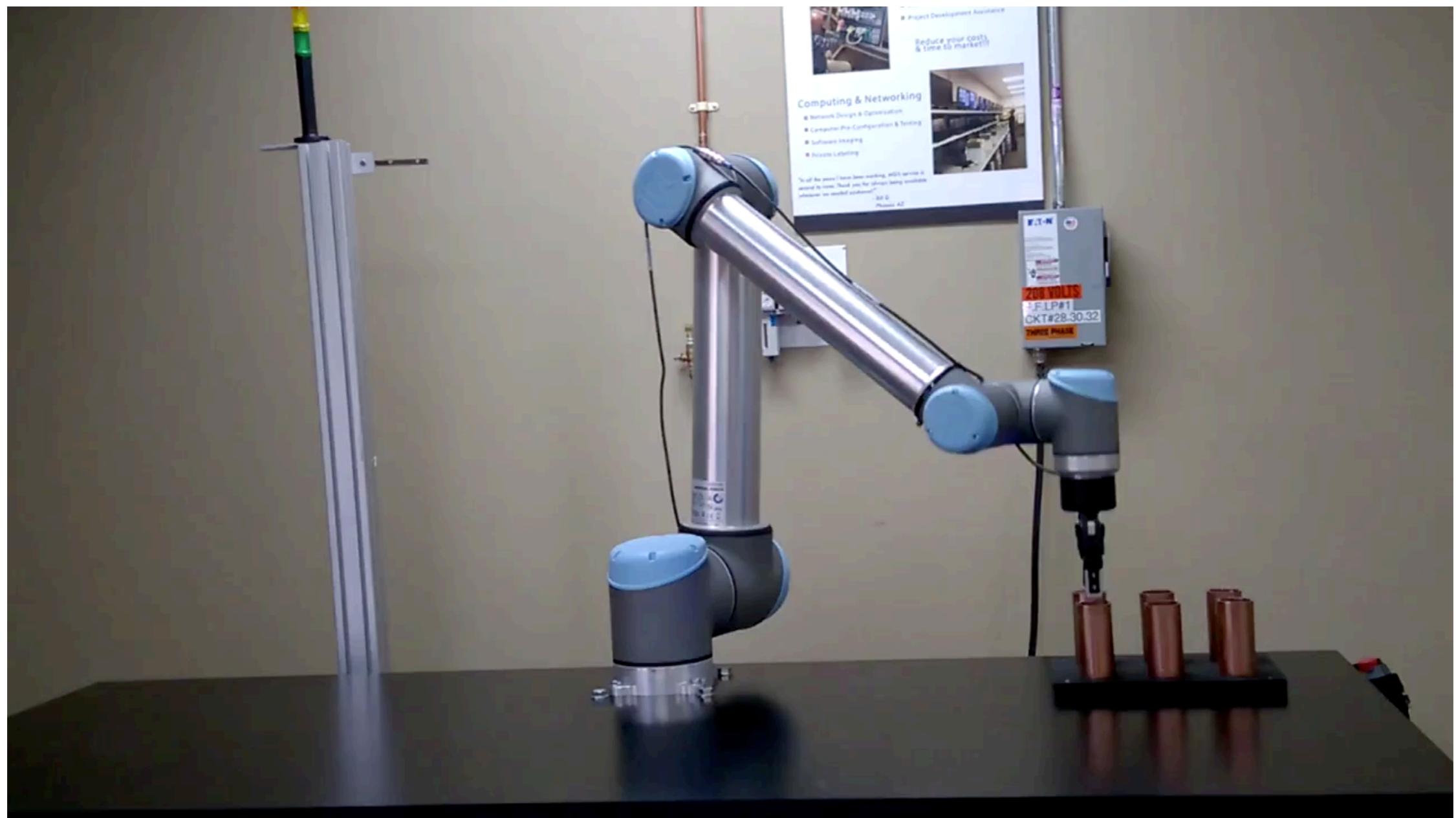


# Robotics Review

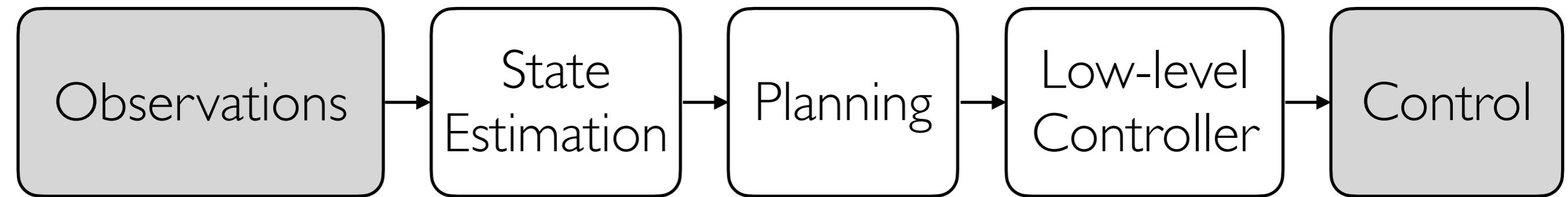
Saurabh Gupta

# Robotic Tasks

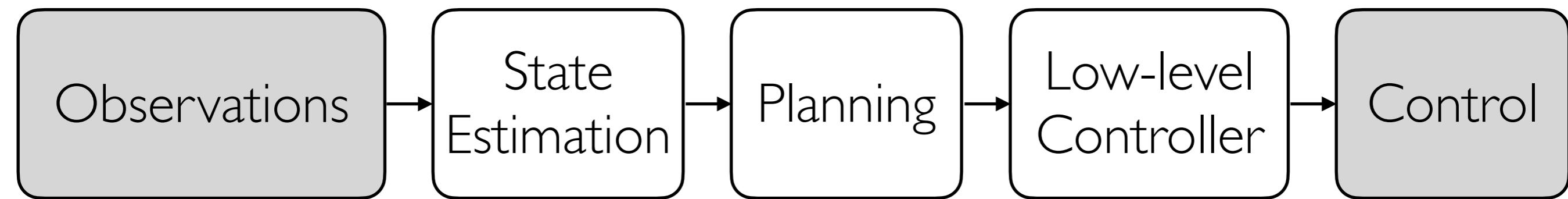
## Manipulation



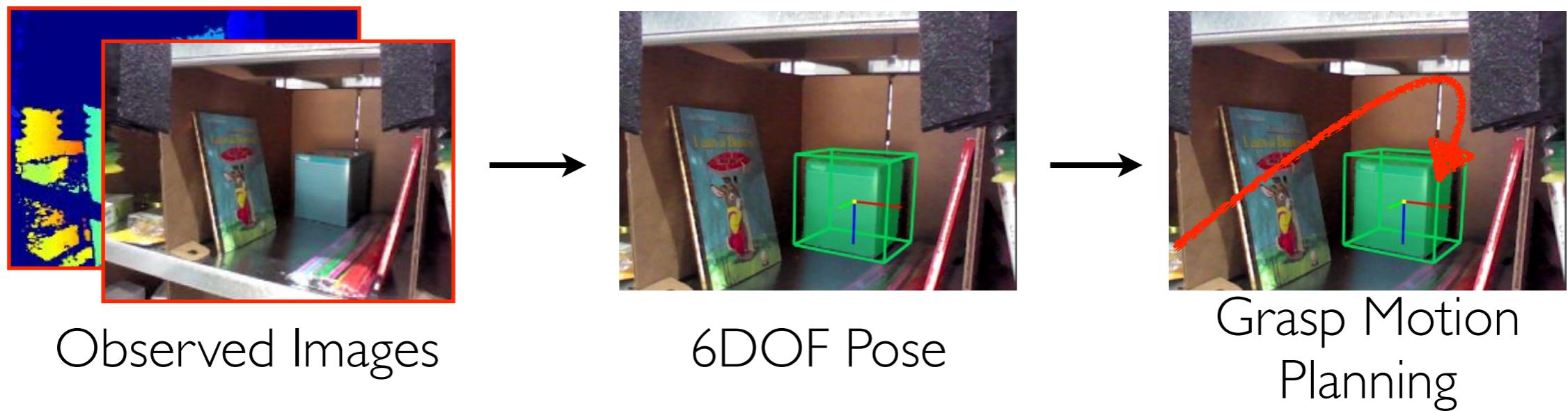
# Typical Robotics Pipeline



# Typical Robotics Pipeline



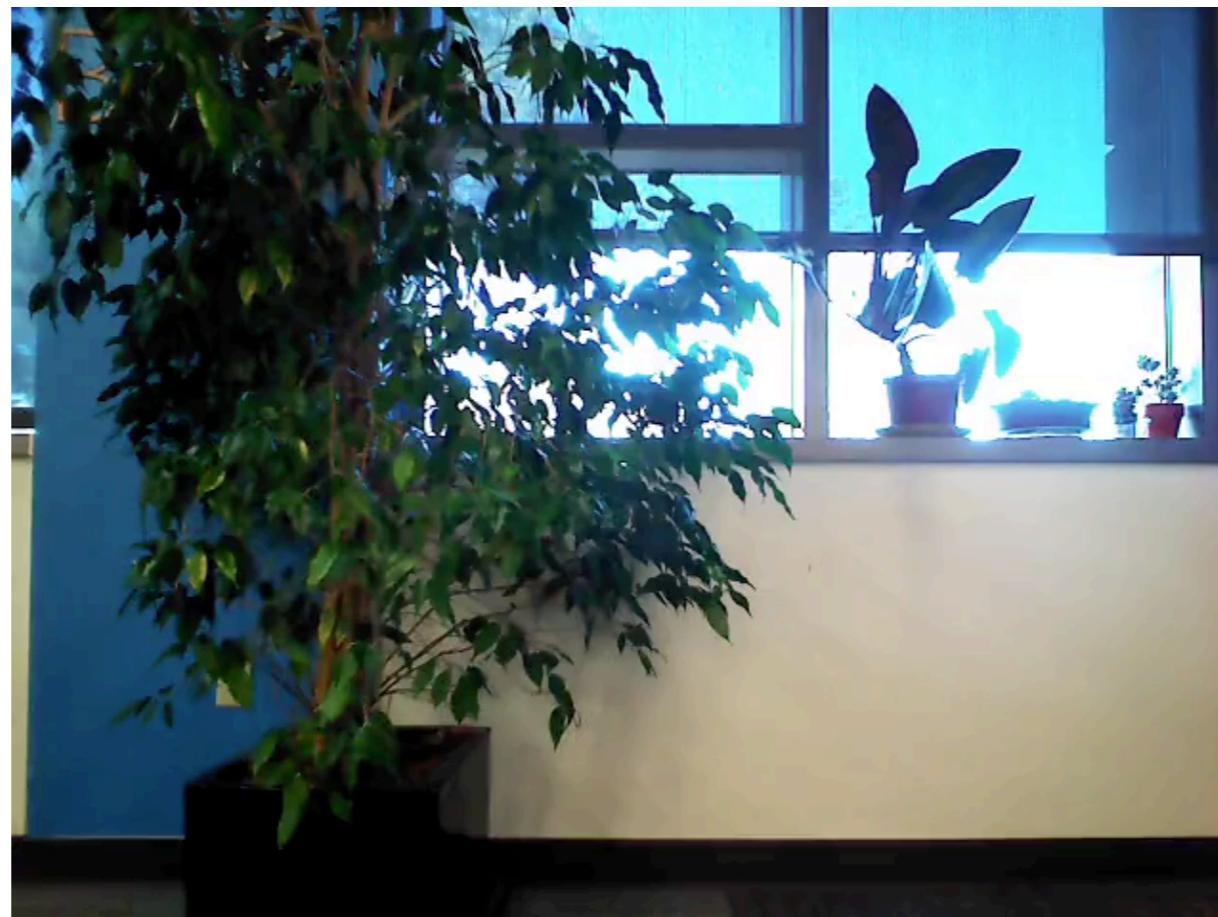
## Manipulation



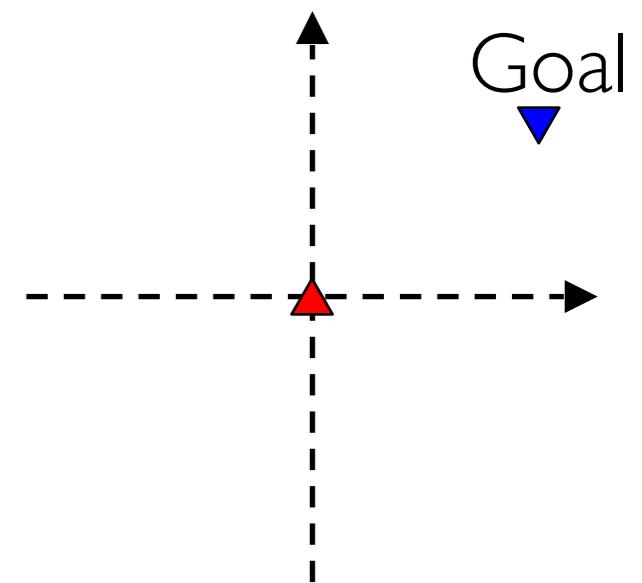
# Robot Navigation



Robot with a first person camera



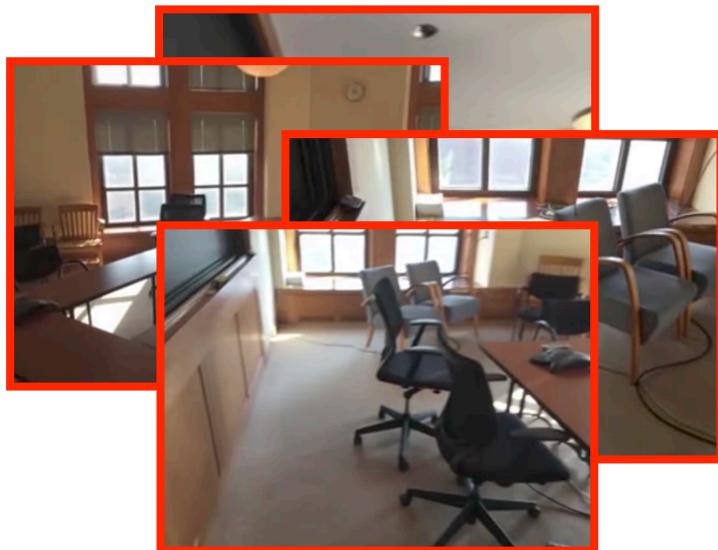
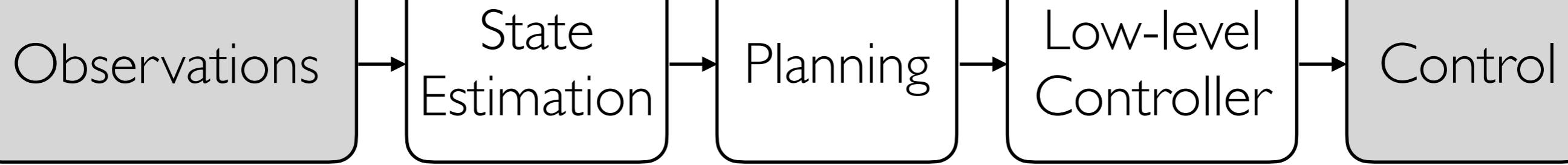
Dropped into a novel environment



“Go  
300 feet North,  
400 feet East”

“Go Find a Chair”

