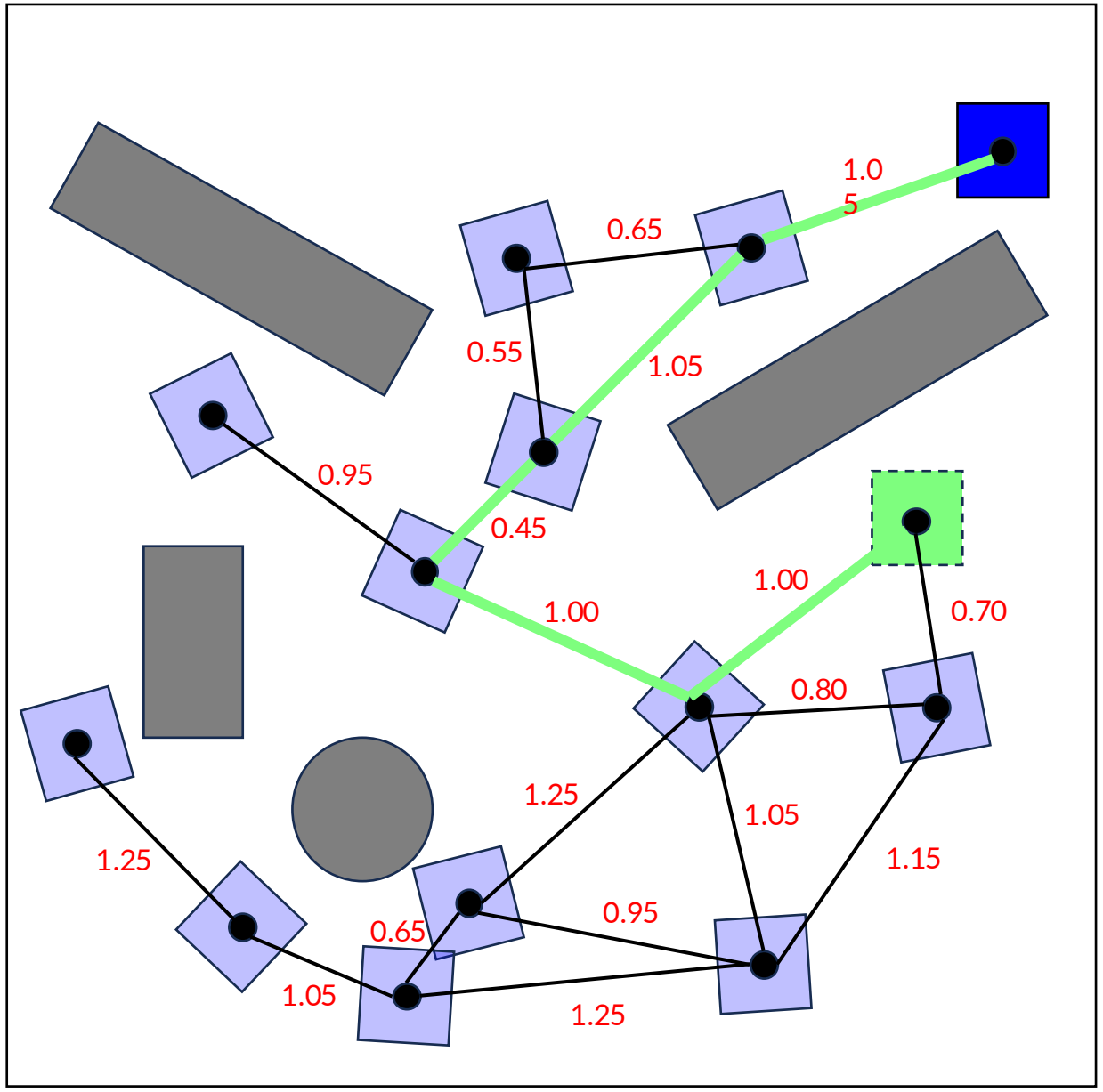
UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, no
return newNode
```

Reuse graph for new query

QueryPRM(x_0, x_g, graph, f)

```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```



BuildPRM($\mathcal{X}$, f)

```
graph = UndirectedGraph()
repeat:
    // Sample from configuration space
    x = sample(𝒳)
    // Skip if not feasible
    if not f(x): continue
    // Update the graph
    UpdatePRM(x, graph, f)
return graph
```

UpdatePRM(x, graph, f)

```
newNode = addNode(graph, x)
for node ∈ getNeighbors(x)
    if pathFeasible(node.conf, x, f):
        addEdge(graph, no
return newNode
```

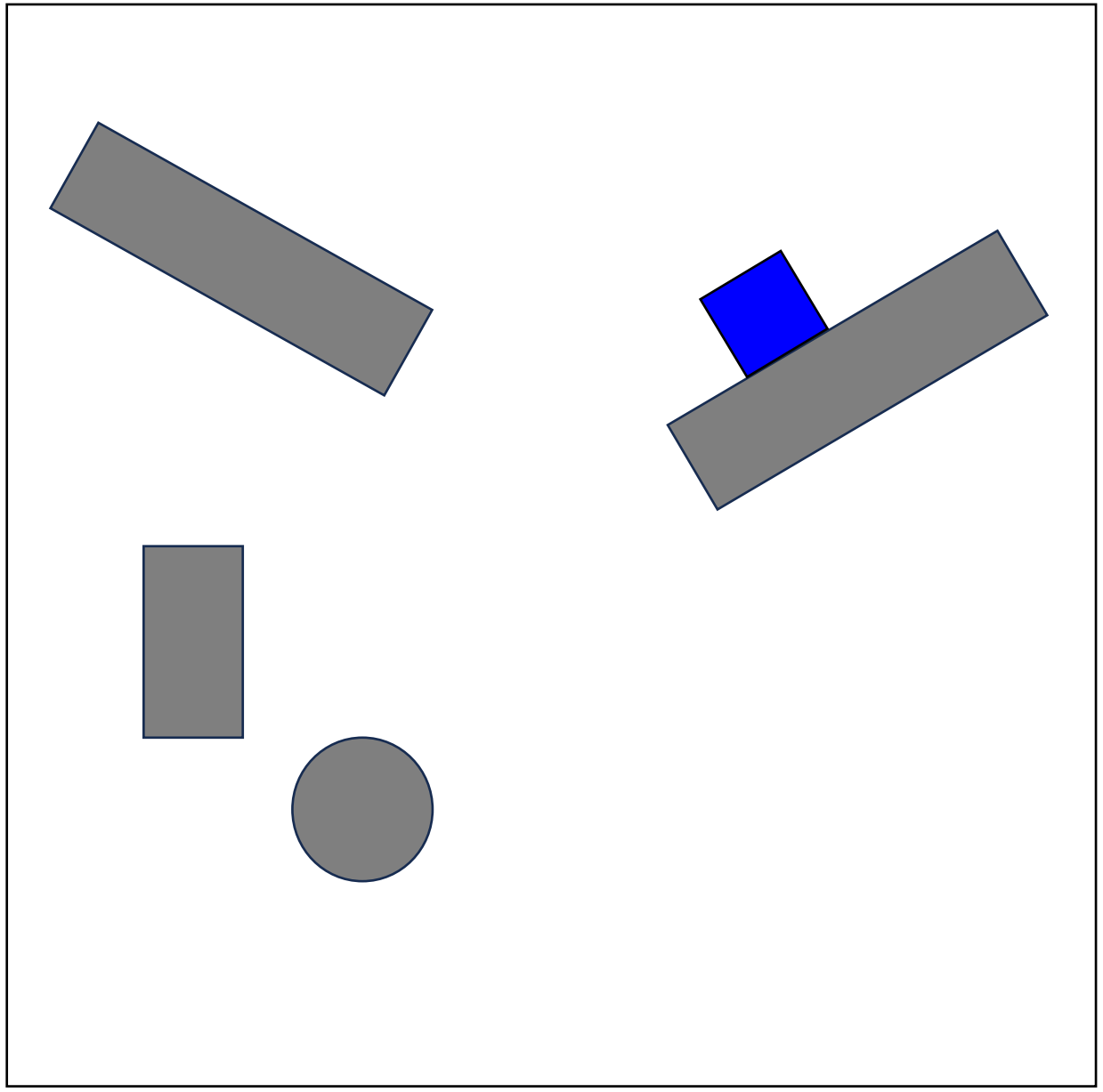
Reuse graph for new query

QueryPRM(x_0, x_g, graph, f)