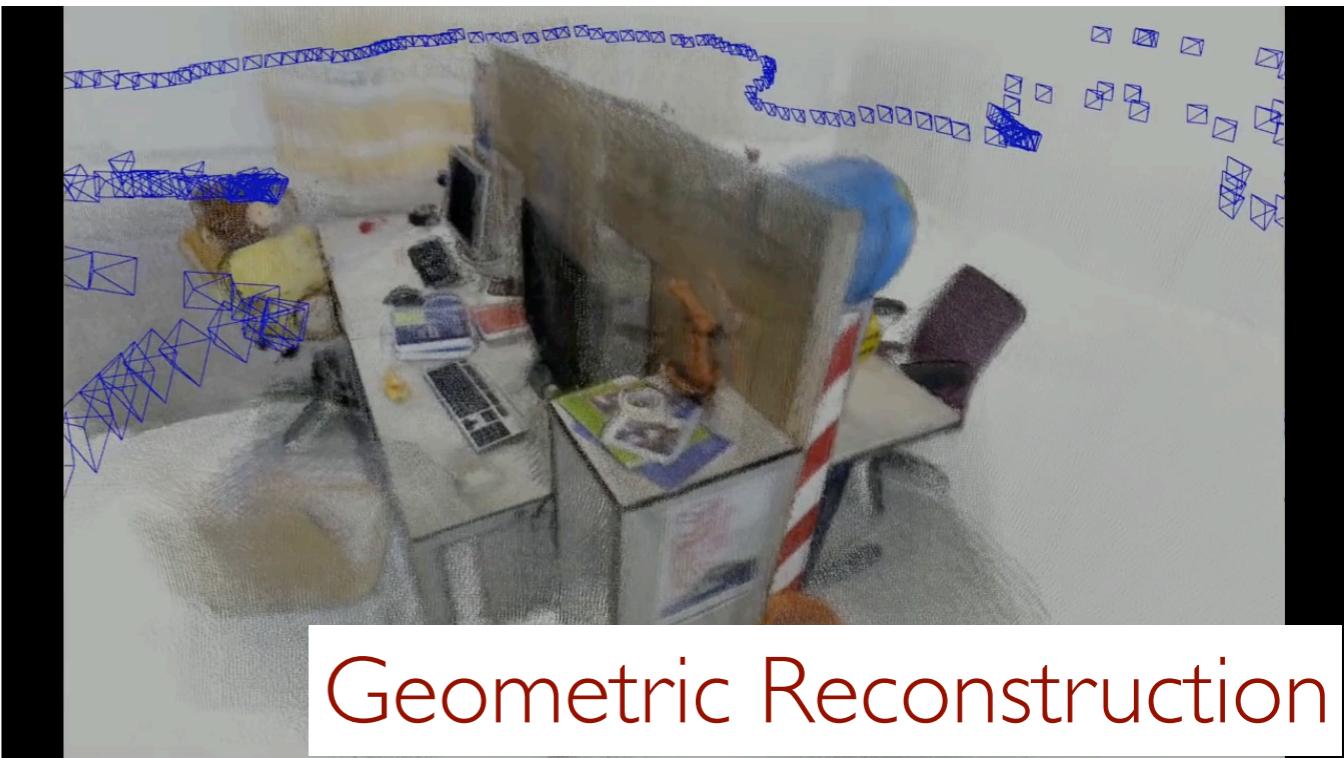
Navigate around



Observed Images

Planning

Mapping



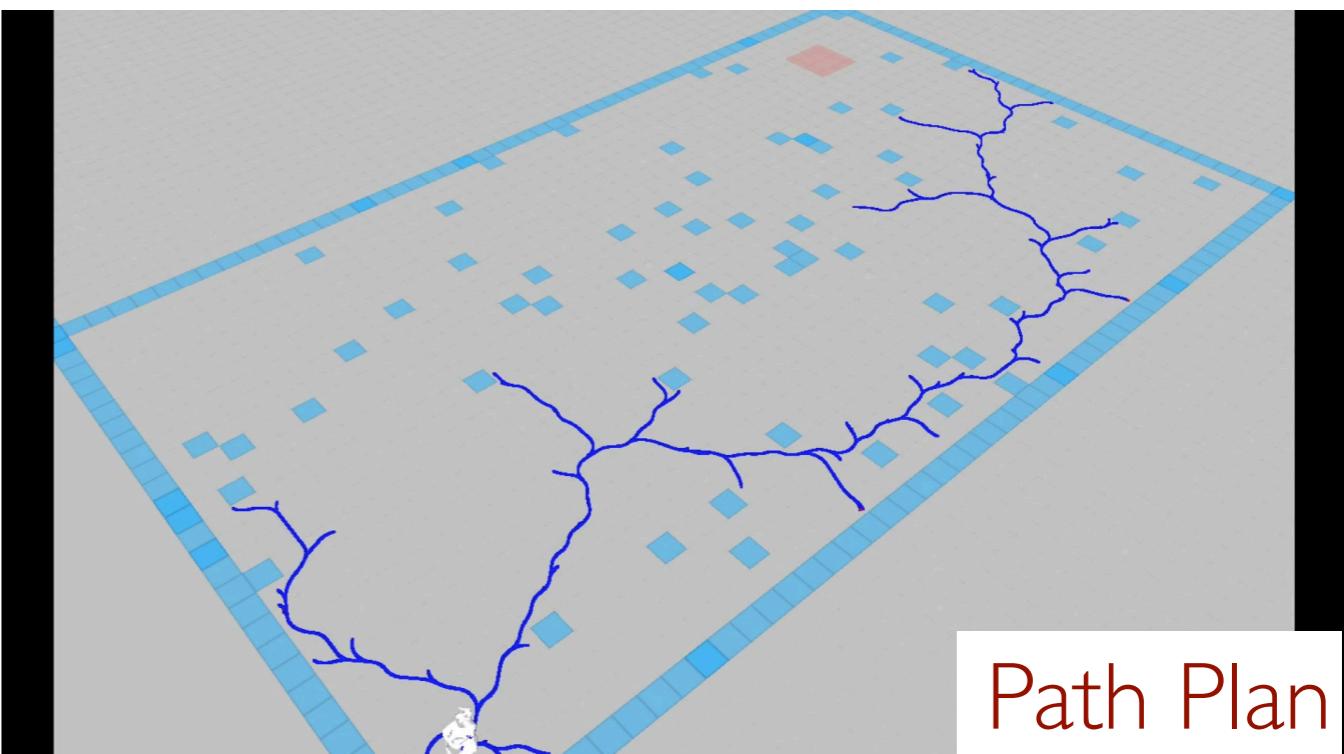
Hartley and Zisserman. 2000. Multiple View Geometry  
in Computer Vision

Thrun, Burgard, Fox. 2005. Probabilistic Robotics

Canny. 1988. The complexity of robot motion planning.

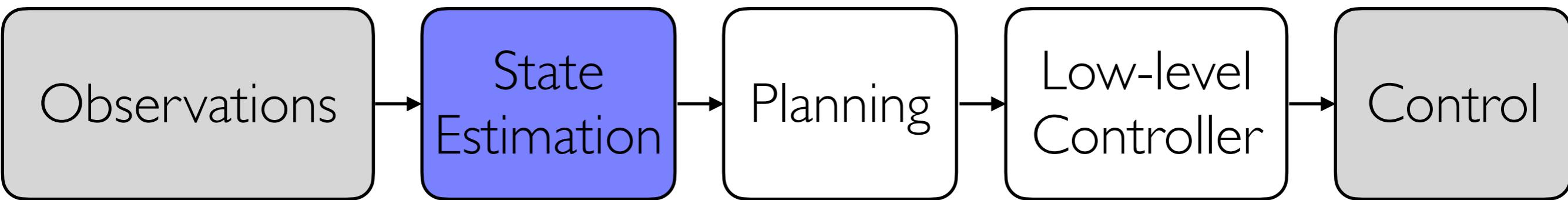
Kavraki et al. RA 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces.

Lavalle and Kuffner. 2000. Rapidly-exploring random trees: Progress and prospects.



Video Credits: Mur-Artal et al., Palmieri et al.

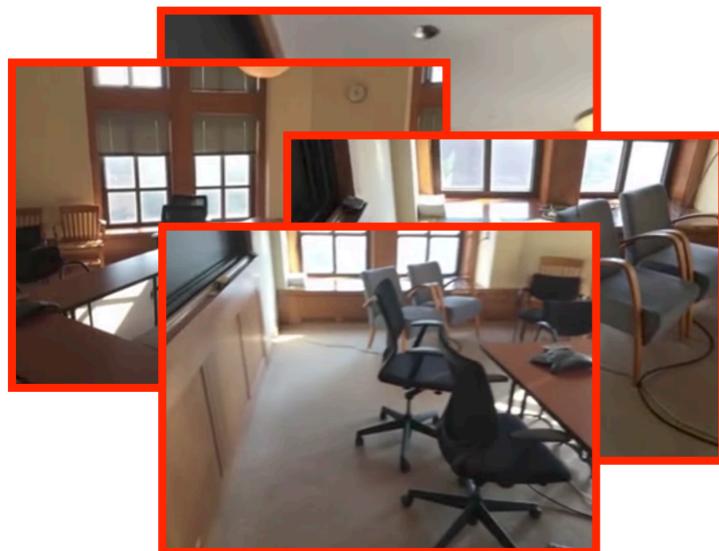
# Typical Robotics Pipeline



Observed Images



6DOF Pose

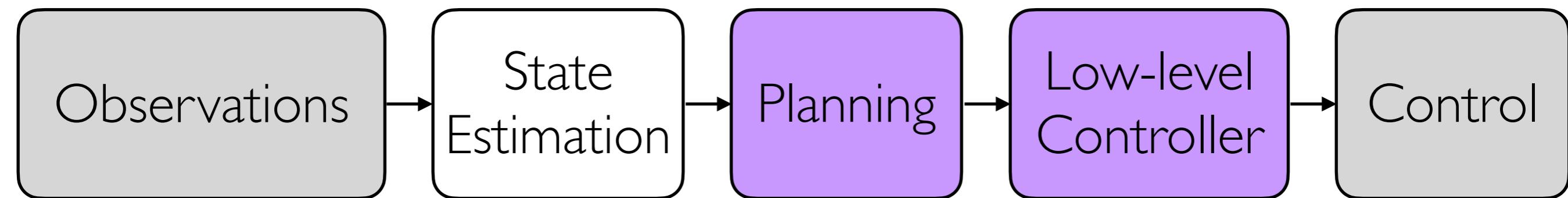


Observed Images

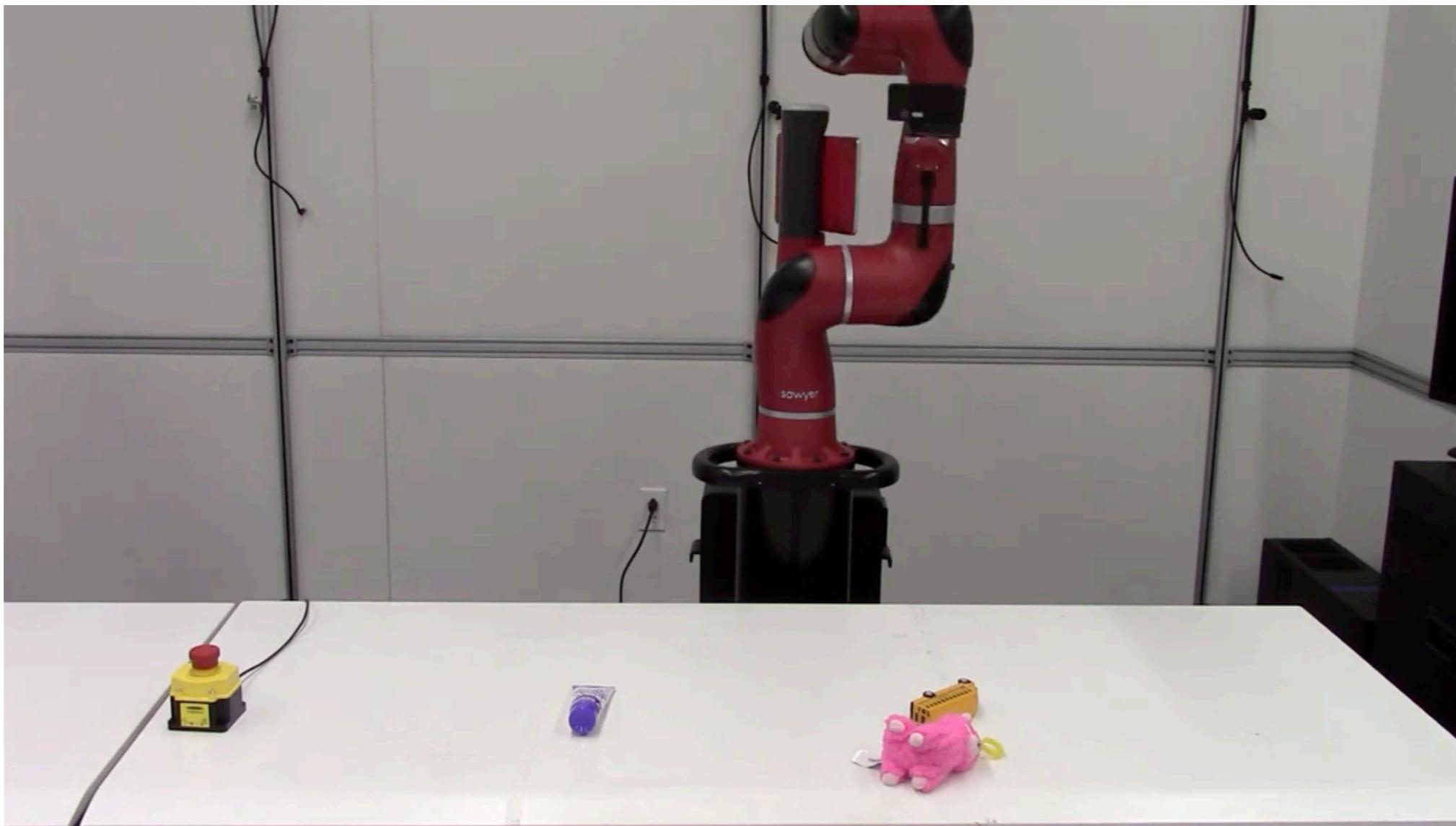


Geometric or Semantic Maps

# Typical Robotics Pipeline



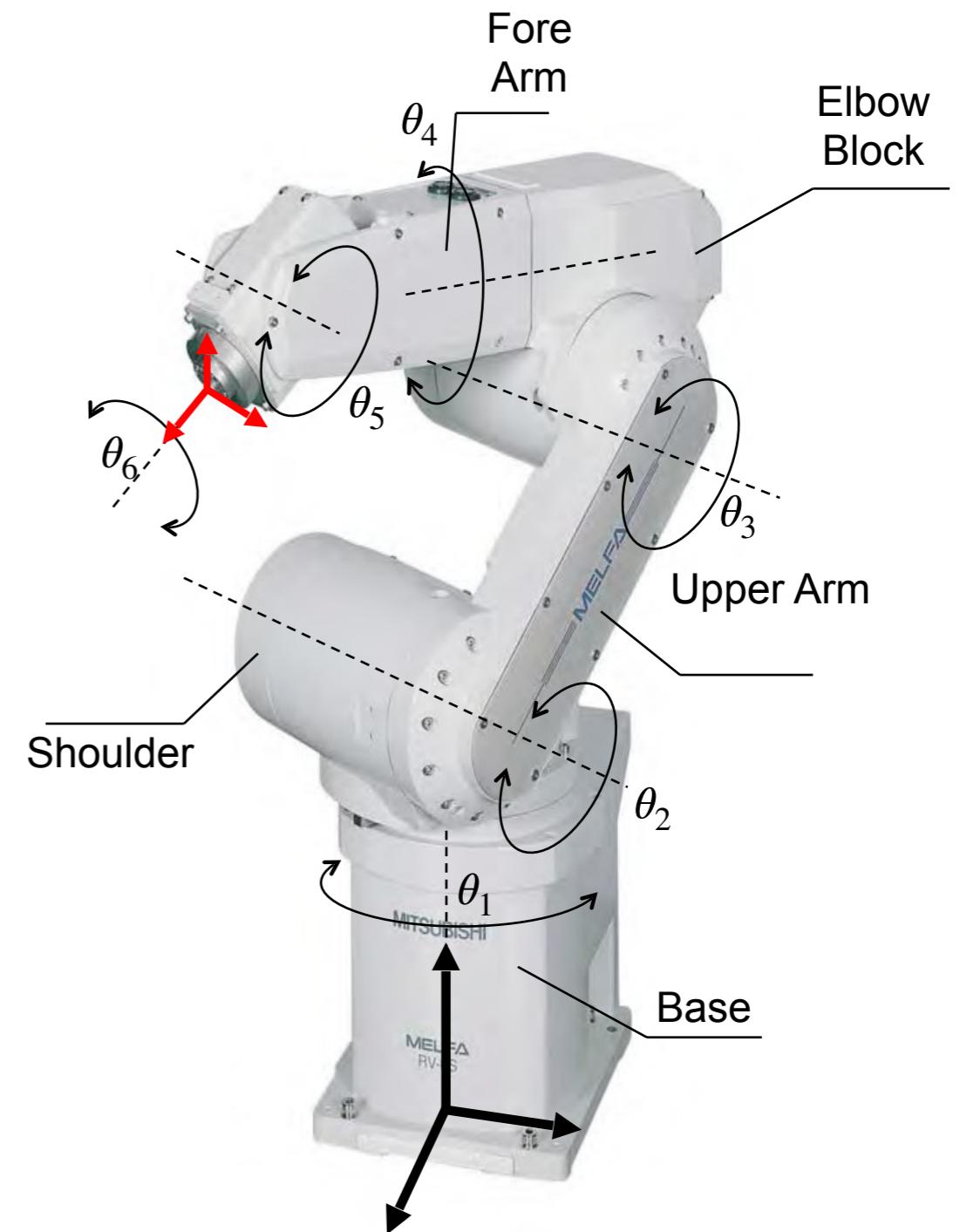
# Understand how to move a robot



Video from Deepak Pathak.

# Terminology

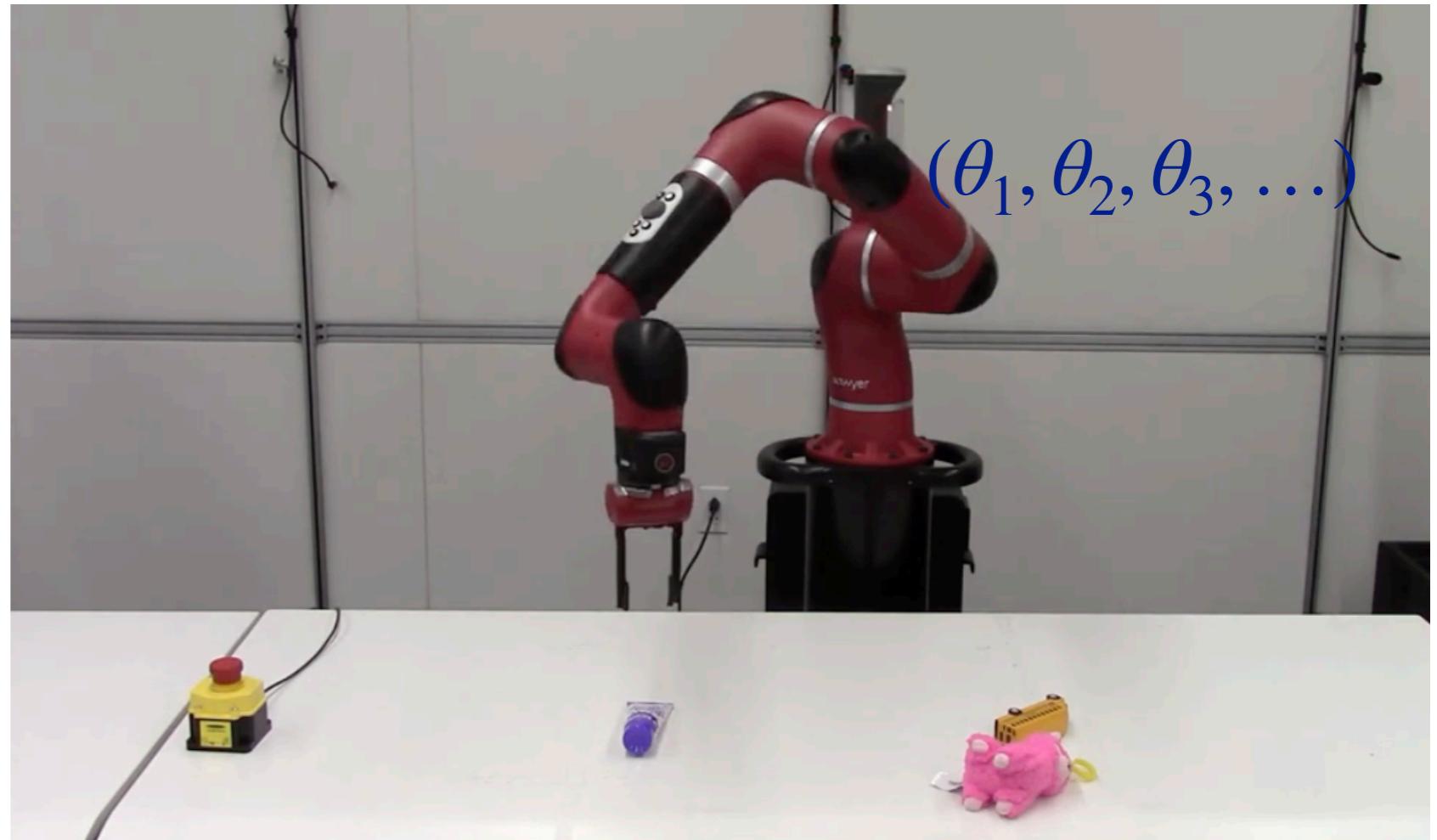
- Link
- Joint
- End Effector
- Base
- Sensors