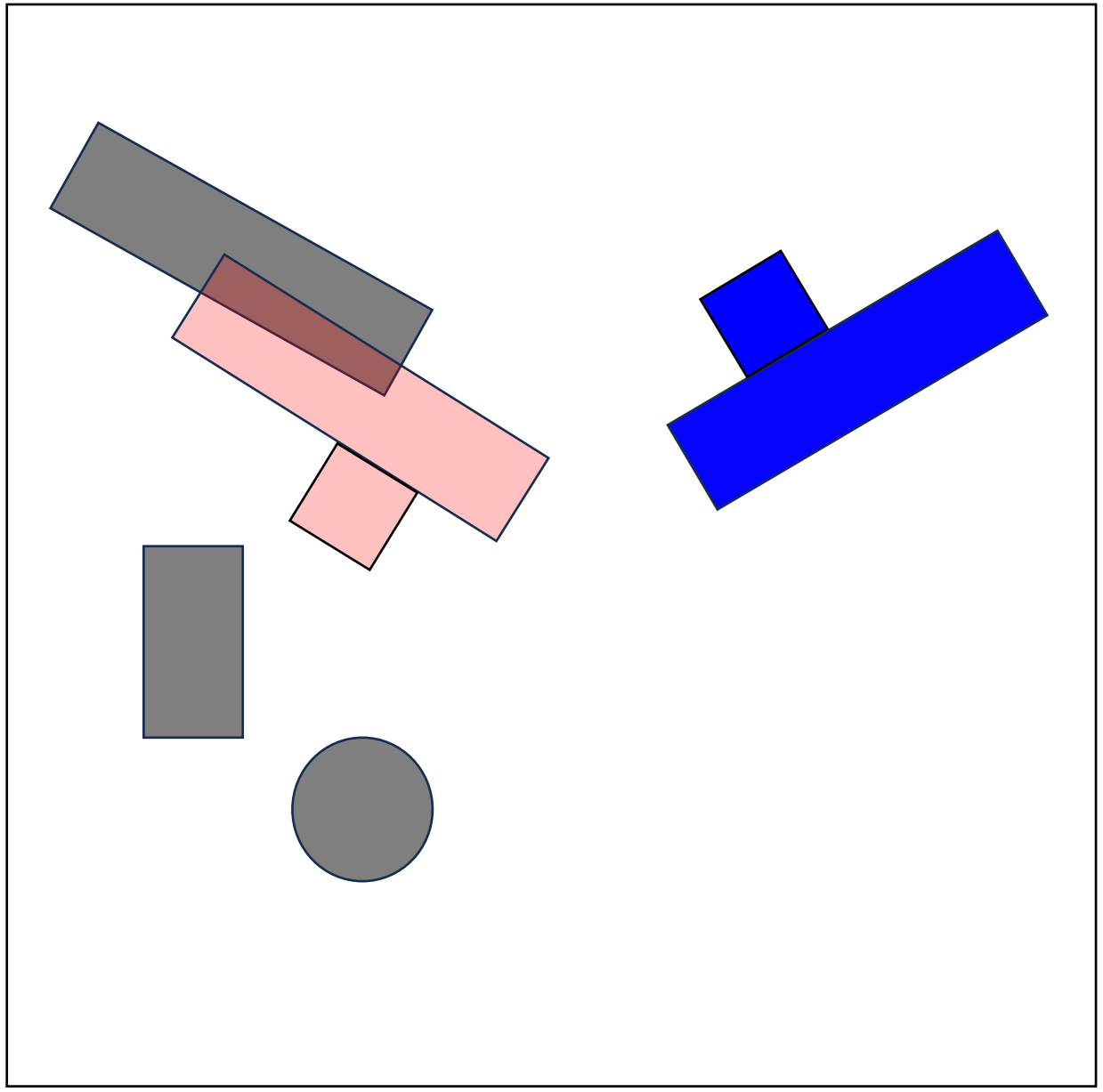
```
initNode = UpdatePRM(x_0, graph, f)
goalNode = UpdatePRM(x_g, graph, f)
nodePath = graphSearch(initNode, goalNode)
return finish(nodePath)
```

Morning Agenda

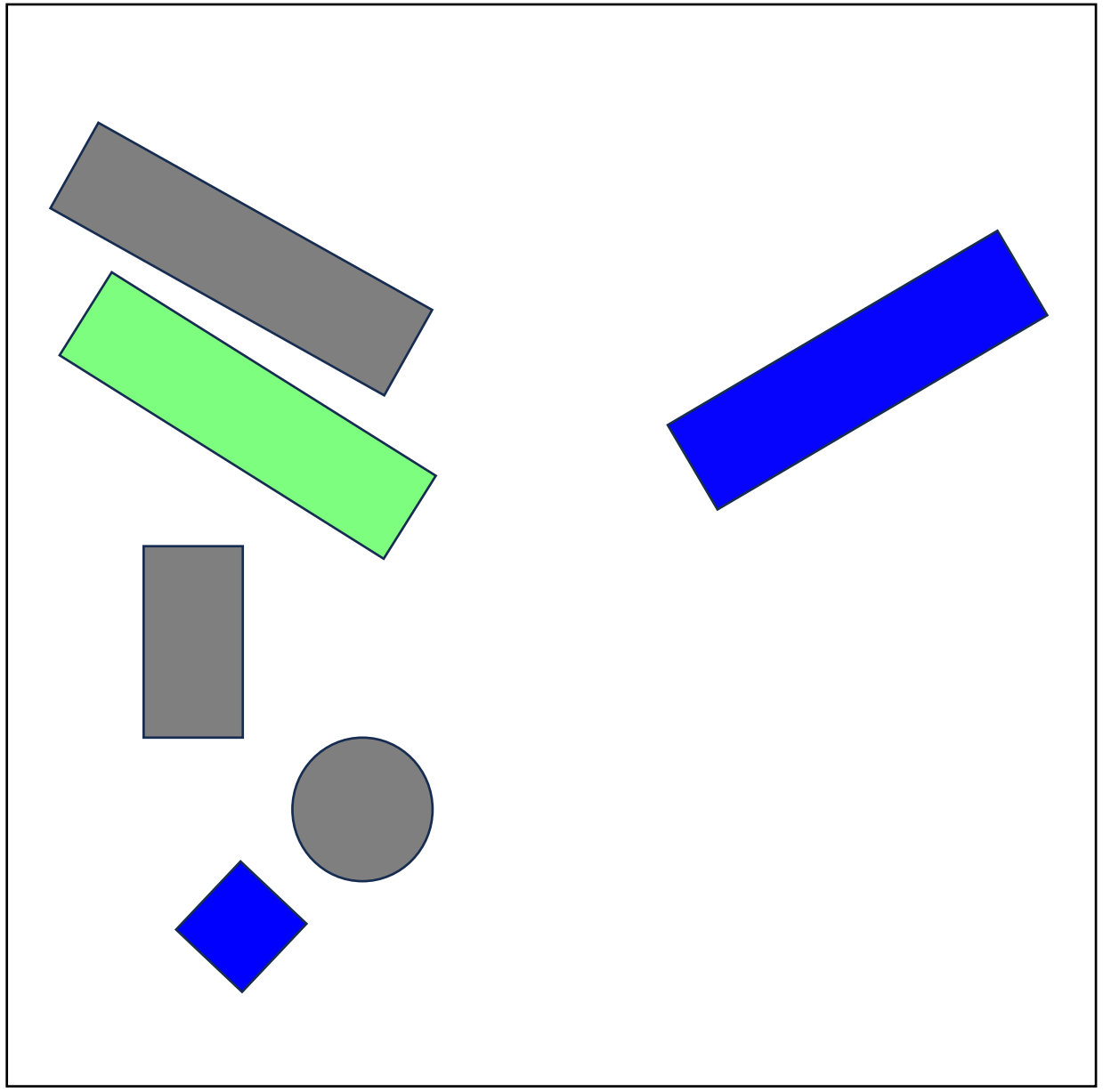
- From Task Planning to Bilevel Planning
- Introduction to Motion Planning
- **Multi-Modal Motion Planning and Beyond**
- TAMP in the Real World (TipTop)



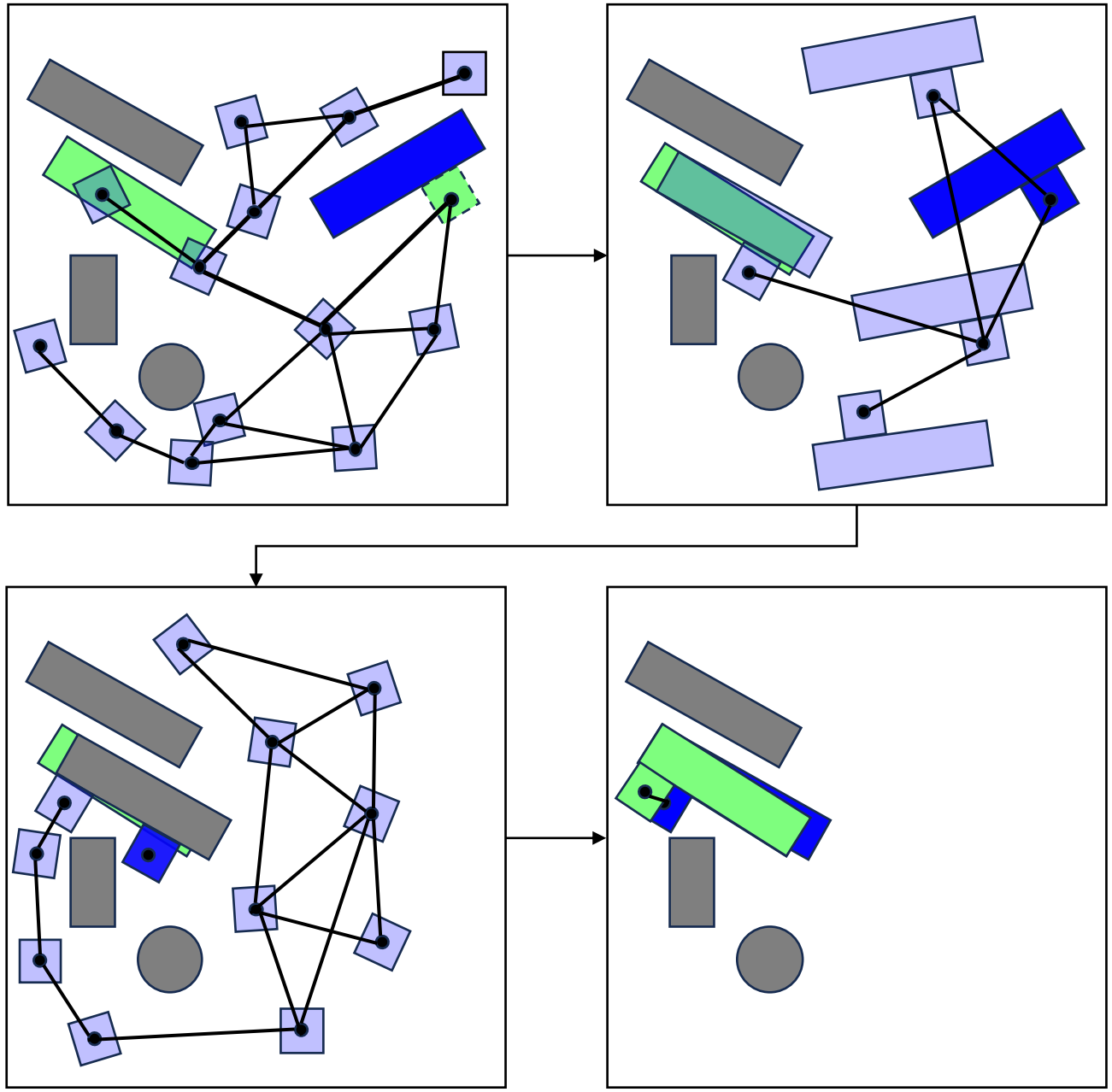
Suppose the robot can attach and move obstacles