# Spaces

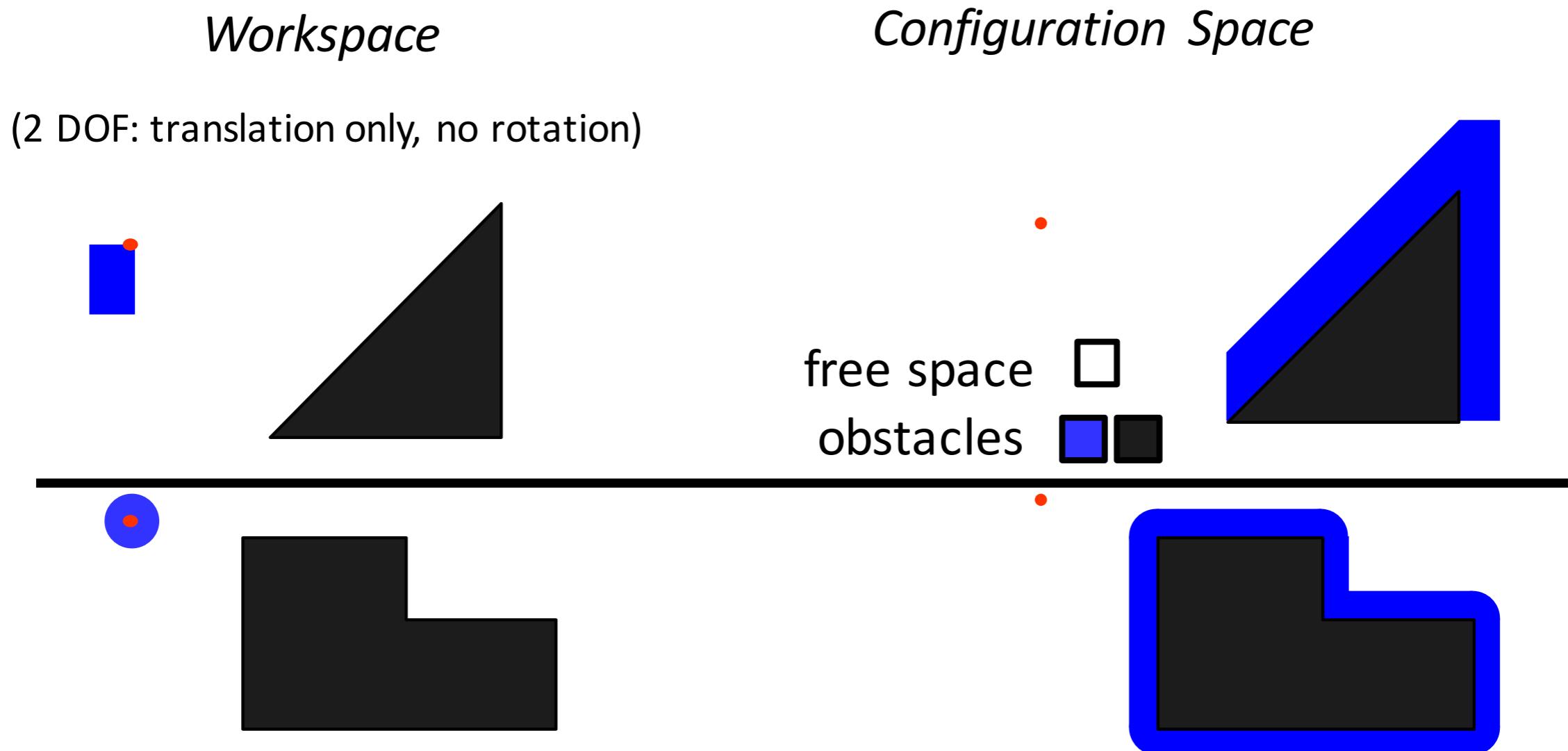
Work Space

Configuration Space



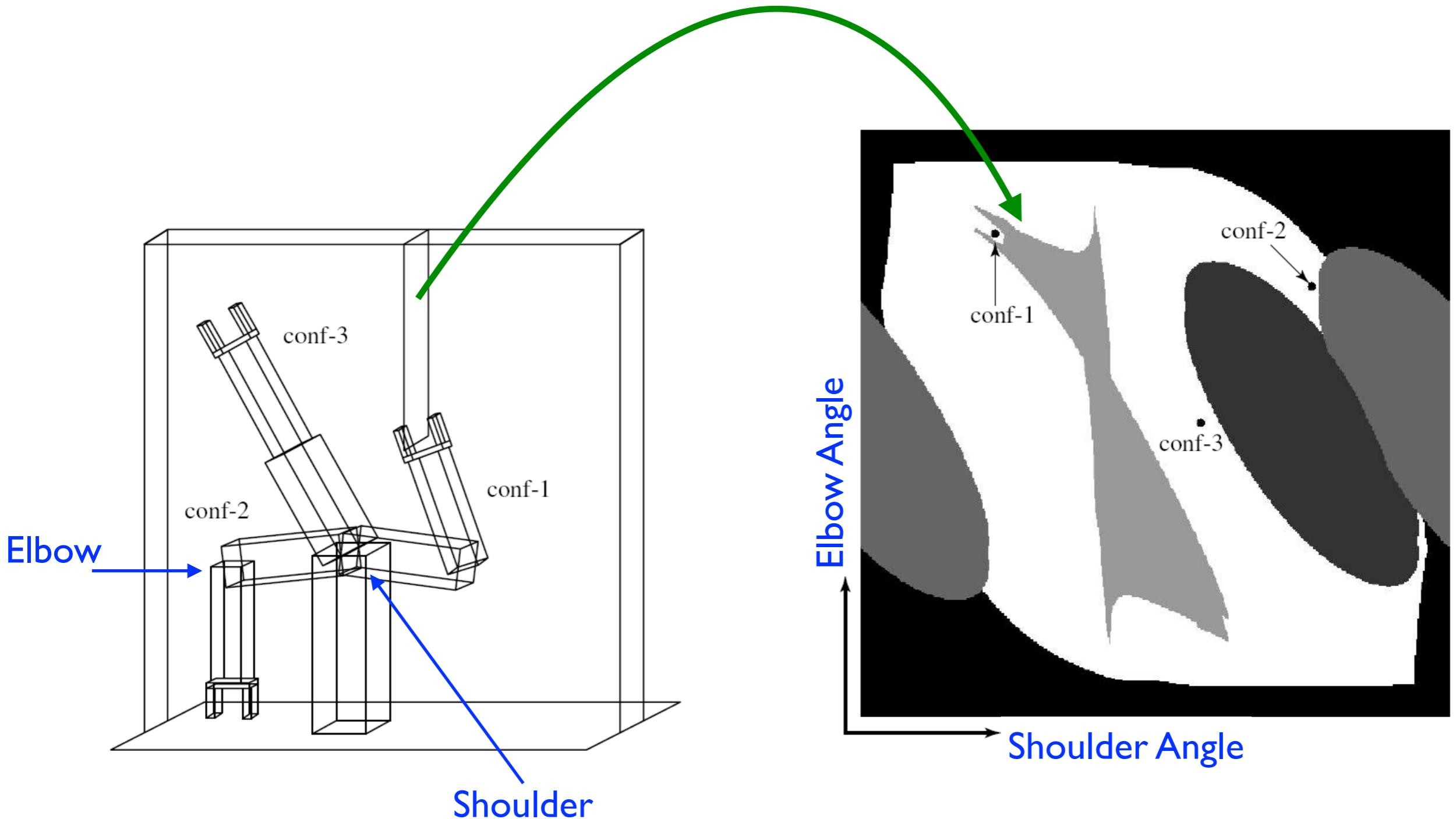
# Configuration Space

obstacles → configuration space obstacles



# Configuration Space

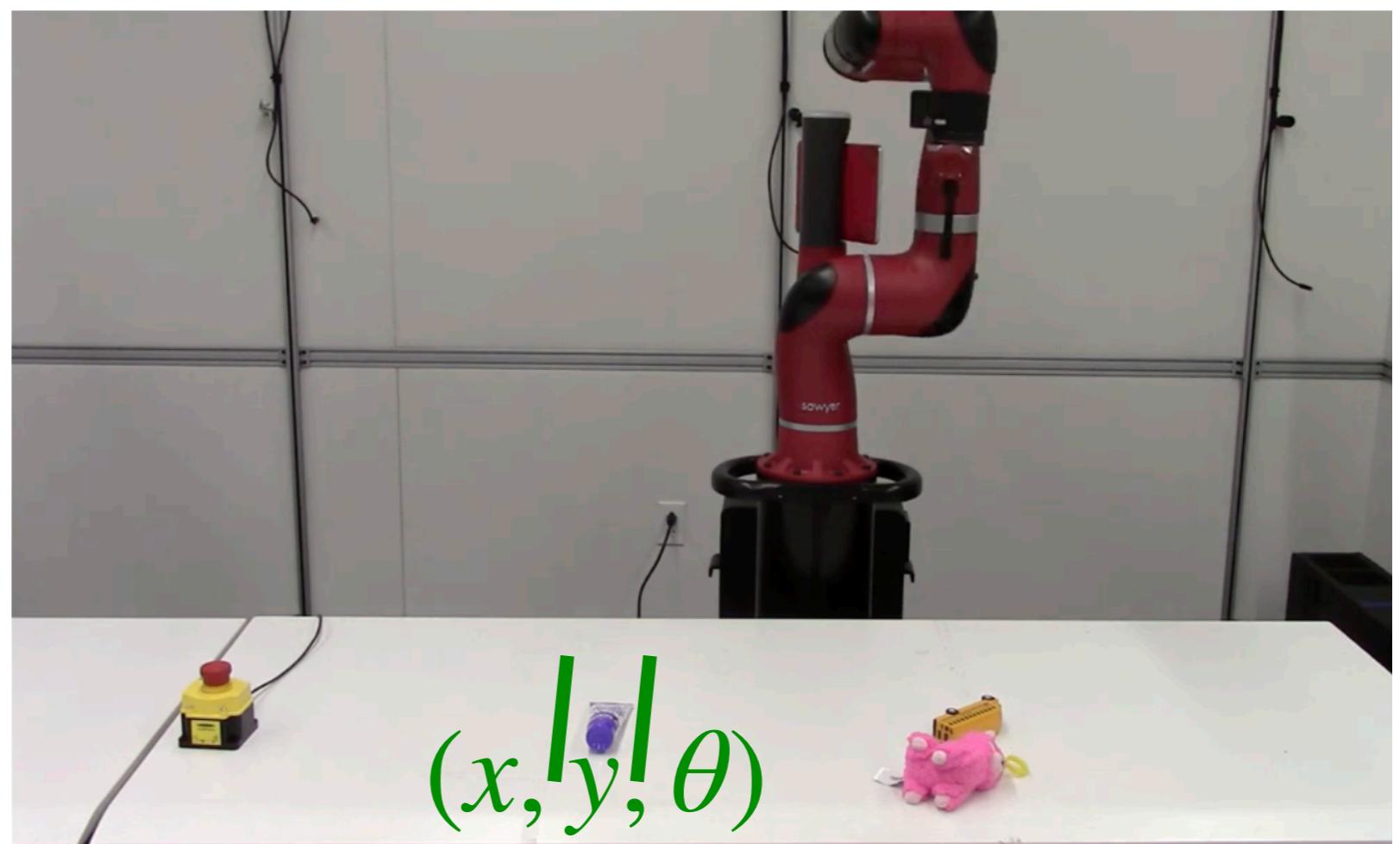
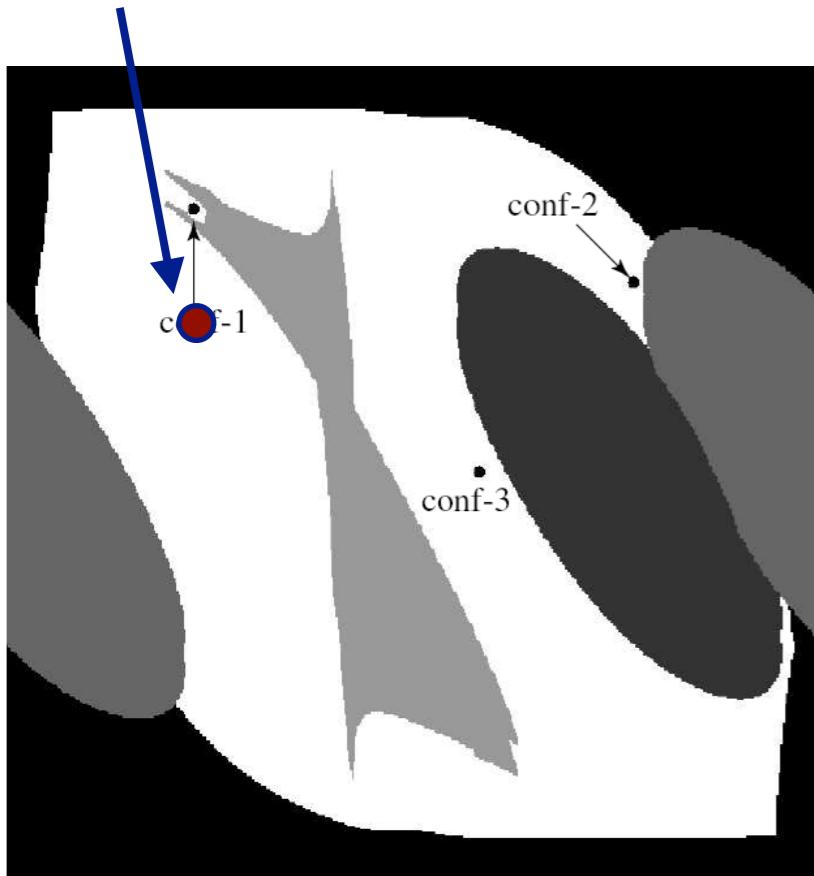
## Another Example



# How to move your robot?

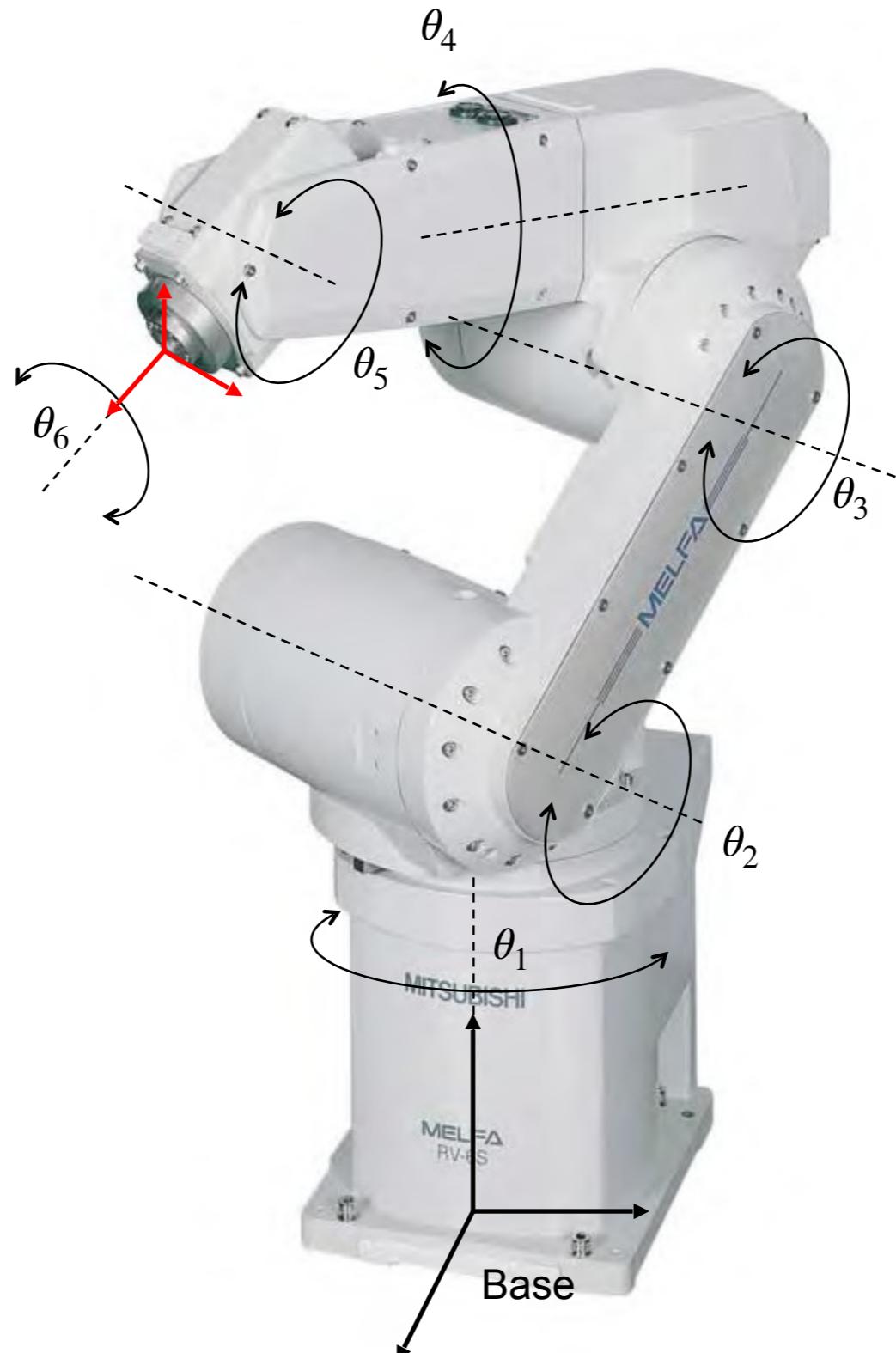
## I. Task space to Configuration space

Initial  
configuration



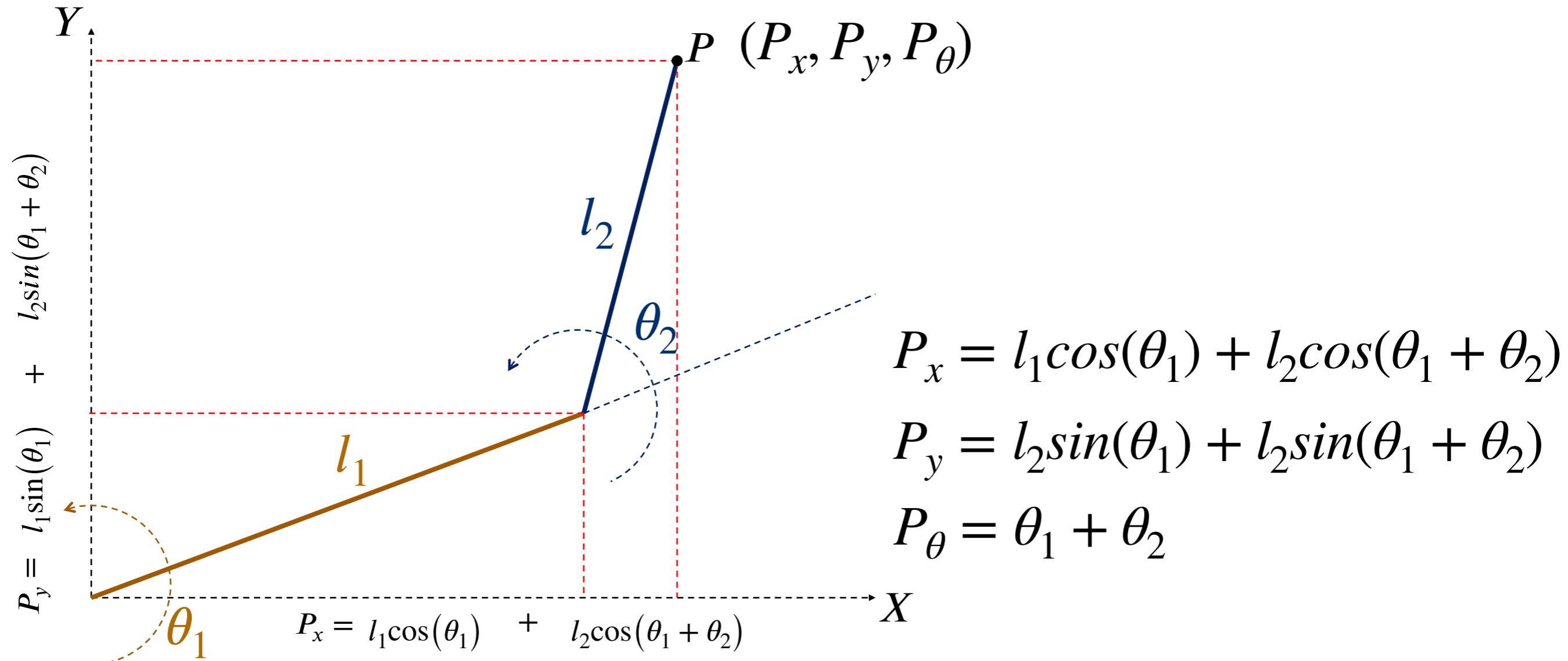
# Configuration Space to Task Space

## Forward Kinematics



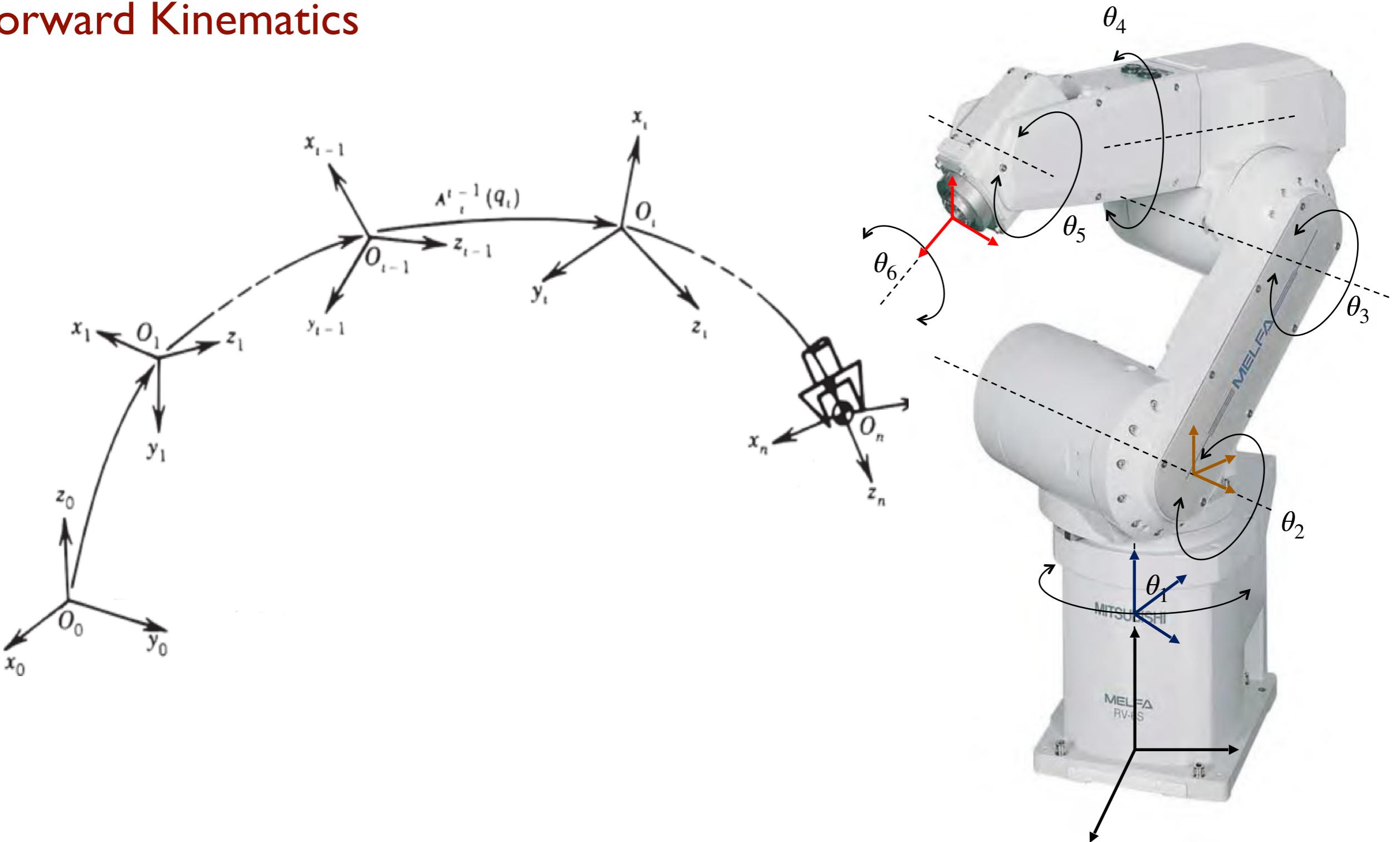
# Configuration Space to Task Space

## Forward Kinematics



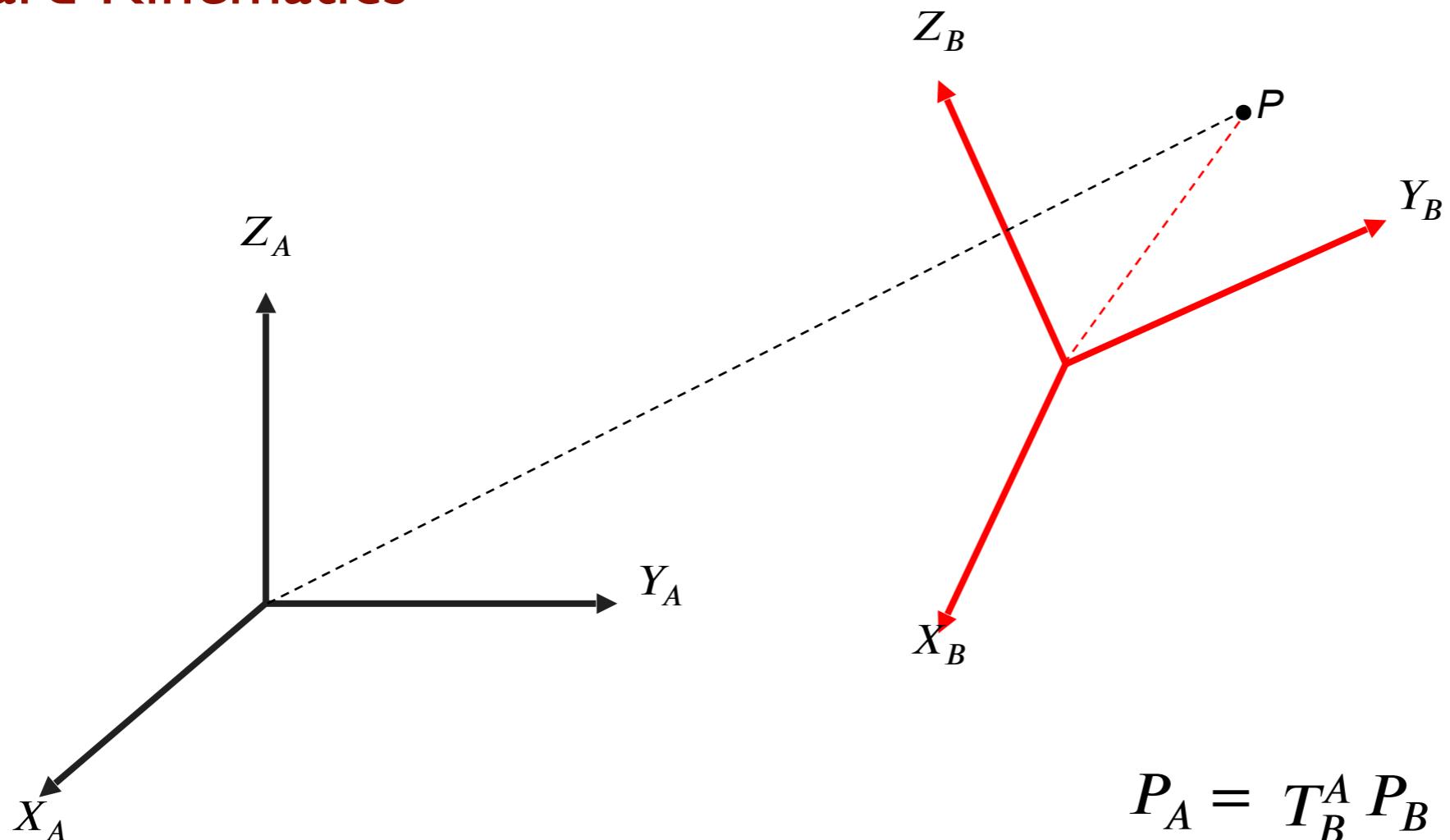
# Configuration Space to Task Space

## Forward Kinematics



# Configuration Space to Task Space

## Forward Kinematics

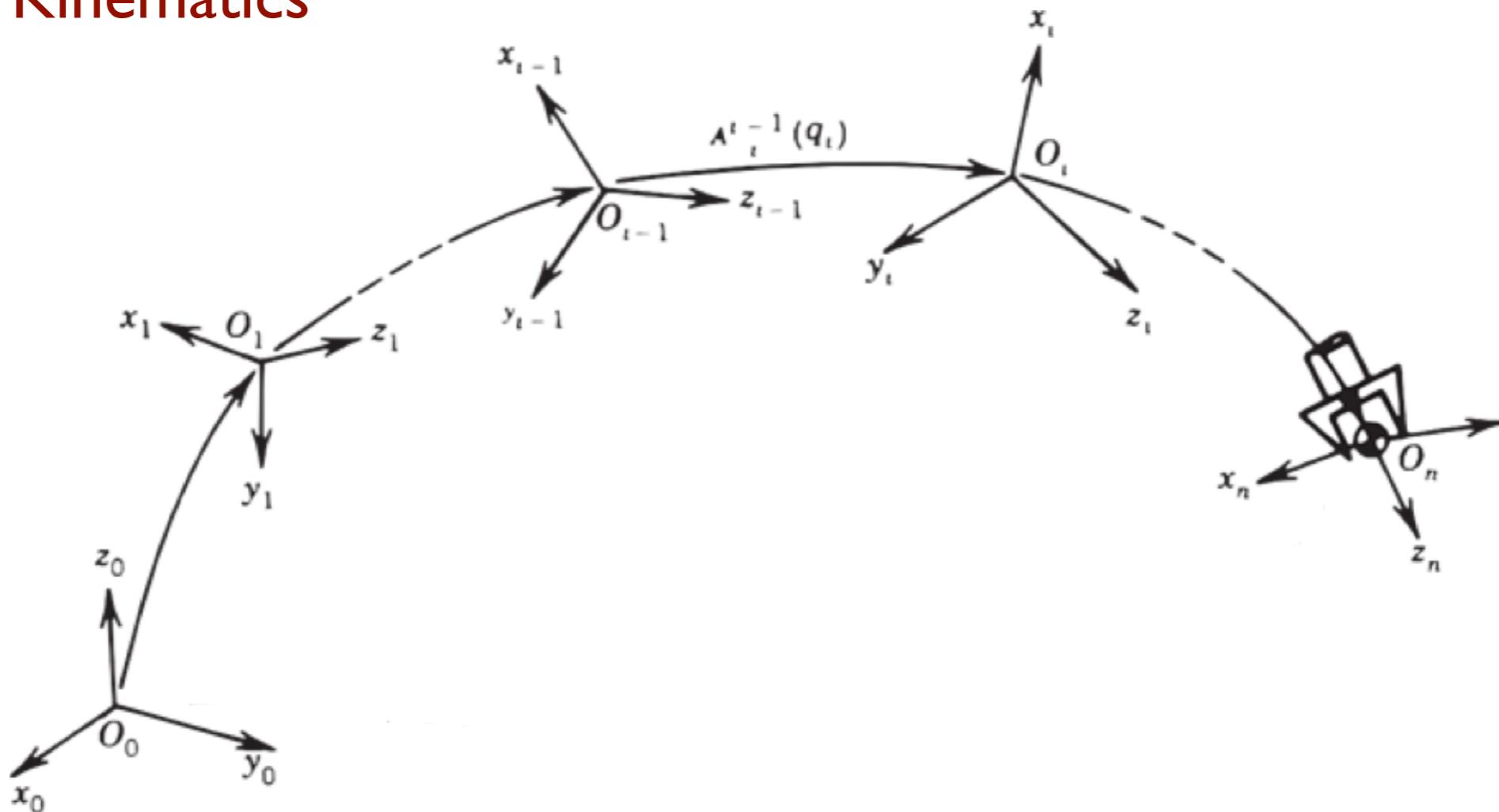


$$P_A = T_B^A P_B$$

$$T_B^A = \begin{bmatrix} \mathbf{r}_{11} & \mathbf{r}_{21} & \mathbf{r}_{31} & \Delta x \\ \mathbf{r}_{12} & \mathbf{r}_{22} & \mathbf{r}_{32} & \Delta y \\ \mathbf{r}_{13} & \mathbf{r}_{23} & \mathbf{r}_{33} & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Configuration Space to Task Space