When attached, feasibility checks change…



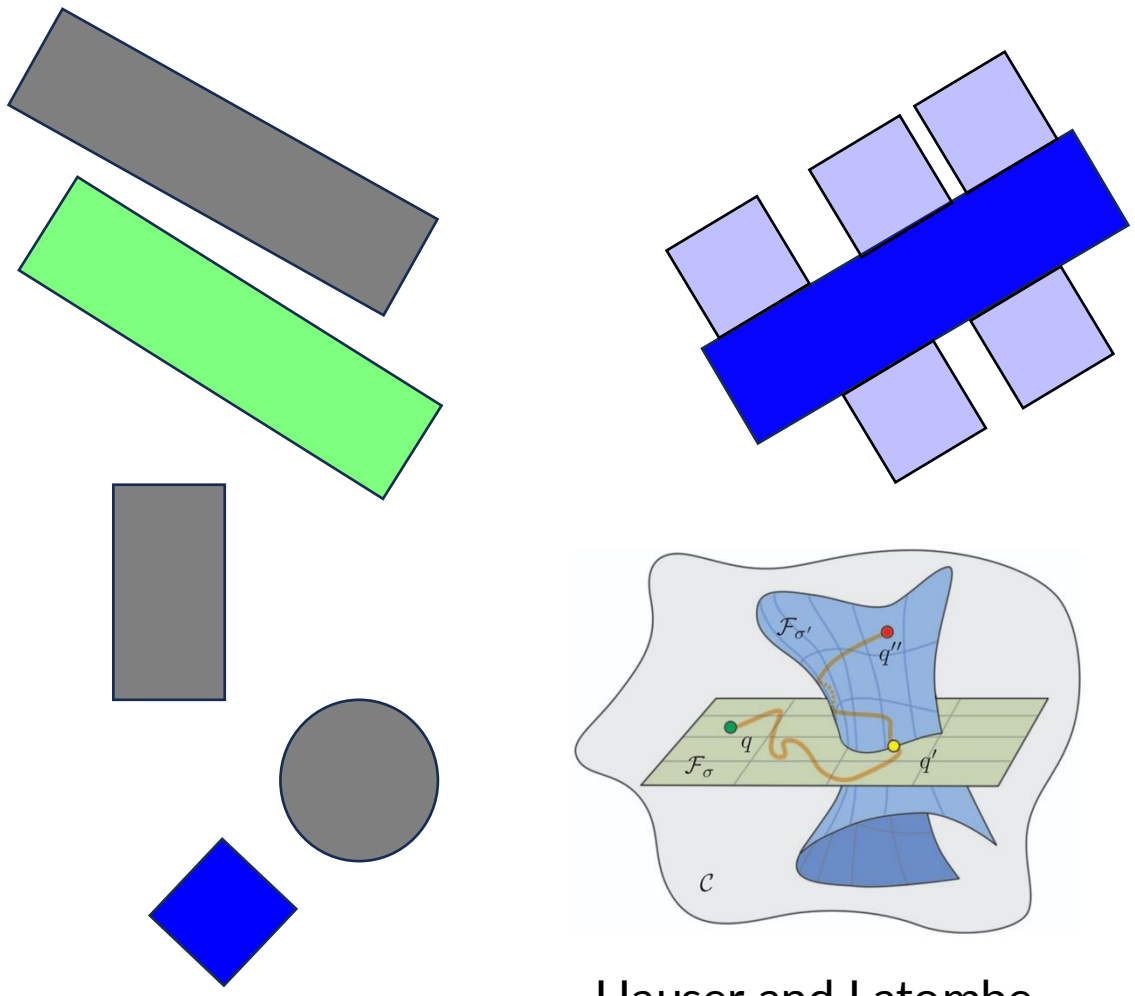
Suppose robot can only attach on long side. How to reach this goal?



Let's call each panel a "mode" = one MP instance

Could we have used original PRM in third panel?

We swept something under the rug… what's the issue?

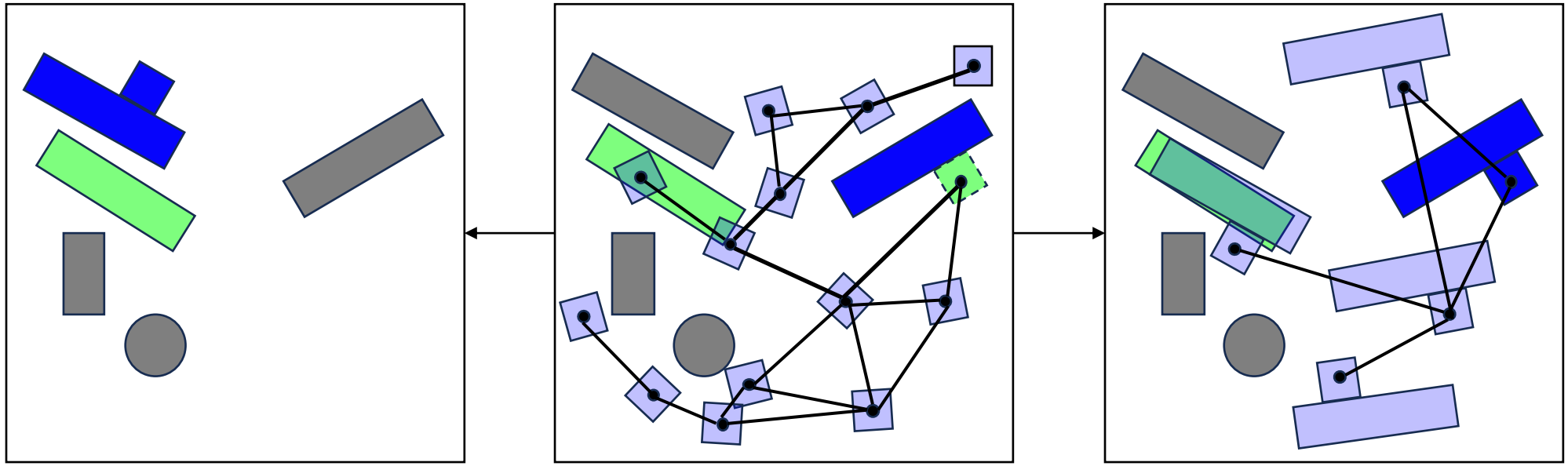


Hauser and Latombe (IJRR 2011)

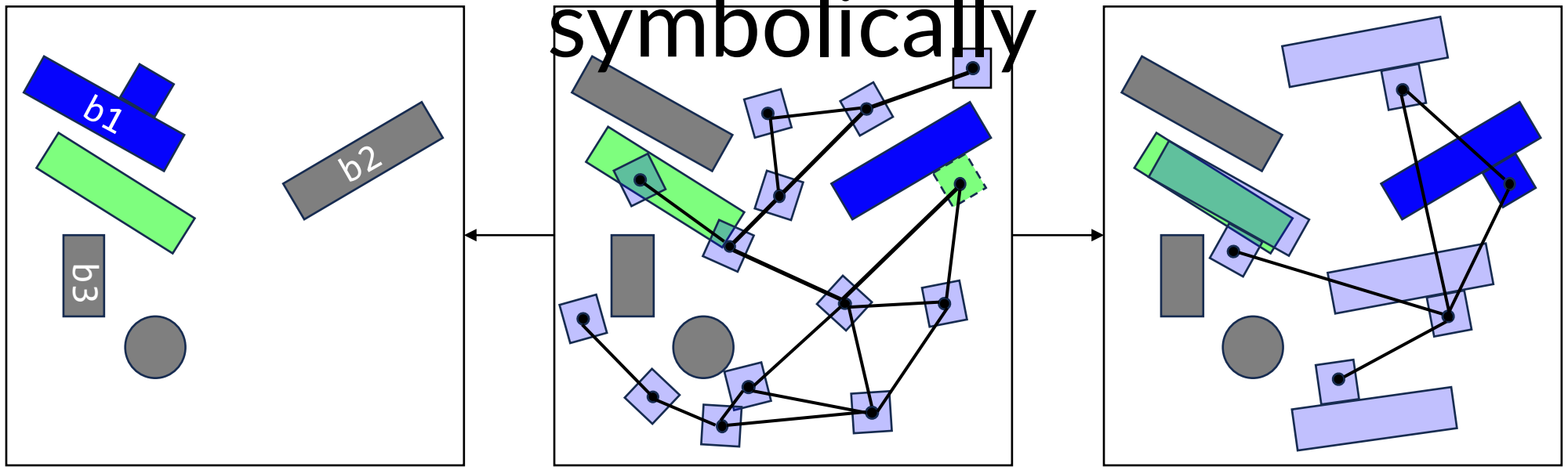
We need to be able to sample not only *within modes* (normal motion planning) but also at *mode intersections*

Reduced dimensionality!

We also don’t know the mode order…

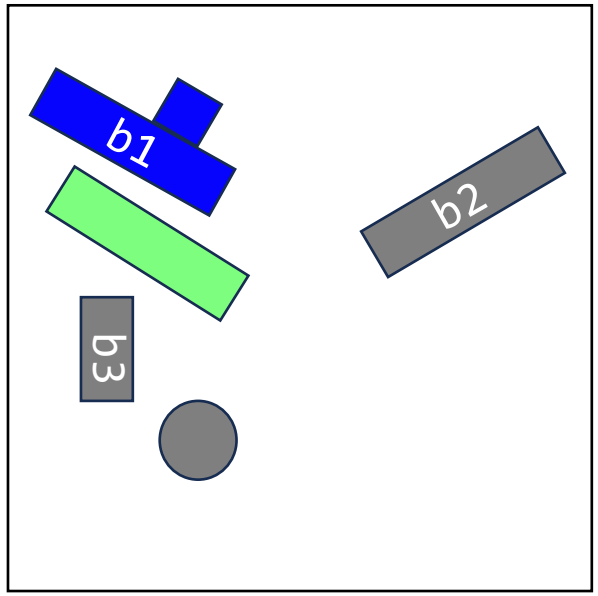


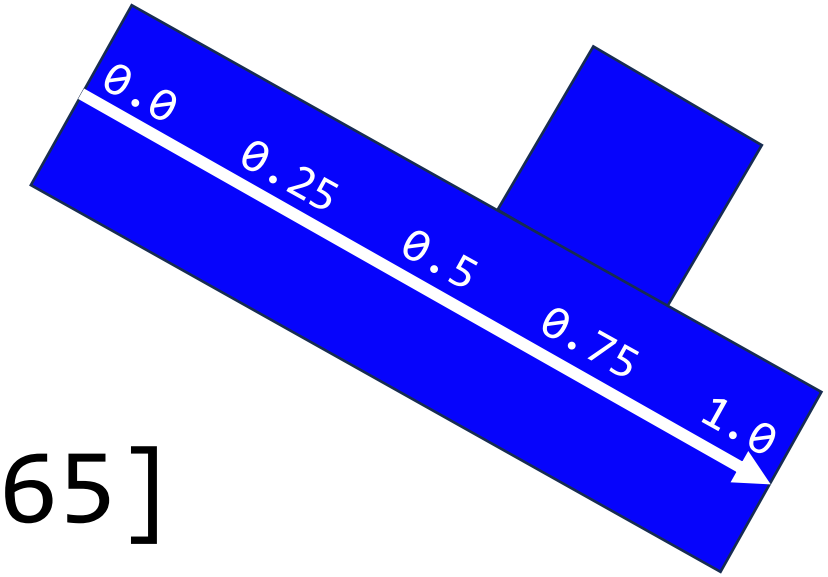
Let's describe these modes symbolically



$\texttt{Holding(b1, }\theta_1\texttt{)}$ $\texttt{BlockAt(b2, }\theta_2\texttt{)}$ $\texttt{BlockAt(b3, }\theta_3\texttt{)}$	$\texttt{BlockAt(b1, }\theta_4\texttt{)}$ $\texttt{BlockAt(b2, }\theta_2\texttt{)}$ $\texttt{BlockAt(b3, }\theta_3\texttt{)}$	$\texttt{BlockAt(b1, }\theta_4\texttt{)}$ $\texttt{Holding(b2, }\theta_5\texttt{)}$ $\texttt{BlockAt(b3, }\theta_3\texttt{)}$

Our symbols have real numbers now



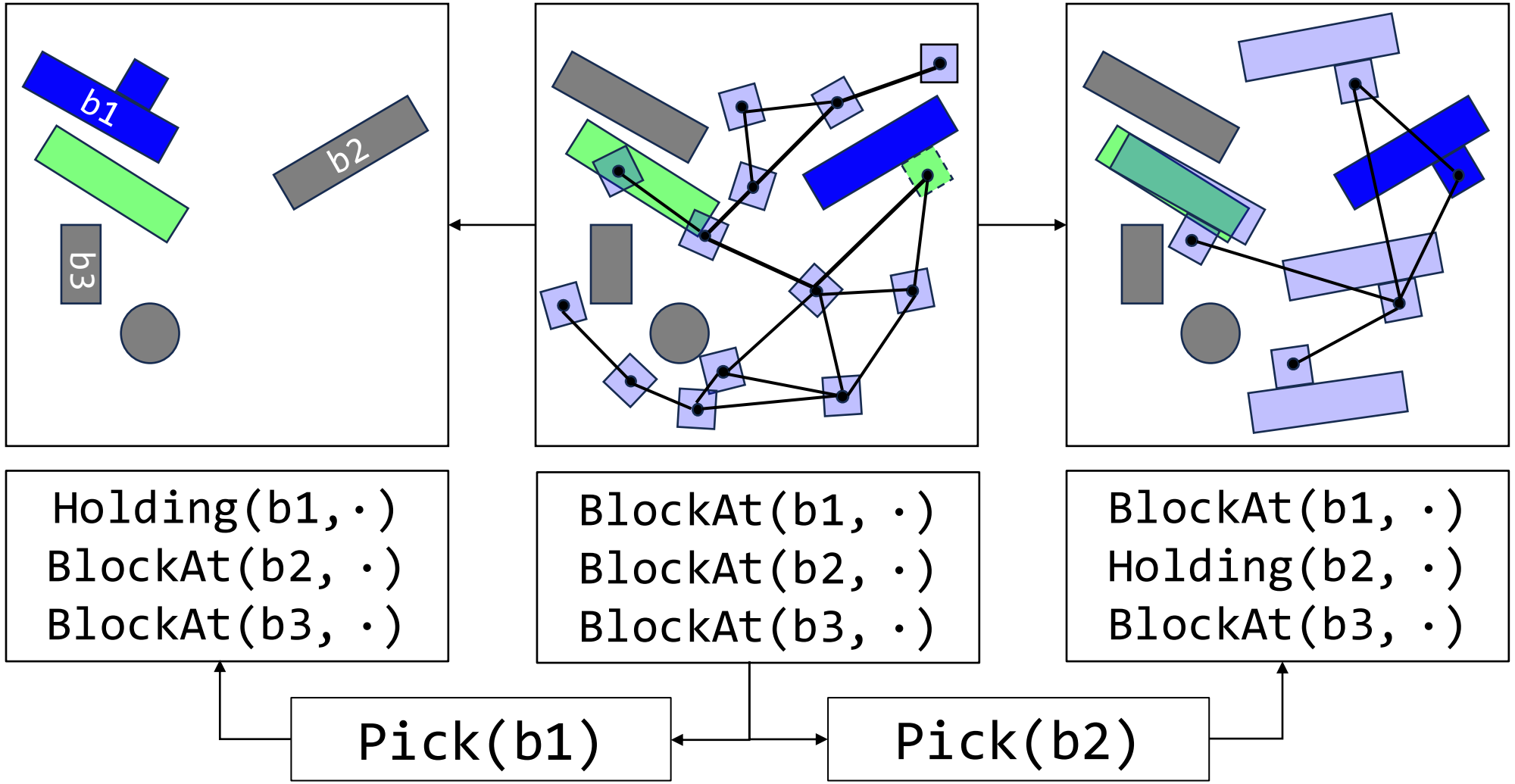


$\Theta_1 = [0.65]$

```
Holding(b1, θ1)
BlockAt(b2, θ2)
BlockAt(b3, θ3)
```

Without this, we would not have a proper mode (MP problem)

But, we can leave “holes” for the real numbers and task plan over modes!