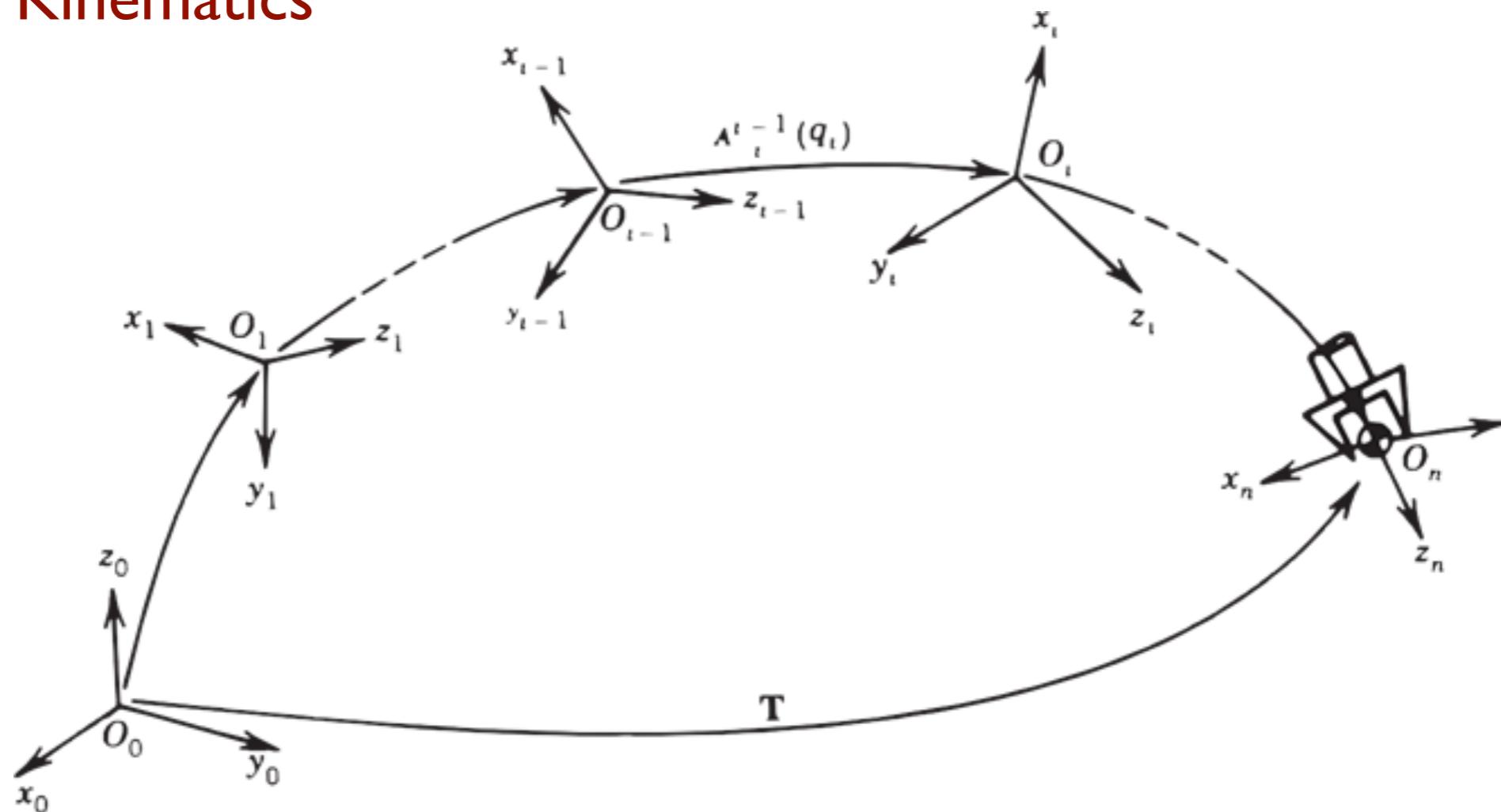
## Forward Kinematics



$$T_1^0(\theta_1) \ T_2^1(\theta_2) \dots \ T_{n-1}^{n-2}(\theta_n) \ T_n^{n-1}(\theta_n)$$

# Configuration Space to Task Space

## Forward Kinematics



Maps configuration  
space to work space

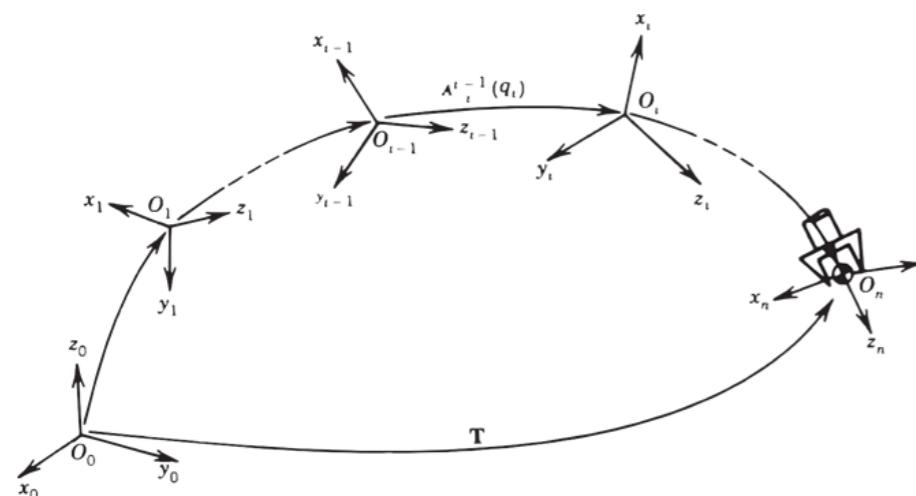
$$T = T_1^0(\theta_1) T_2^1(\theta_2) \dots T_{n-1}^{n-2}(\theta_{n-1}) T_n^{n-1}(\theta_n)$$
$$= \begin{bmatrix} r_{11} & r_{21} & r_{31} & \Delta x \\ r_{12} & r_{22} & r_{32} & \Delta y \\ r_{13} & r_{23} & r_{33} & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$x = f(\theta) = f(\theta_1, \theta_2, \dots, \theta_{n-1}, \theta_n)$$

# Task Space to Configuration Space

Forward Kinematics

Inverse Kinematics

Numerical IK



$$x = f(\theta)$$

Solve for  $\theta_d$  in:

$$x_d - f(\theta_d) = 0$$

Analytical IK

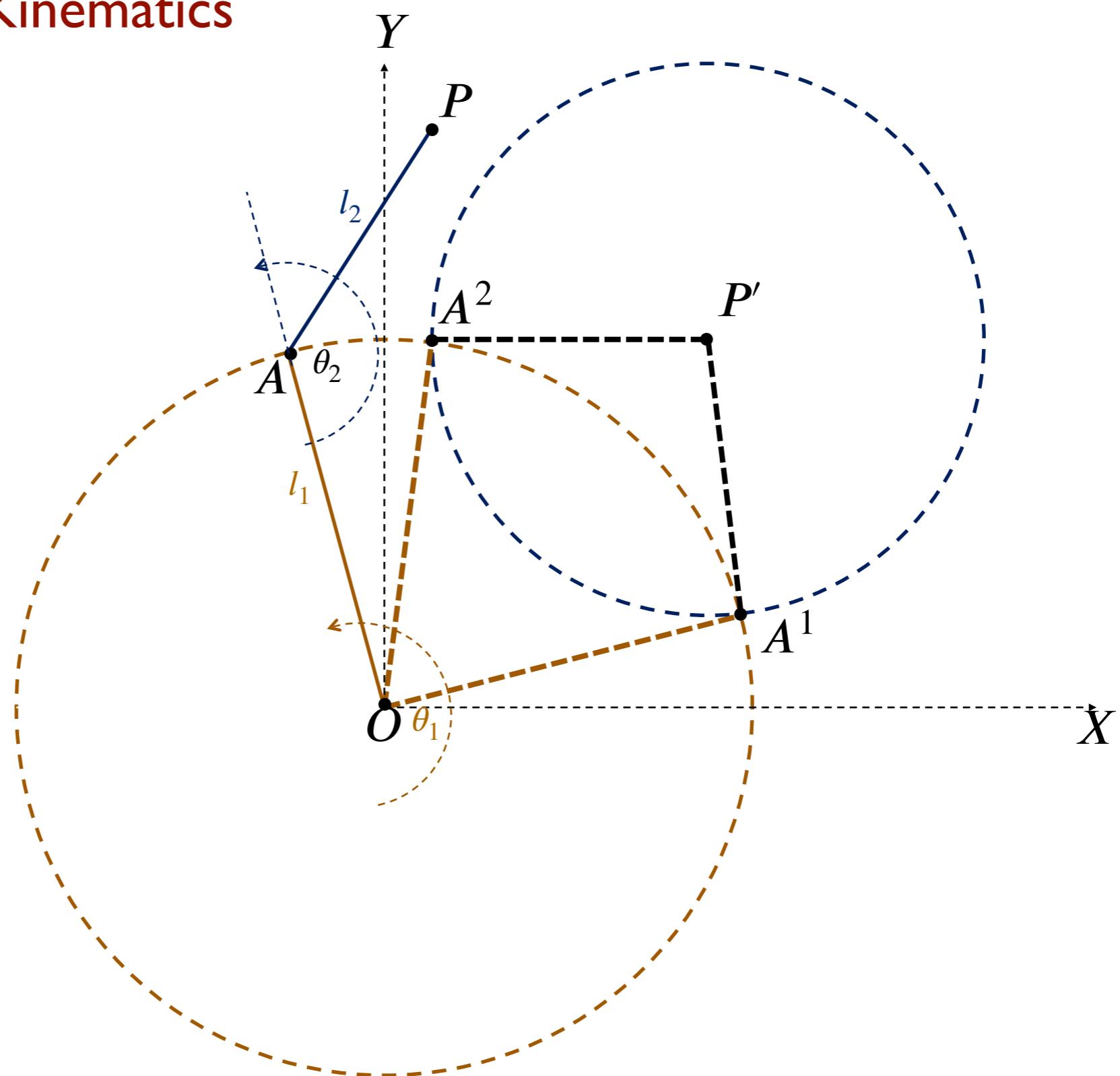
- Robot Specific
- Fast
- Characterize the solution space

Maps configuration space to work space

Find configuration(s) that map to a given work space point

# Task Space to Configuration Space

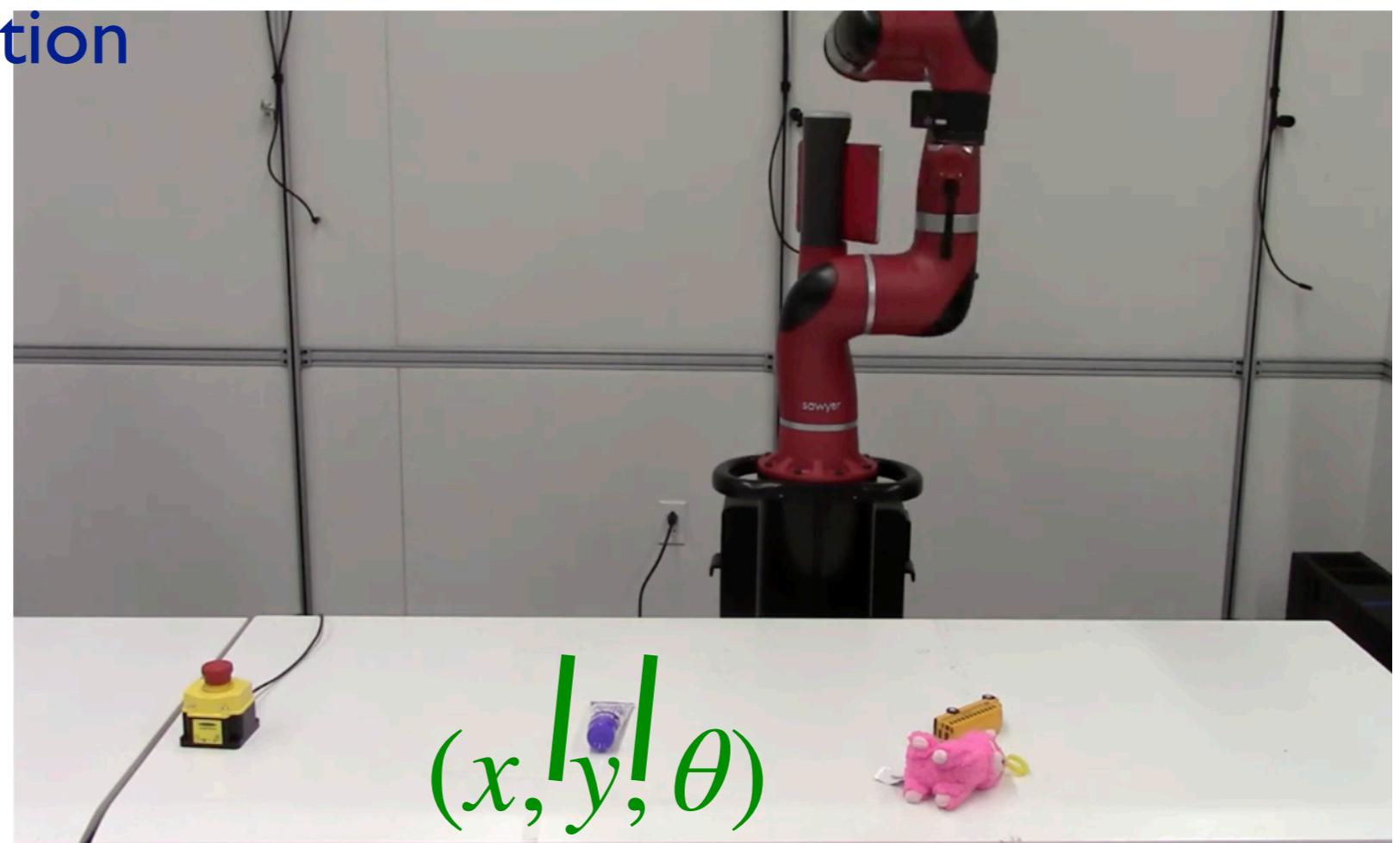
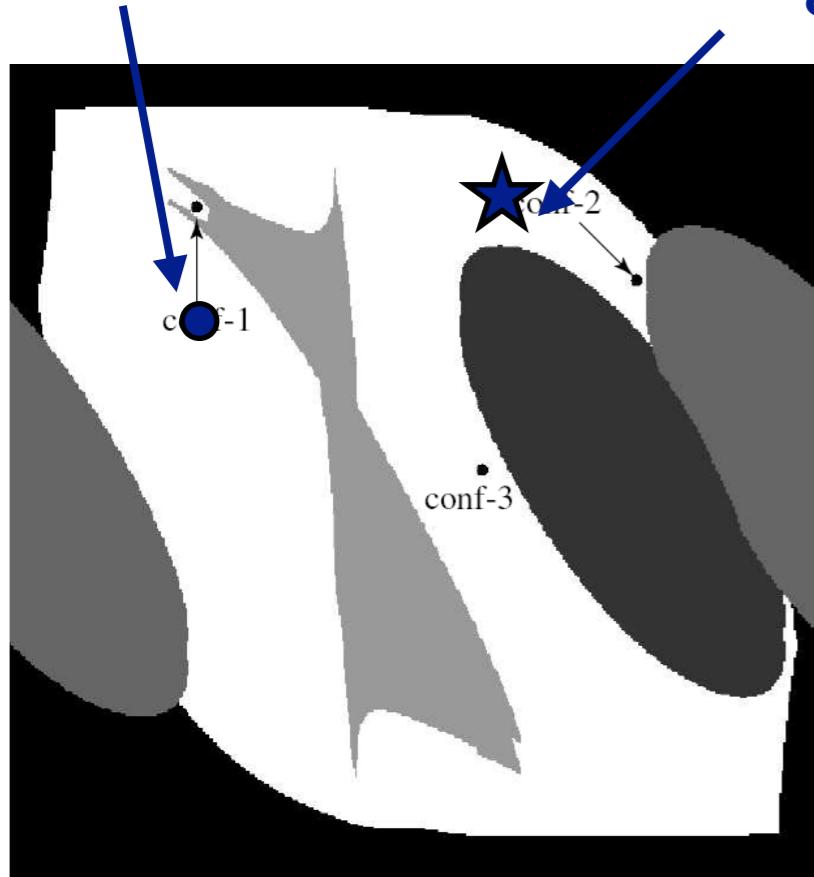
## Analytical Inverse Kinematics



# How to move your robot?

## I. Task space to Configuration space

Initial configuration      Desired configuration



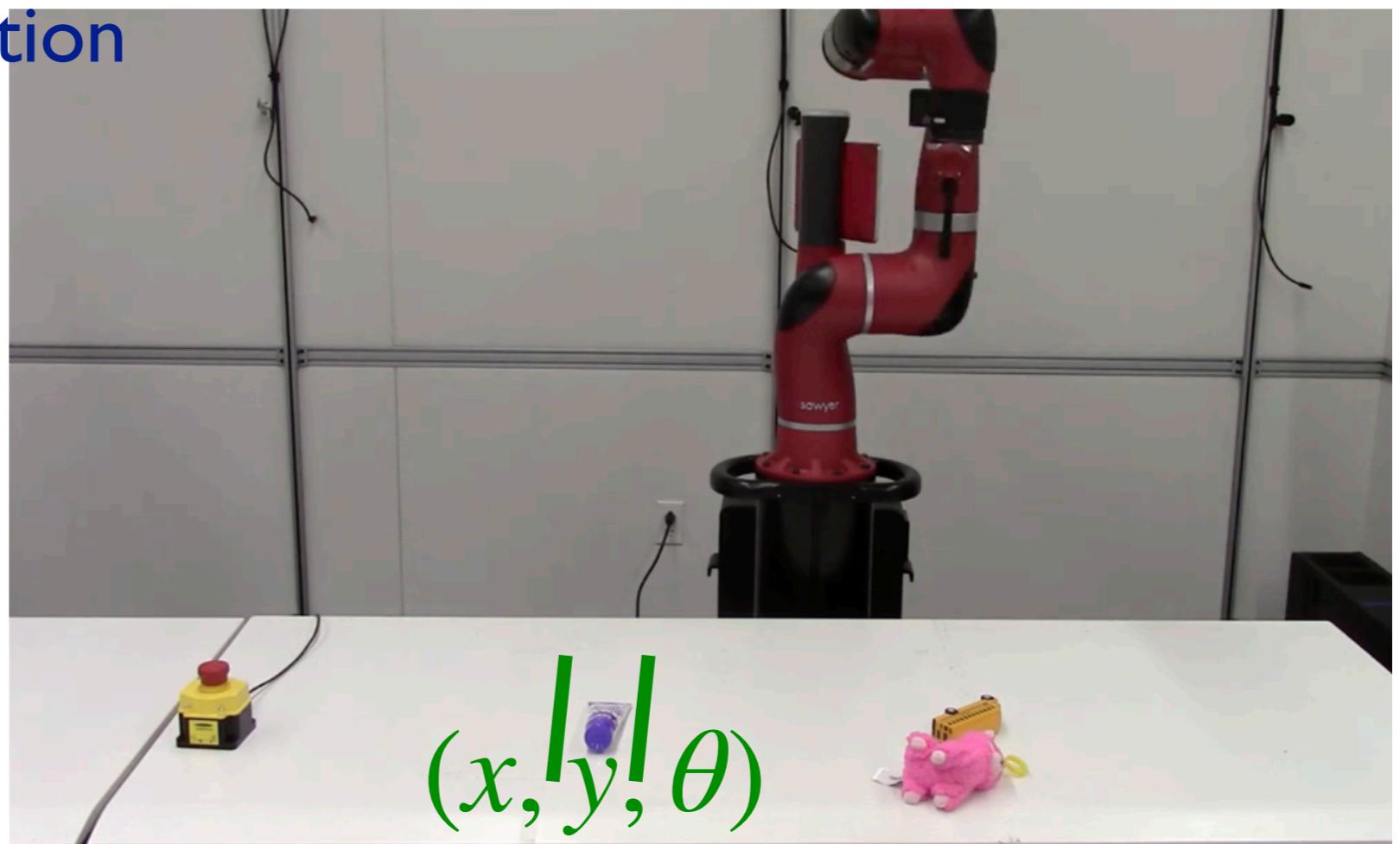
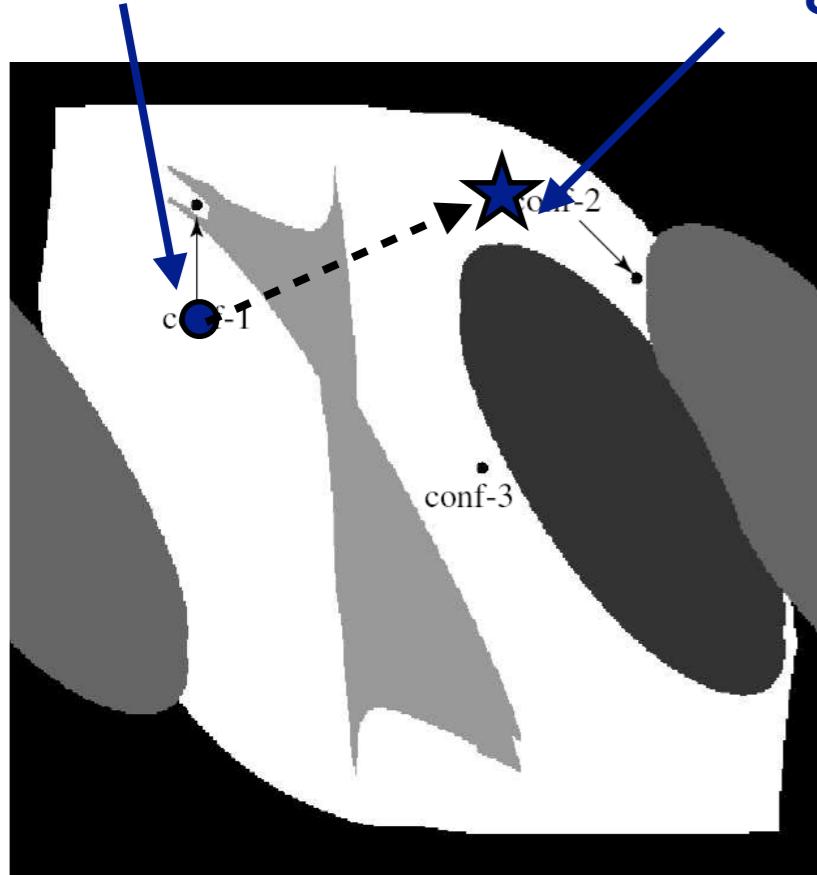
$(x, y, \theta)$