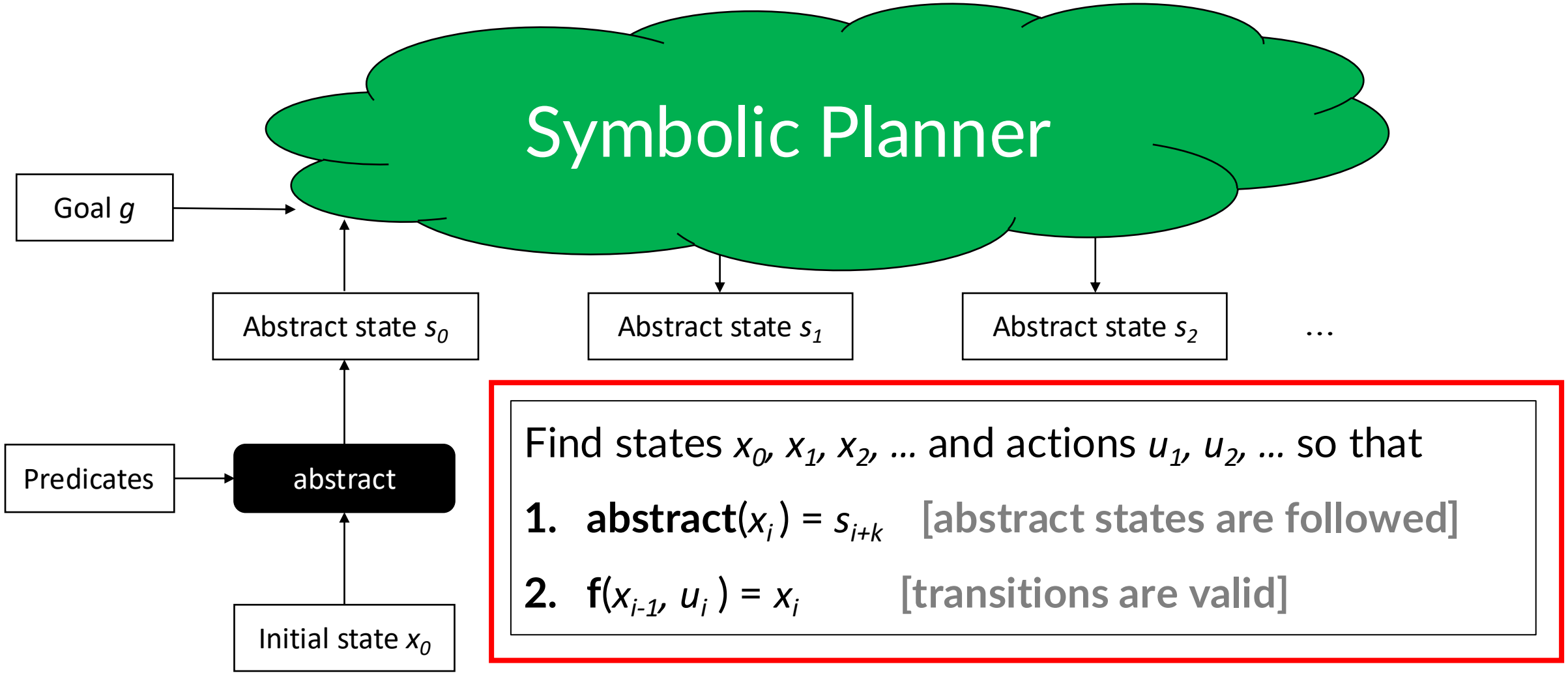


Logic-Geometric Programming

Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning

Marc Toussaint, Kelsey R Allen, Kevin A Smith & Josh Tenenbaum

In Proceedings of *Robotics: Science and Systems (R:SS 2018)*


Symbolic Planner
Goal g
Abstract state s_0
Abstract state s_1
Abstract state s_2
...
Predicates
abstract
Initial state x_0
Find states x_0, x_1, x_2, ... and actions u_1, u_2, ... so that
1. **abstract**$(x_i) = s_{i+k}$ [abstract states are followed]
2. **f**$(x_{i-1}, u_i) = x_i$ [transitions are valid]
Solve this directly (without policies)!

Morning Agenda

- From Task Planning to Bilevel Planning
- Introduction to Motion Planning
- Multi-Modal Motion Planning and Beyond
- **TAMP in the Real World (TipTop)**

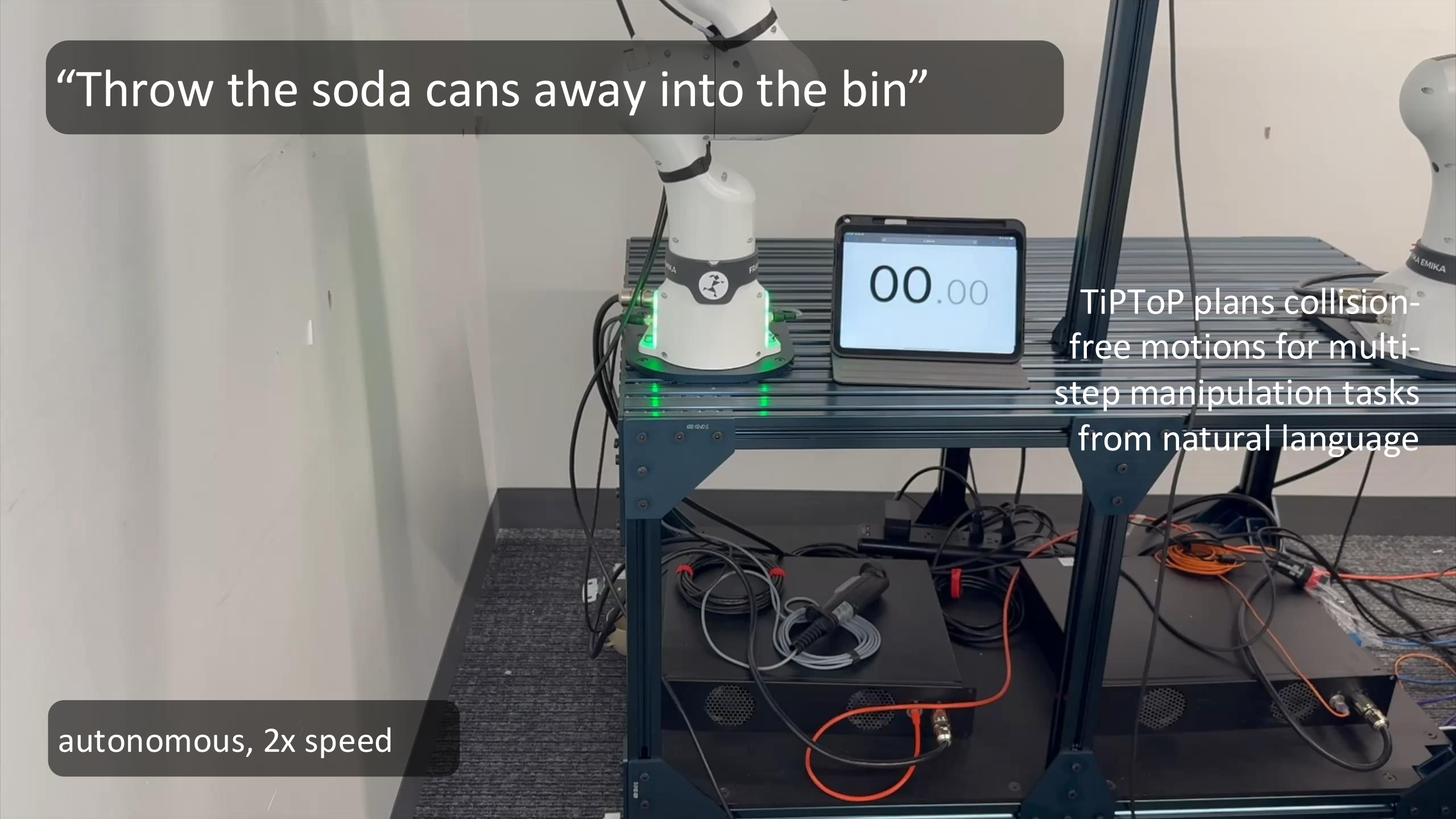
TiPToP: A Modular Open-Vocabulary Planning System for Robotic Manipulation

William Shen[1*], Nishanth Kumar[1*], Sahit Chintalapudi[1], Jie Wang[2], Christopher Watson[2], Edward S. Hu[2], Jing Cao[1], Dinesh Jayaraman[2], Leslie Pack Kaelbling[1], Tomás Lozano-Pérez[1]

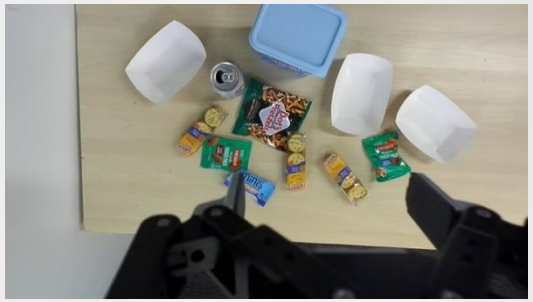
[1] LIS Group, MIT CSAIL [2] University of Pennsylvania

*Indicates equal contribution.

Slides adapted from Nishanth Kumar (thanks!)


“Throw the soda cans away into the bin”
00.00
TiPToP plans collision-free motions for multi-step manipulation tasks from natural language
autonomous, 2x speed

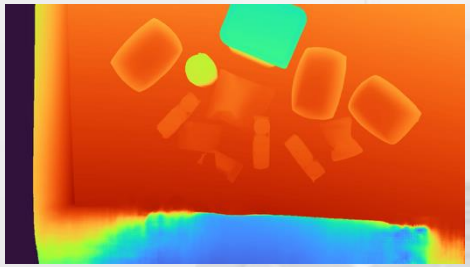
Image



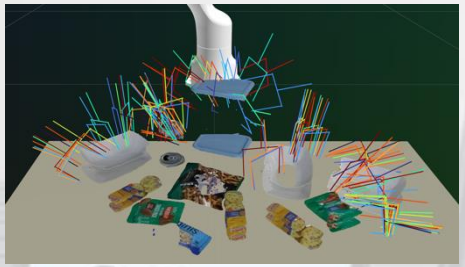
Task Instruction

“Serve peanut butter crackers on each tray”

Depth Prediction



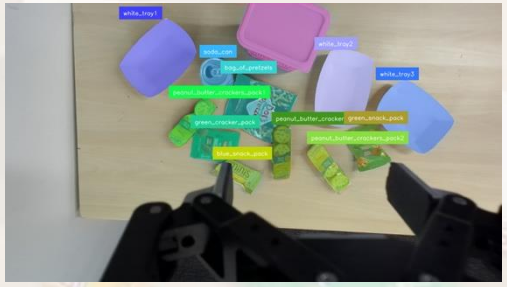
Grasp Prediction



3D Vision Branch

Object Detection

Object Segmentation



Semantic Branch

Task and Motion Planner

cuTAMP

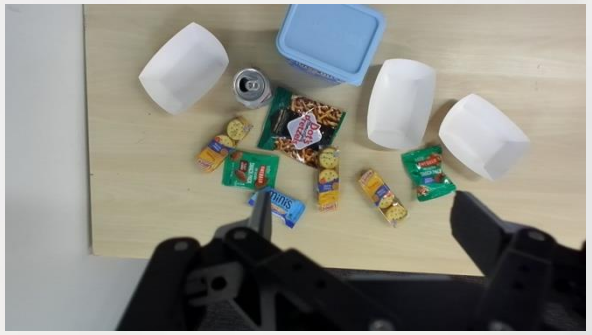
Image

Wrist Camera

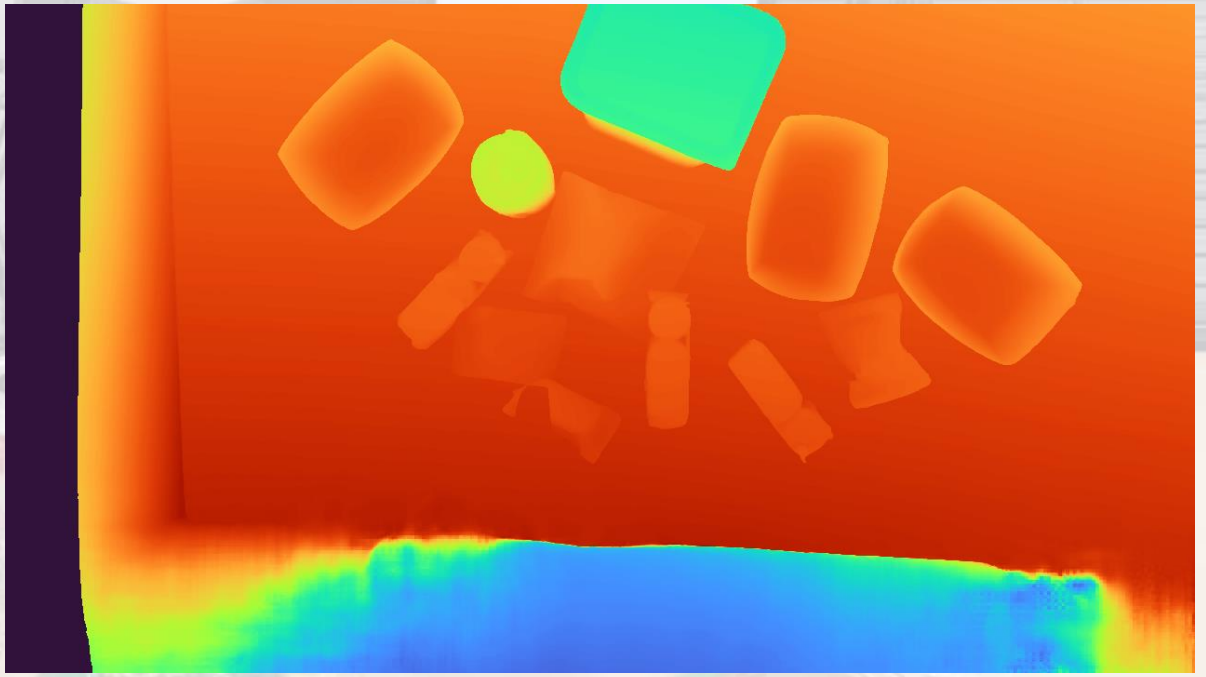
Task Instruction

“Serve peanut butter crackers on each tray”

Image



Depth Prediction

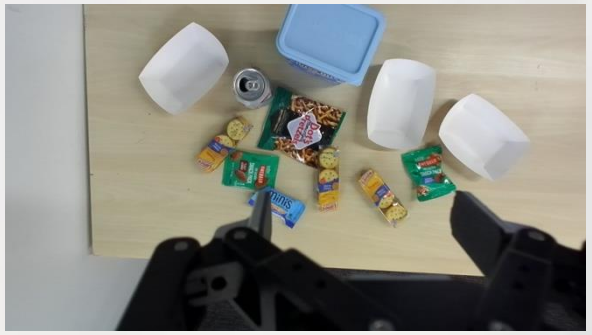


Task Instruction

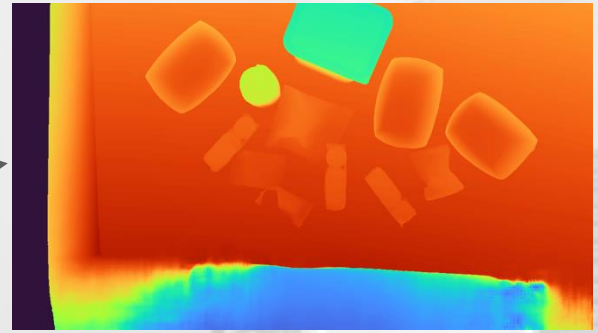
“Serve peanut butter crackers on each tray”

“FoundationStereo”, Wen et. al. 2025.

Image



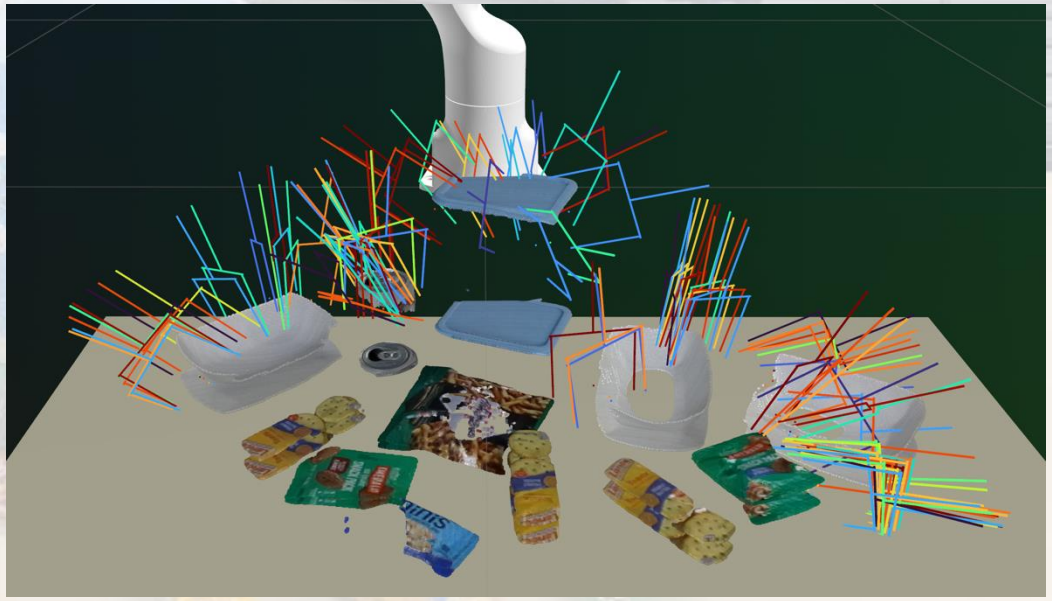
Depth Prediction



Project to 3D Point Cloud

Task Instruction

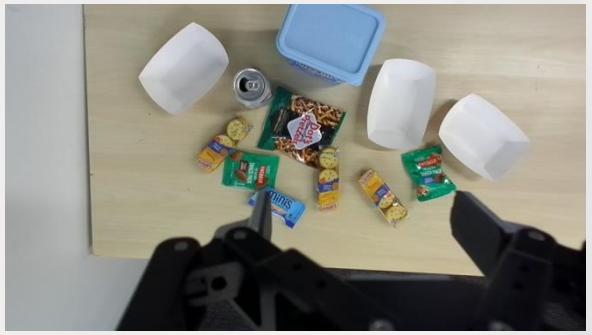
“Serve peanut butter crackers on each tray”



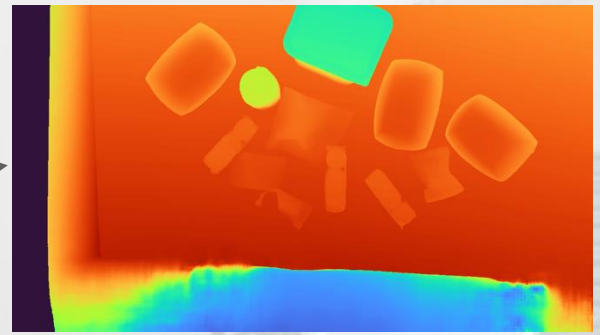
6-DOF Grasp Prediction

“M2T2”, Yuan et. al. 2023.