# How to move your robot?

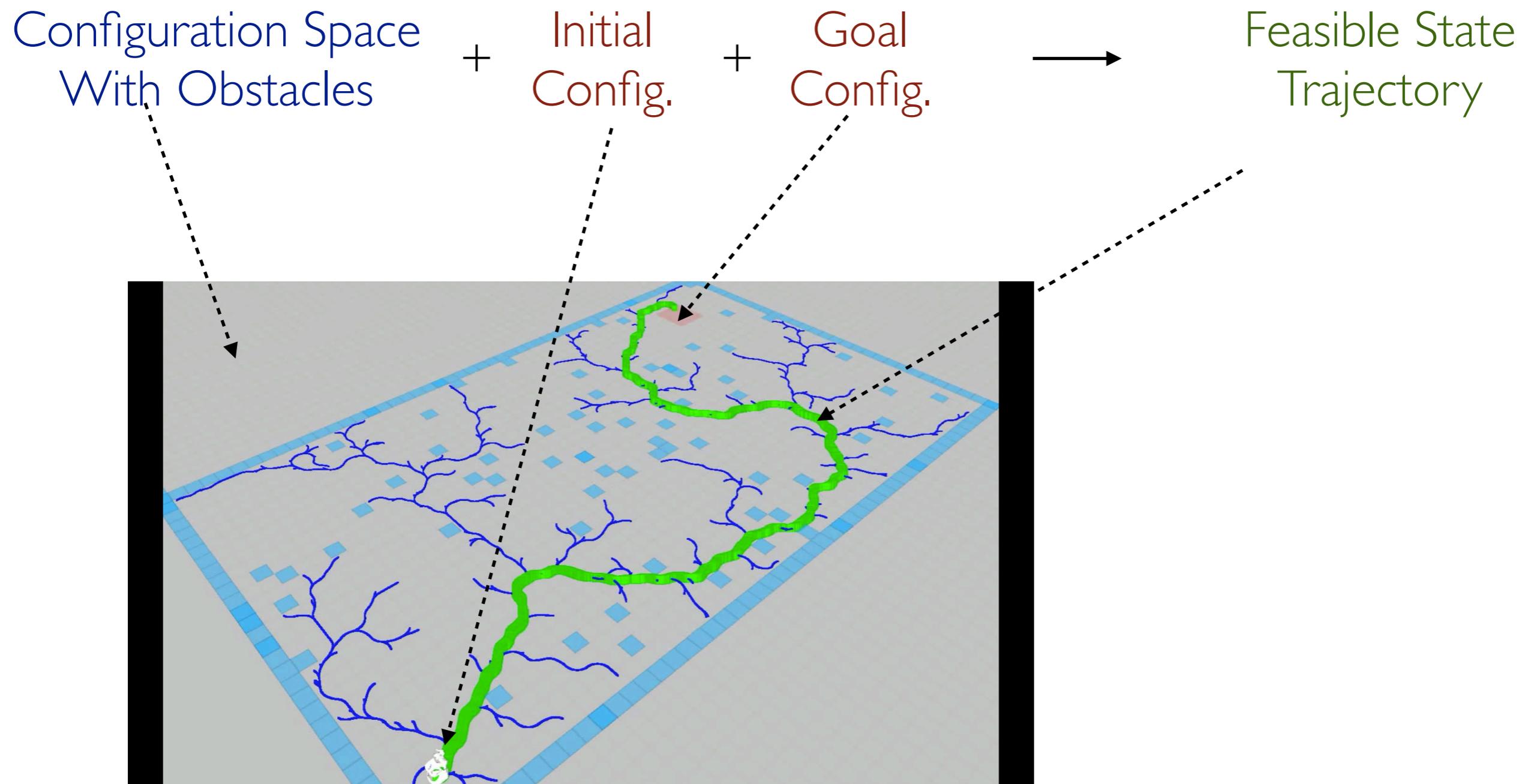
## I. Task space to Configuration space

## 2. Configuration space trajectory

Initial configuration      Desired configuration



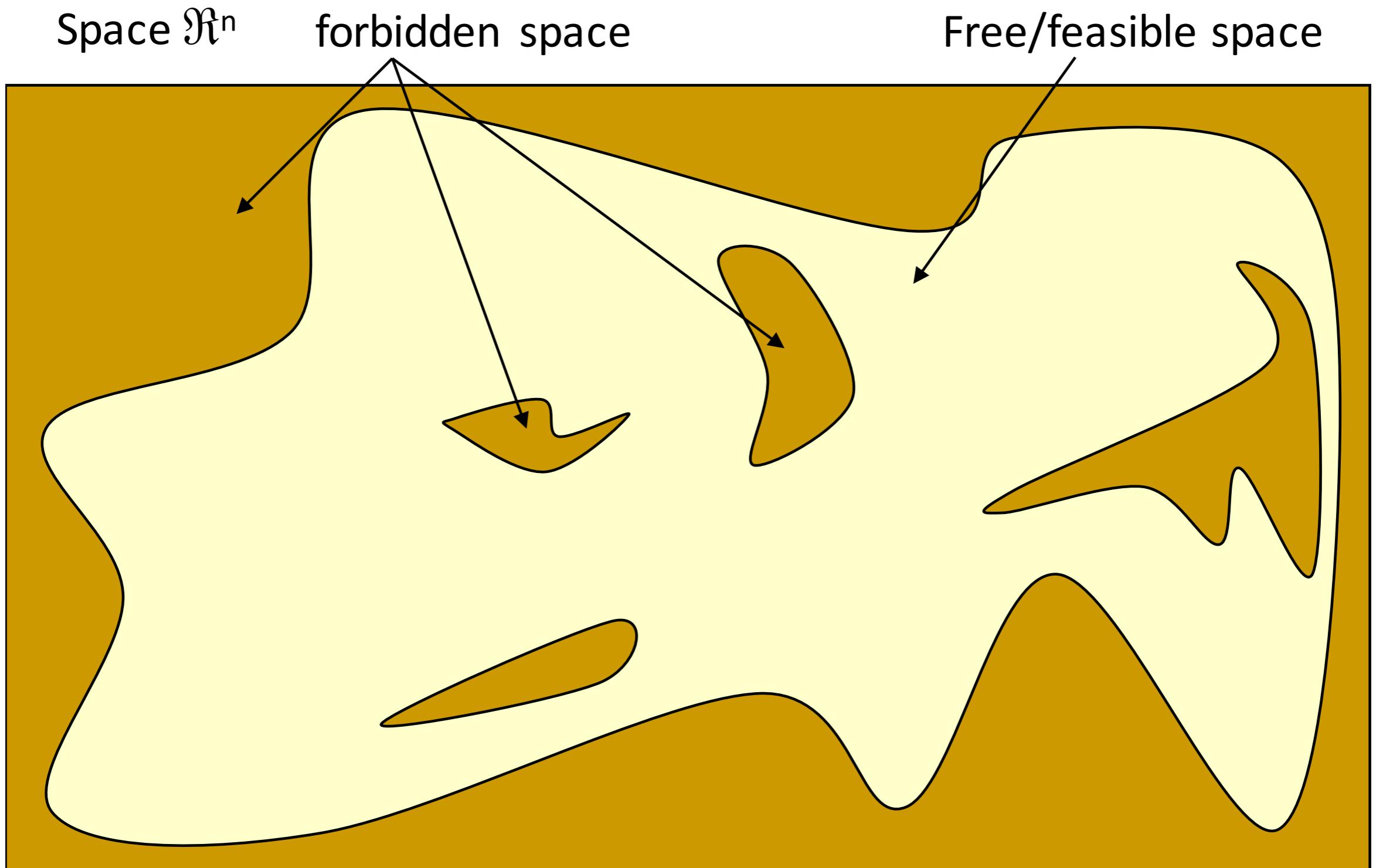
# Path Planning



# Path Planning

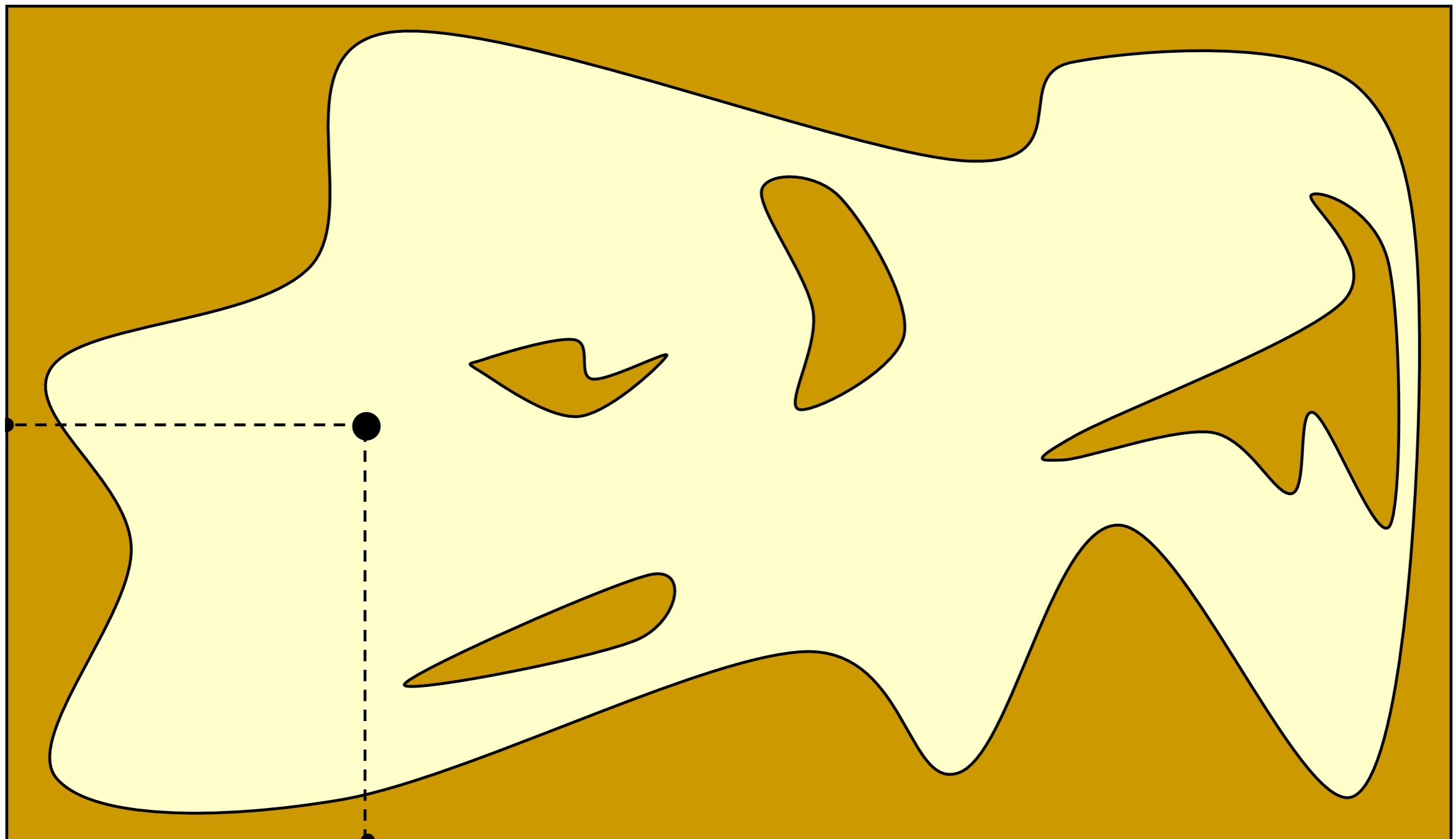
- 1. Complete Methods
- 2. Grid Methods
- 3. Sampling Methods
- 4. Potential Fields
- 5. Trajectory Optimization