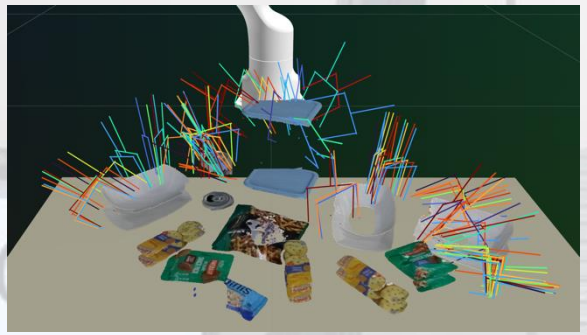
Image



Depth Prediction



Grasp Prediction

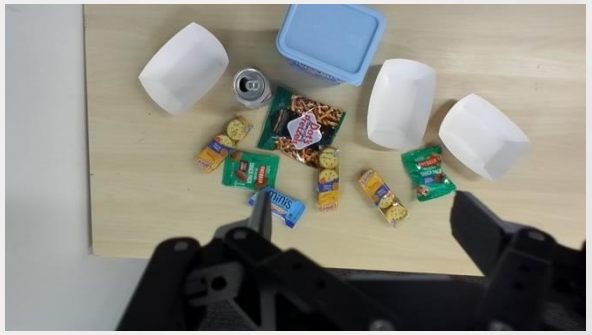


3D Vision Branch

Task Instruction

“Serve peanut butter crackers on each tray”

Image



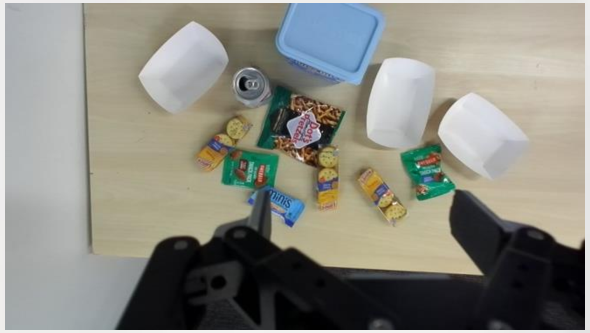
Task Instruction

“Serve peanut butter crackers on each tray”

Vision Language Model (VLM)-based Object Detection

“Gemini 1.5 ER”, Google. 2025.

Image

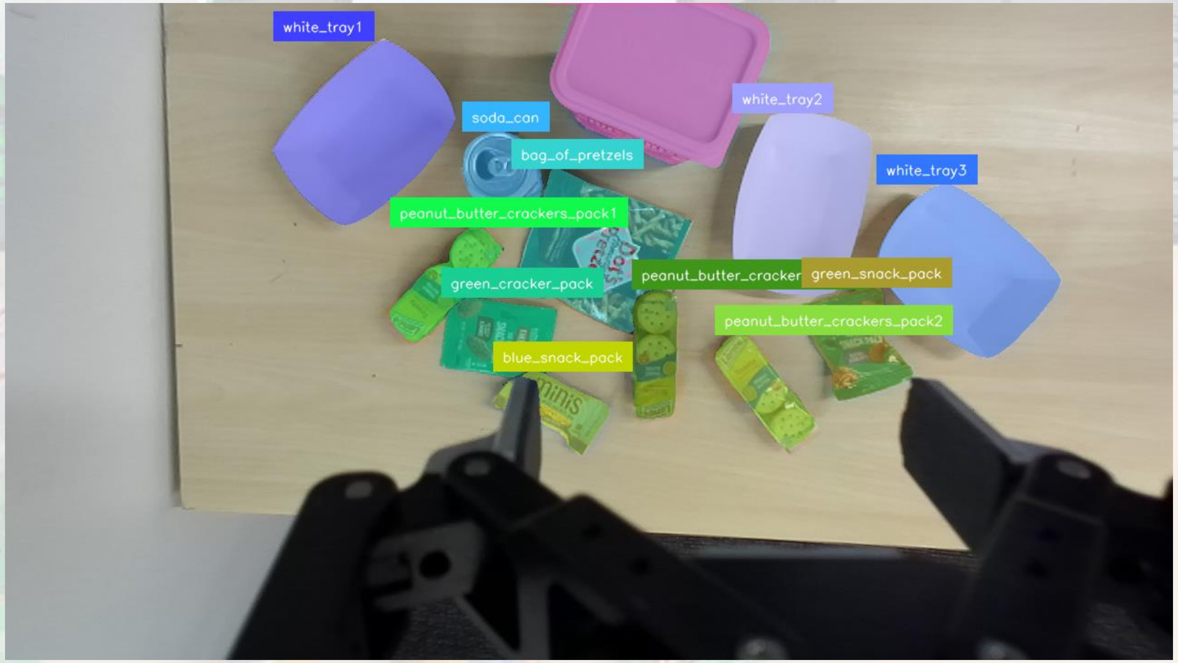


Task Instruction

“Serve peanut butter crackers on each tray”

Object Detection

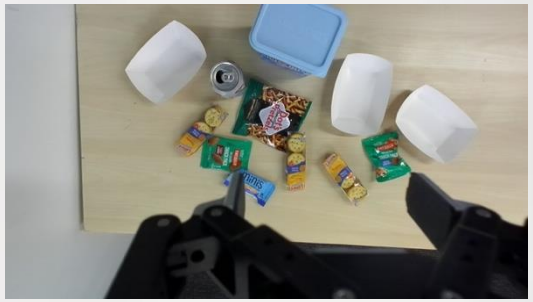
Bounding Boxes



Object Segmentation

“SAM 2”, Meta. 2024.

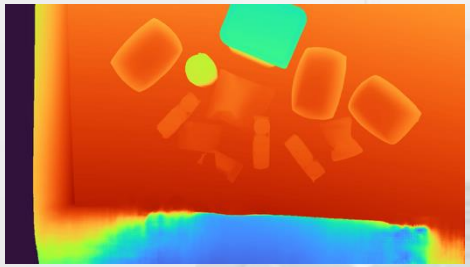
Image



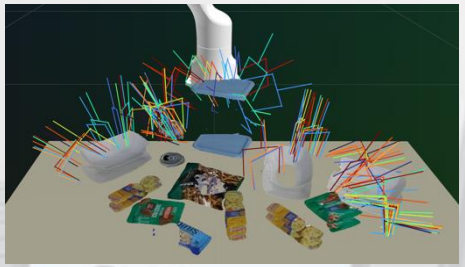
Task Instruction

“Serve peanut butter crackers on each tray”

Depth Prediction

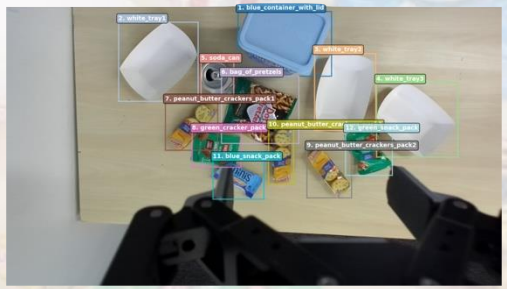


Grasp Prediction

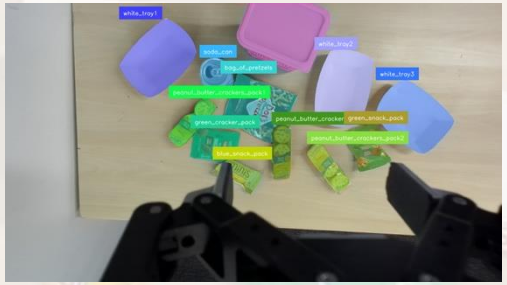


3D Vision Branch

Object Detection



Object Segmentation



Semantic Branch

Task and Motion Planner

cuTAMP

cuTAMP Optimization

"Serve peanut butter crackers on each tray"

Initial Scene

*Slowed down for visualization

TiP ToP

cuTAMP Plan

"Serve peanut butter crackers on each tray"