# Probabilistic Roadmaps



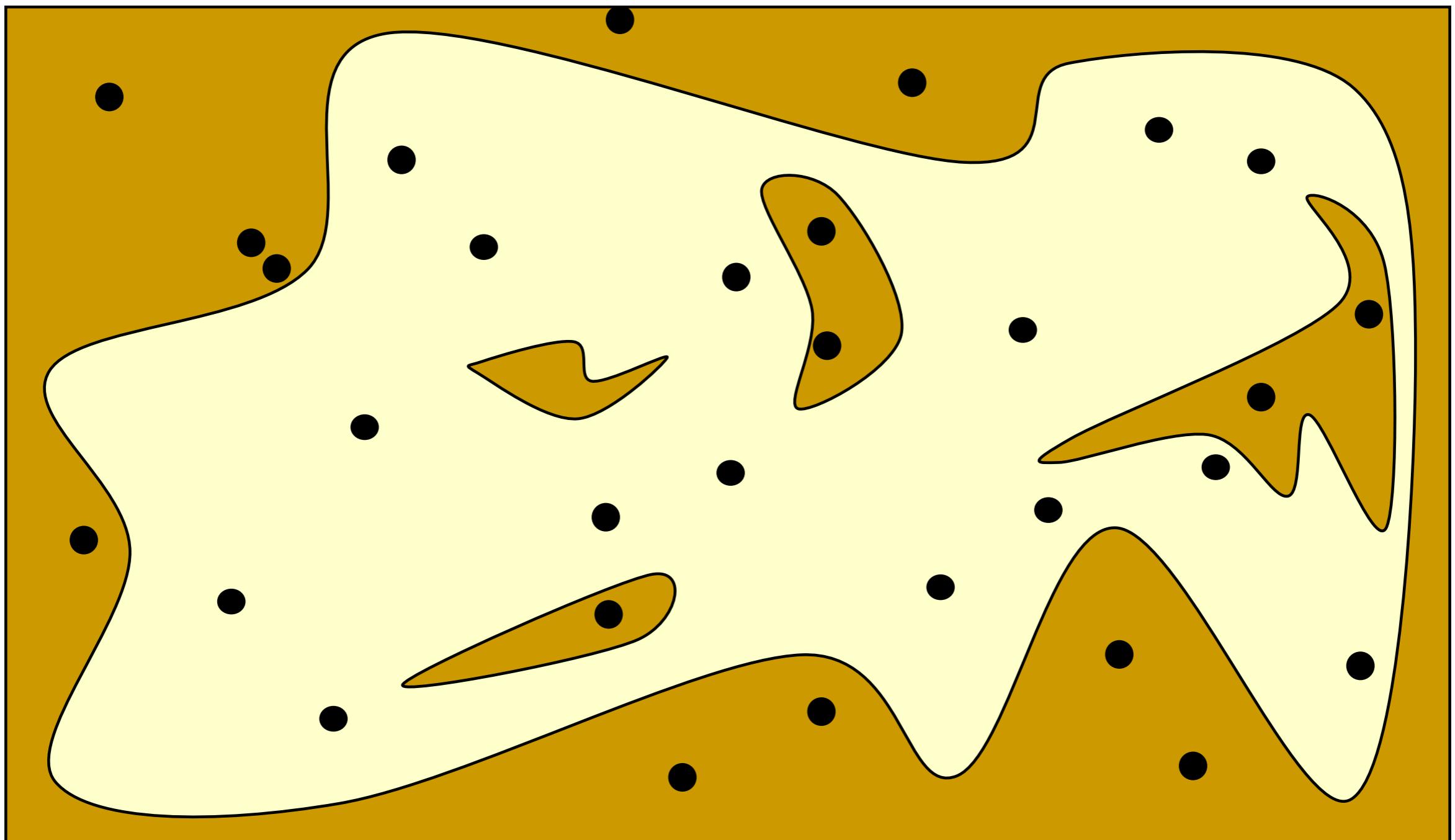
# Probabilistic Roadmaps

Randomly Sample Configurations



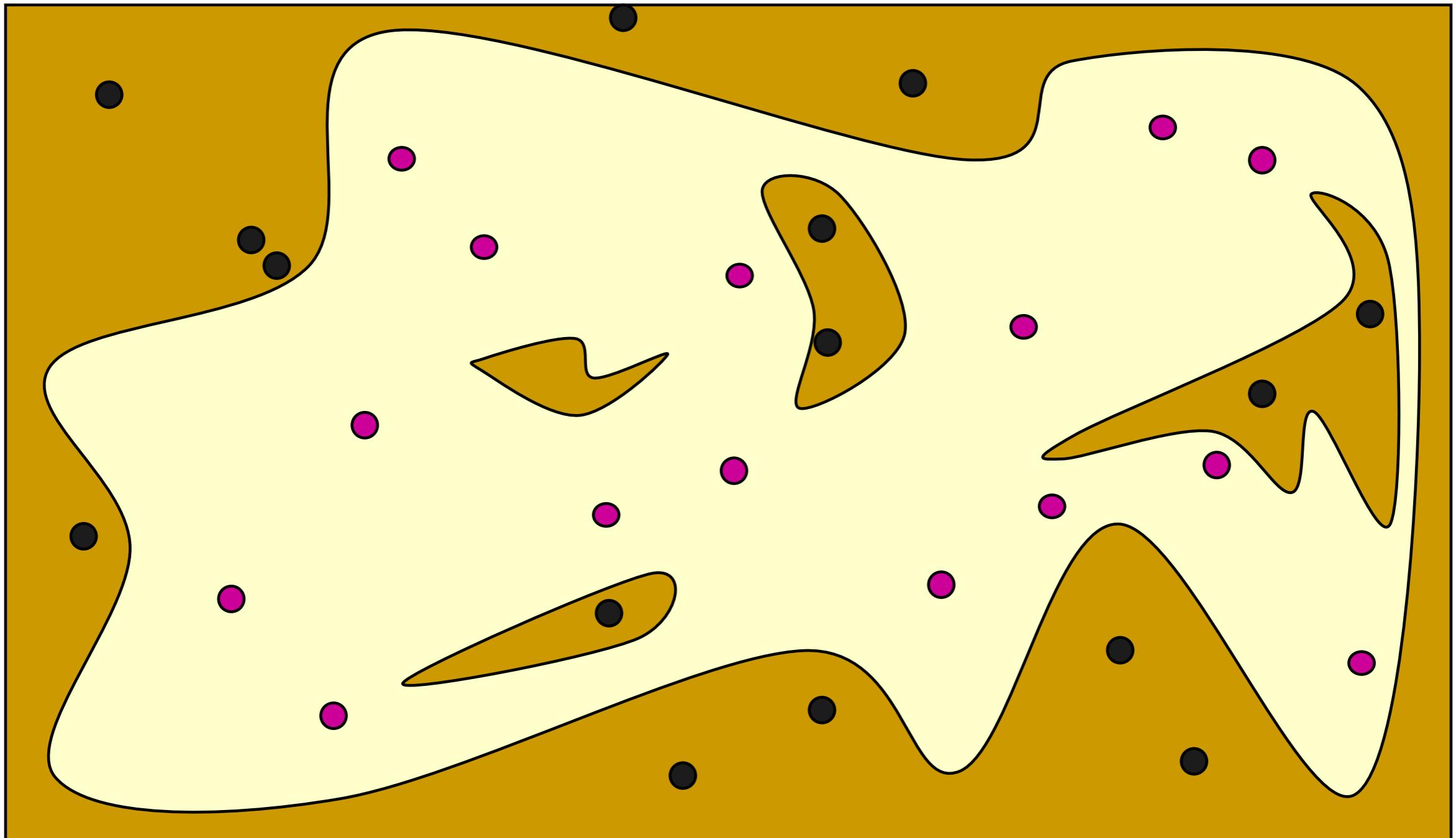
# Probabilistic Roadmaps

Randomly Sample Configurations



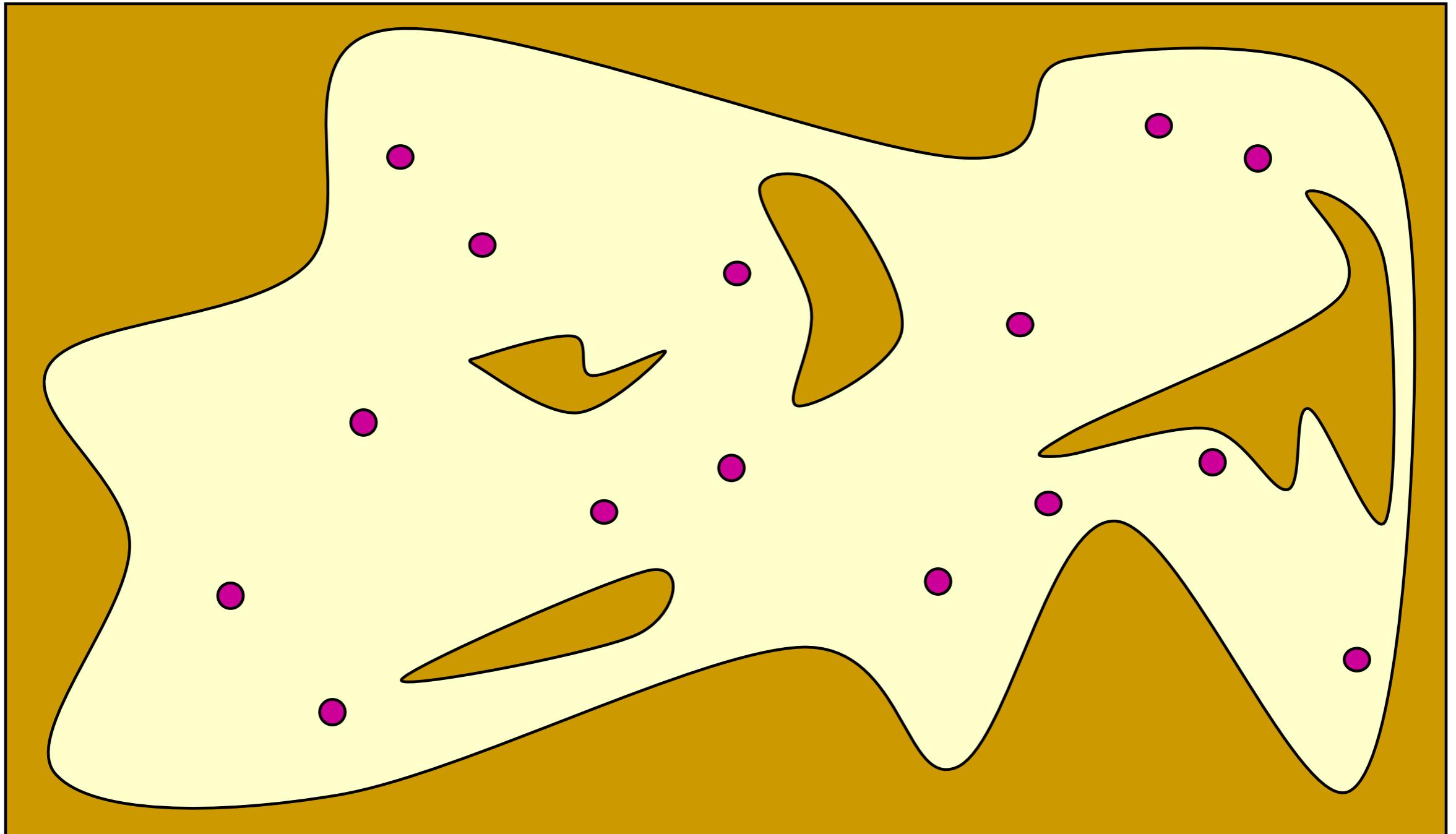
# Probabilistic Roadmaps

Test Sampled Configurations for Collisions



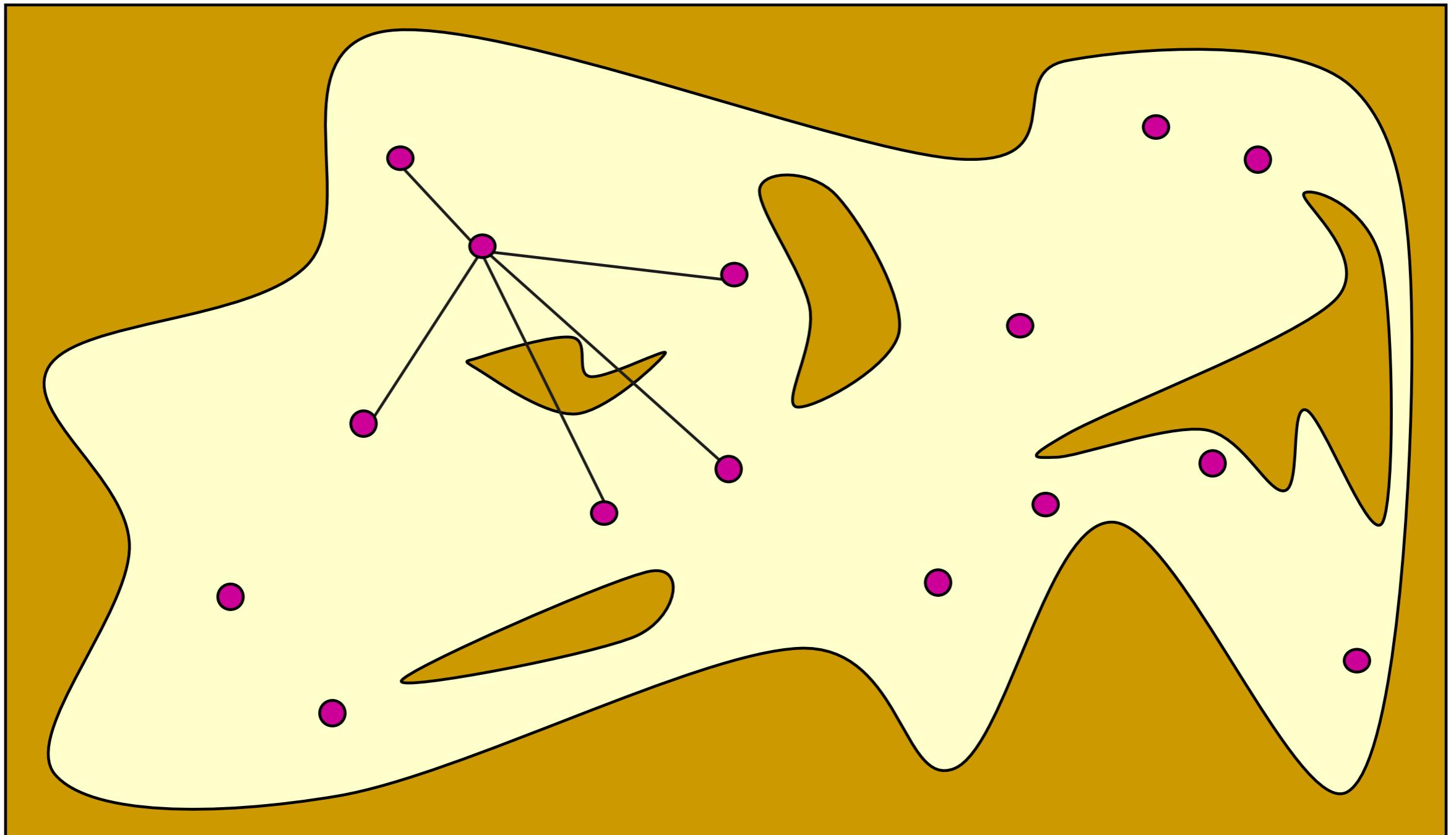
# Probabilistic Roadmaps

The collision-free configurations are retained as milestones



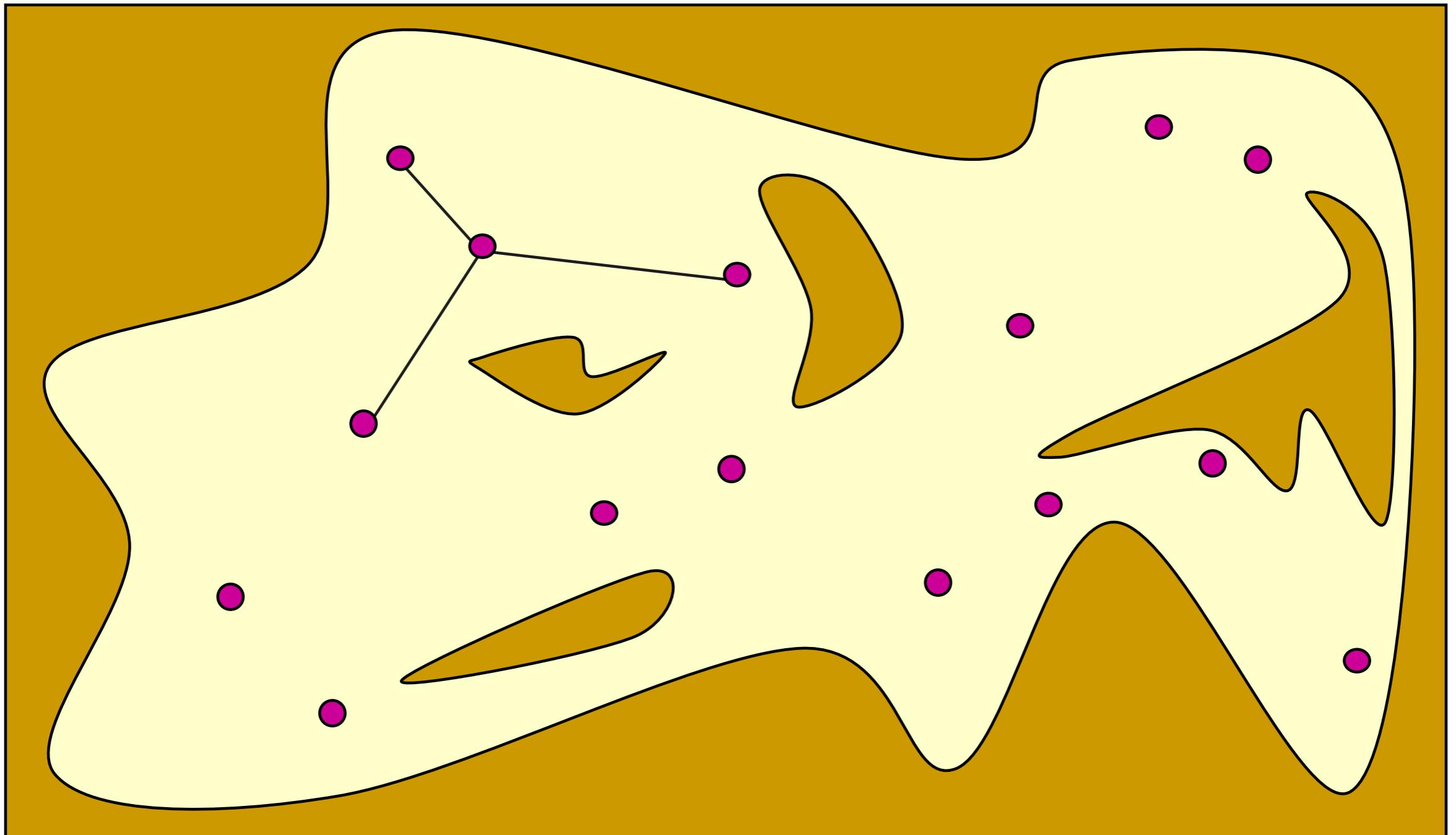
# Probabilistic Roadmaps

Each milestone is linked by straight paths to its nearest neighbors



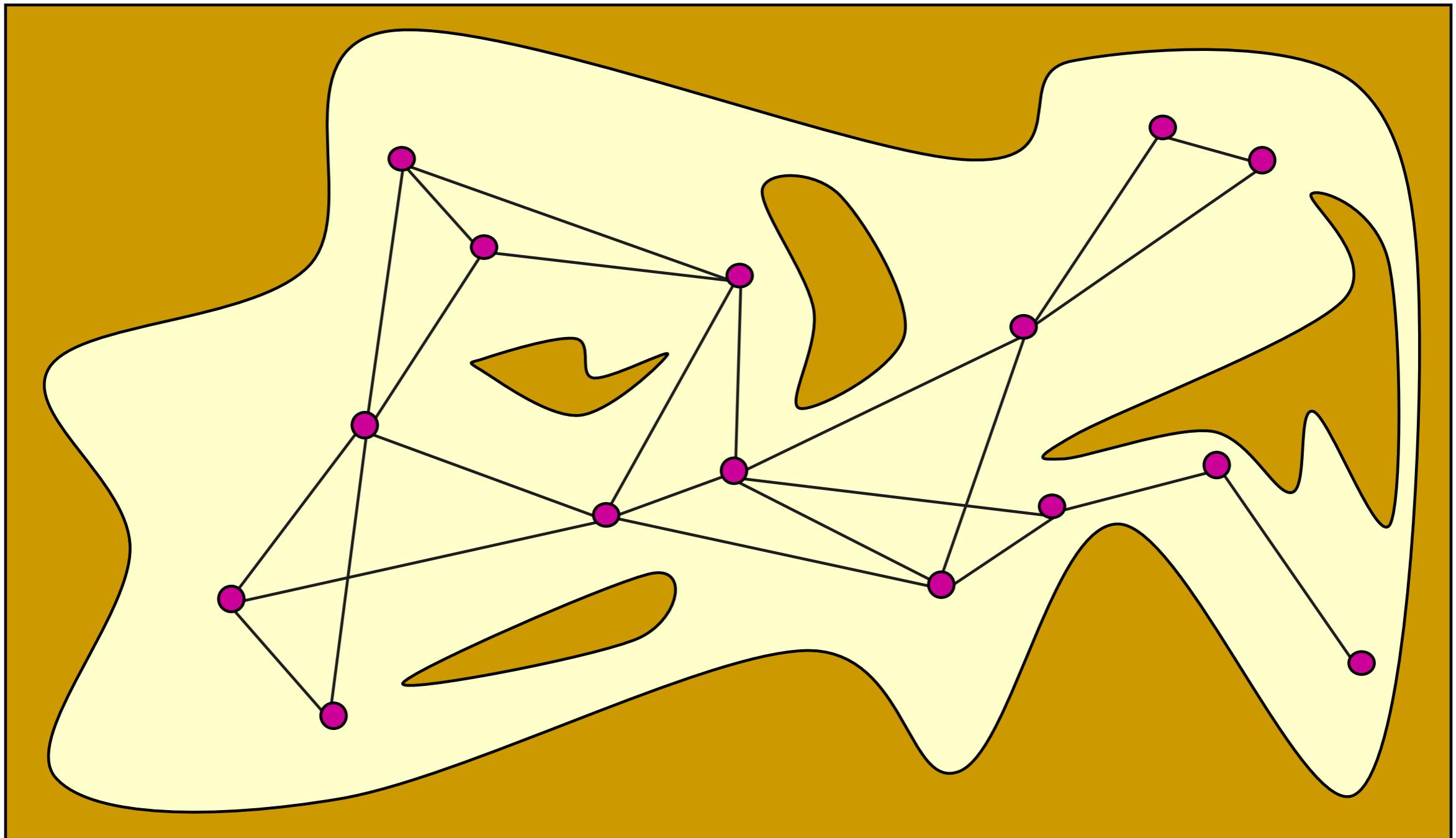
# Probabilistic Roadmaps

Paths that undergo collisions are removed