Initial Scene

TiP ToP

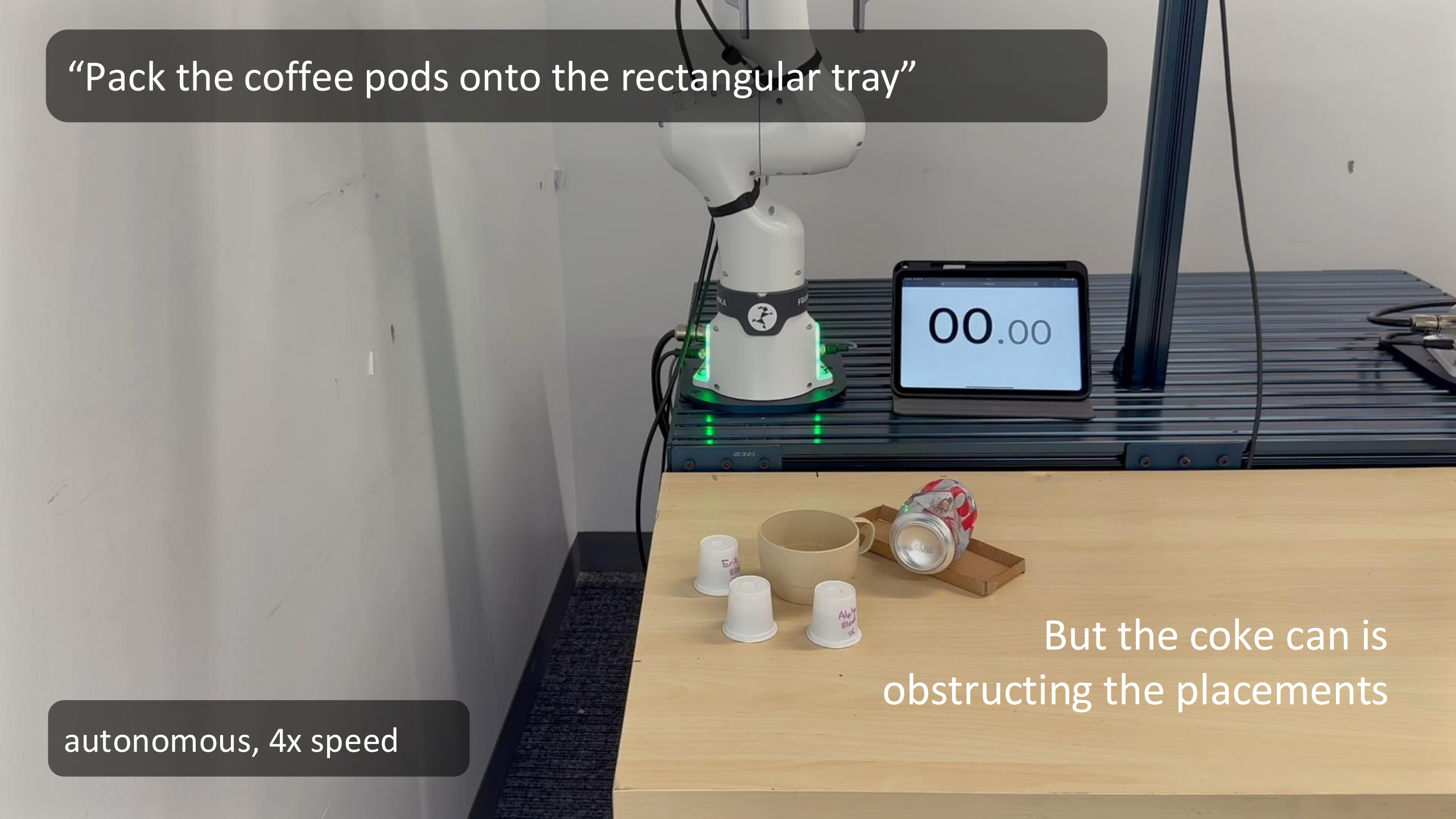
Real-World Execution

“Serve peanut butter crackers on each tray”

Initial Scene

2x speed

TiP ToP


“Pack the coffee pods onto the rectangular tray”
00.00
But the coke can is obstructing the placements
autonomous, 4x speed

TiPToP

A **pip-installable** compositional manipulation system!

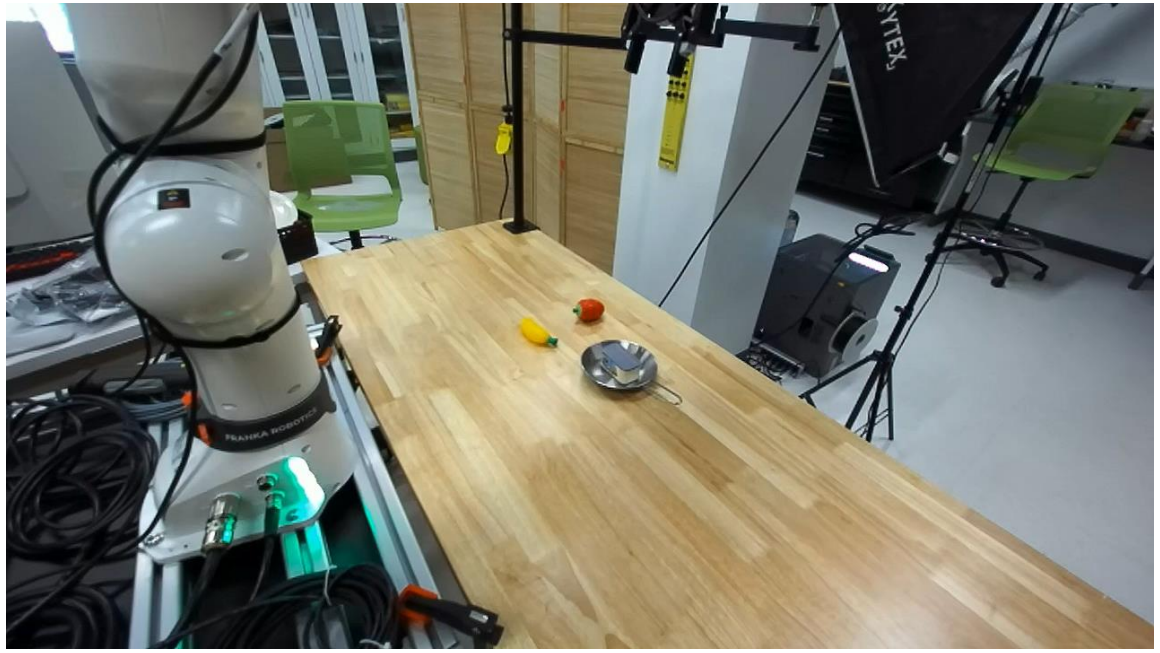
Generalizes to unseen objects, environments, and embodiments

Running TipTop at Princeton

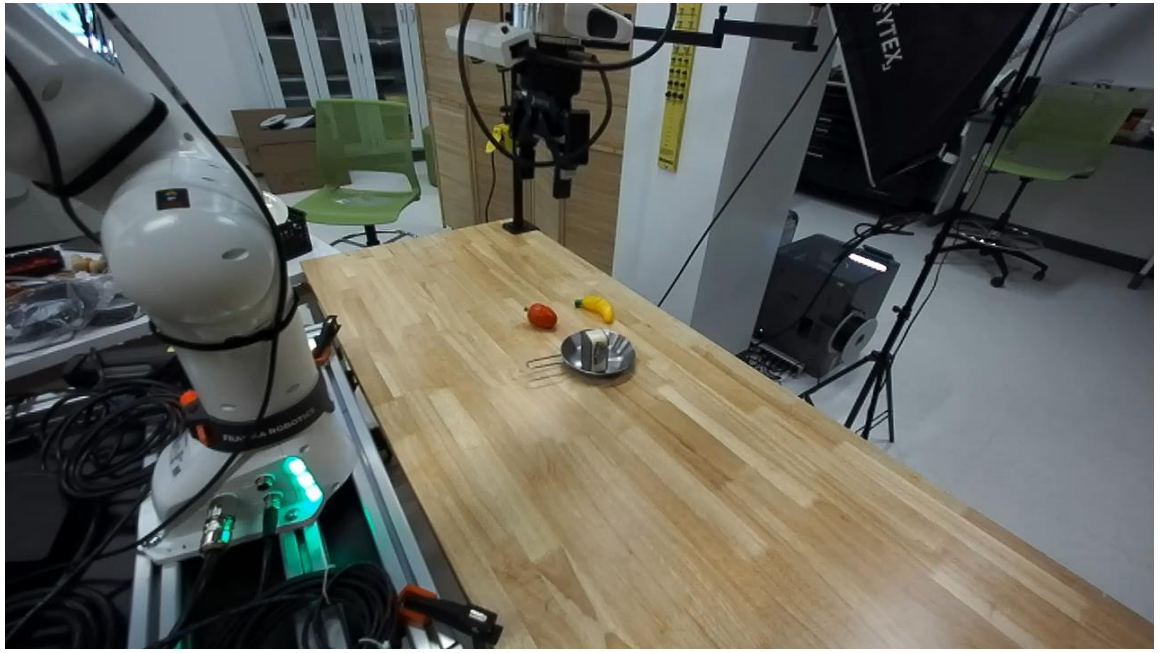
Pack only the fruits into the pan

Samrat Sahoo

TipTop



Pi-0.5



Thanks! See Surveys For More

“Integrated Task and Motion Planning.” Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, Tomás Lozano-Pérez. ARCRAS 2021.

“A Survey of Optimization-based Task and Motion Planning: From Classical To Learning Approaches.” Zhigen Zhao, Shuo Cheng, Yan Ding, Ziyi Zhou, Shiqi Zhang, Danfei Xu, Ye Zhao. IEEE/ASME Transactions on Mechatronics 2024.