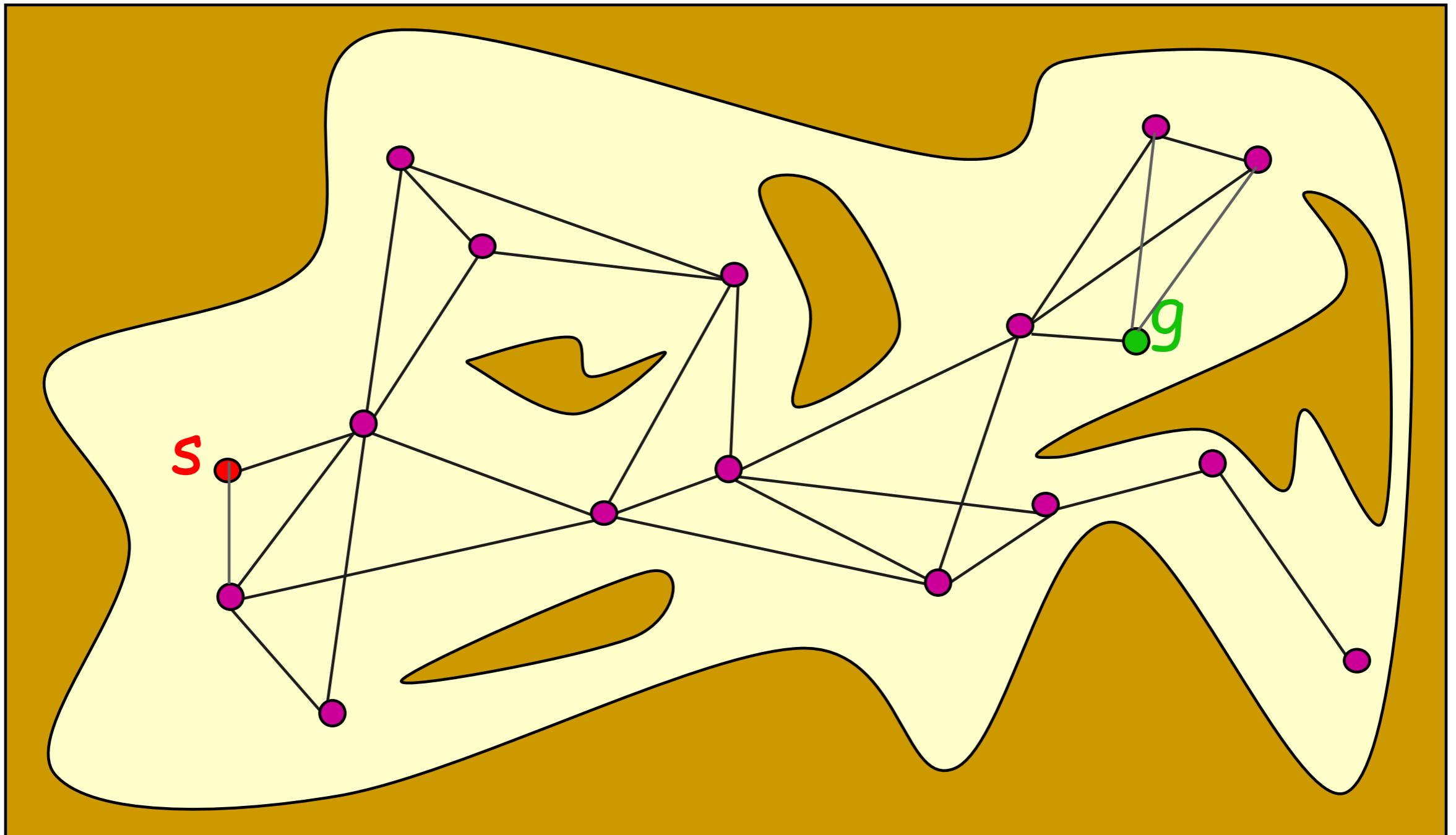
# Probabilistic Roadmaps

The collision-free links are retained as local paths to form the PRM



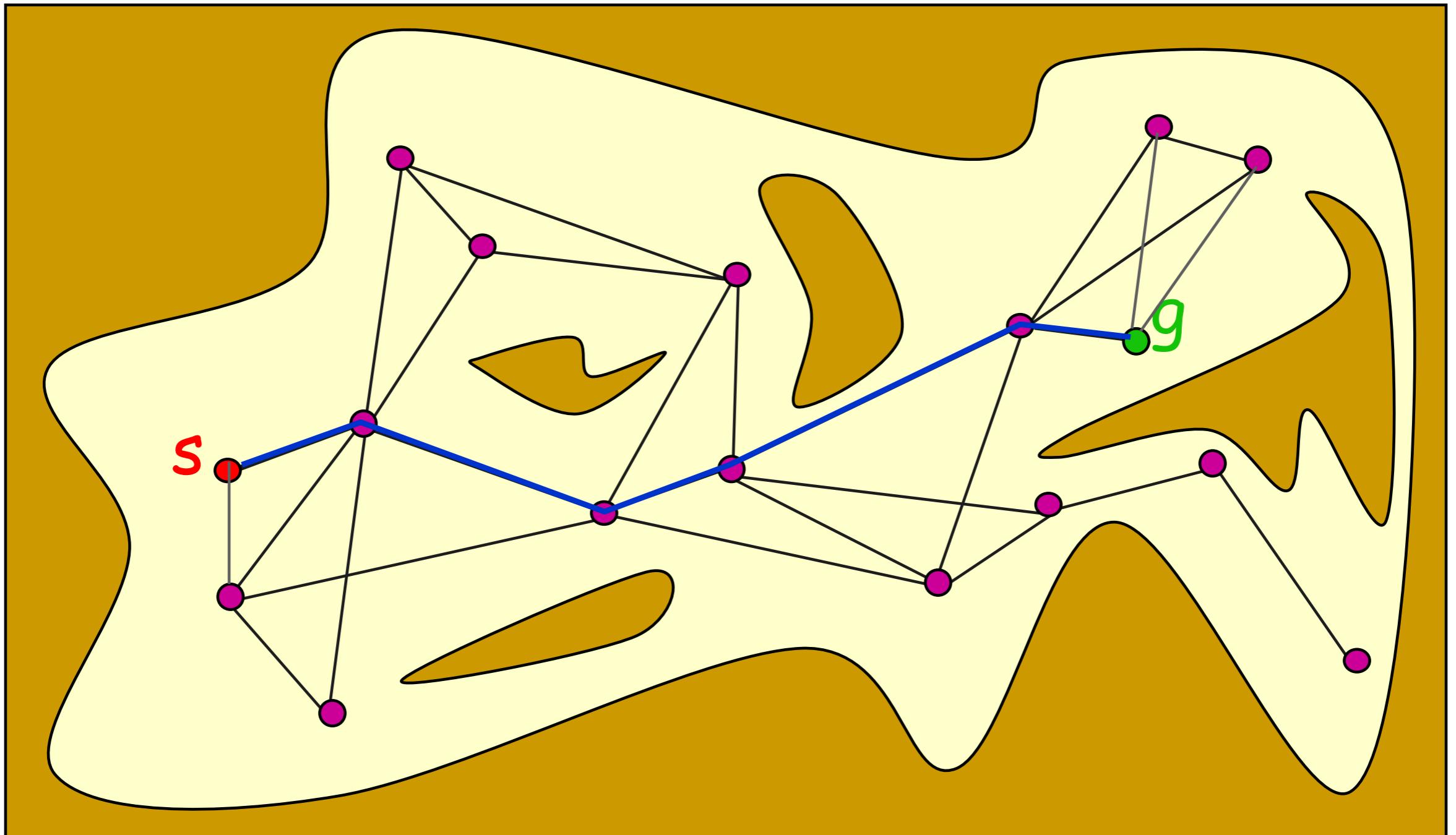
# Probabilistic Roadmaps

The start and goal configurations are included as milestones



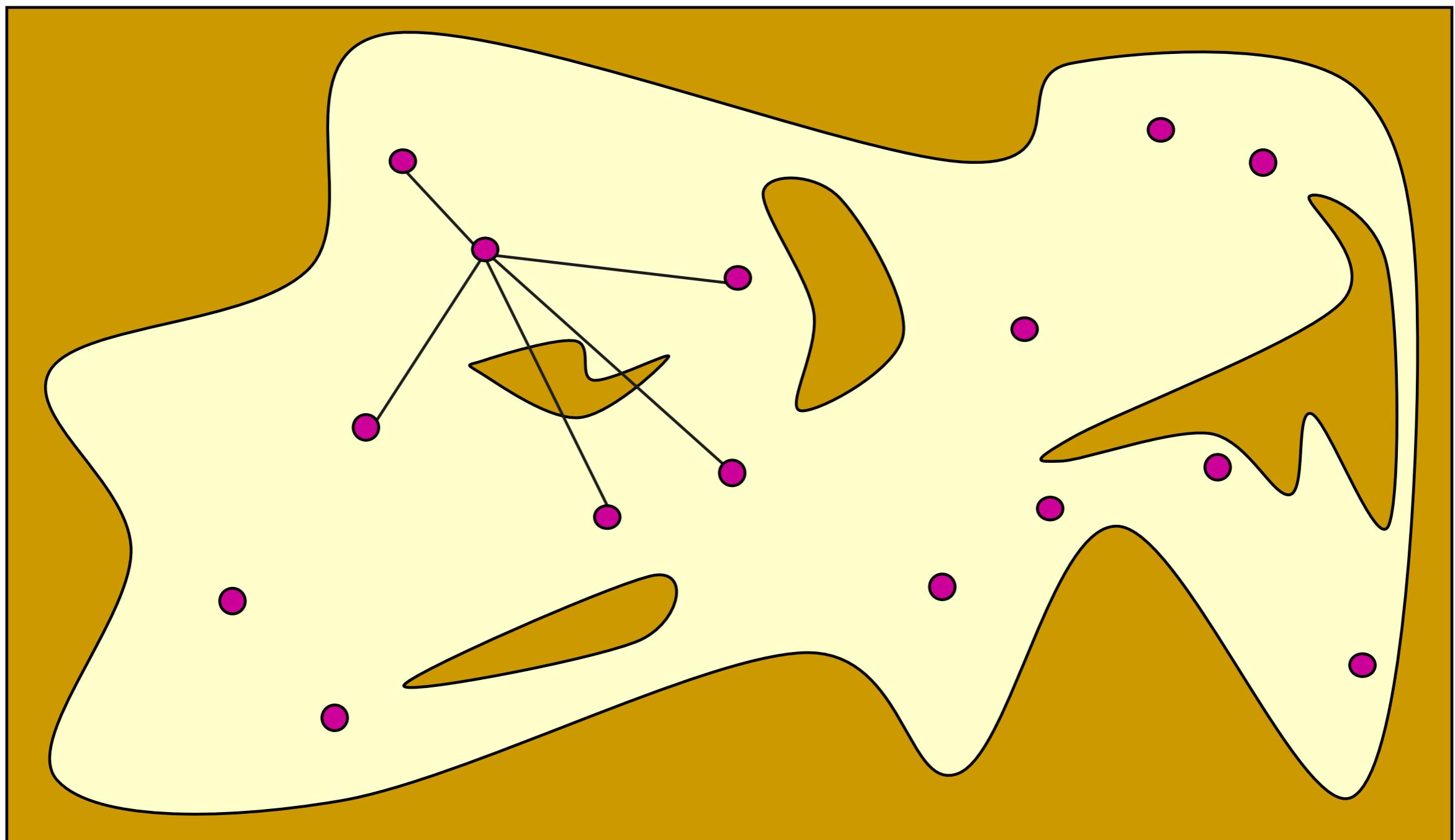
# Probabilistic Roadmaps

The PRM is searched for a path from  $s$  to  $g$



# Probabilistic Roadmaps

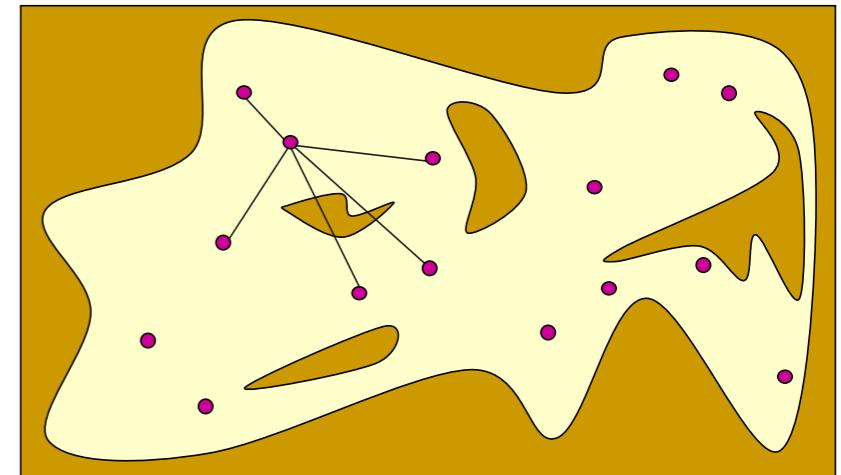
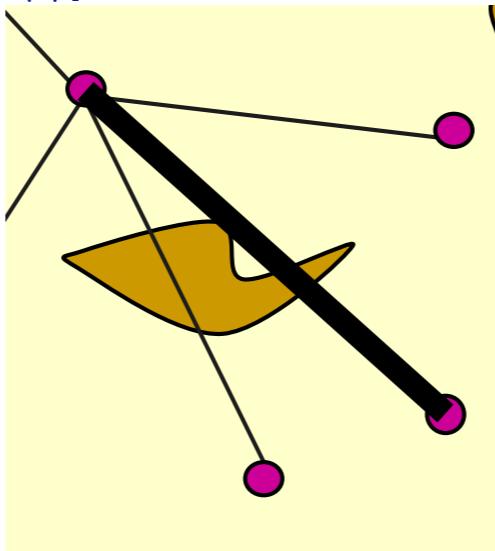
Challenging to link milestones.



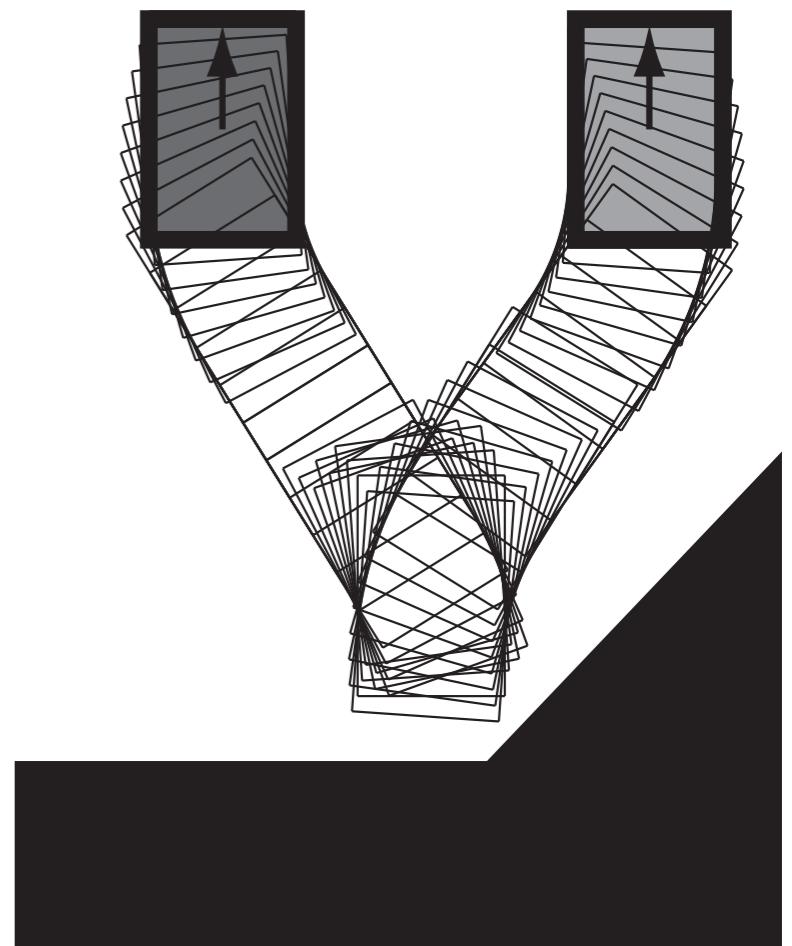
# Probabilistic Roadmaps

Challenging to link milestones.

Collision checking can be slow.



start      goal

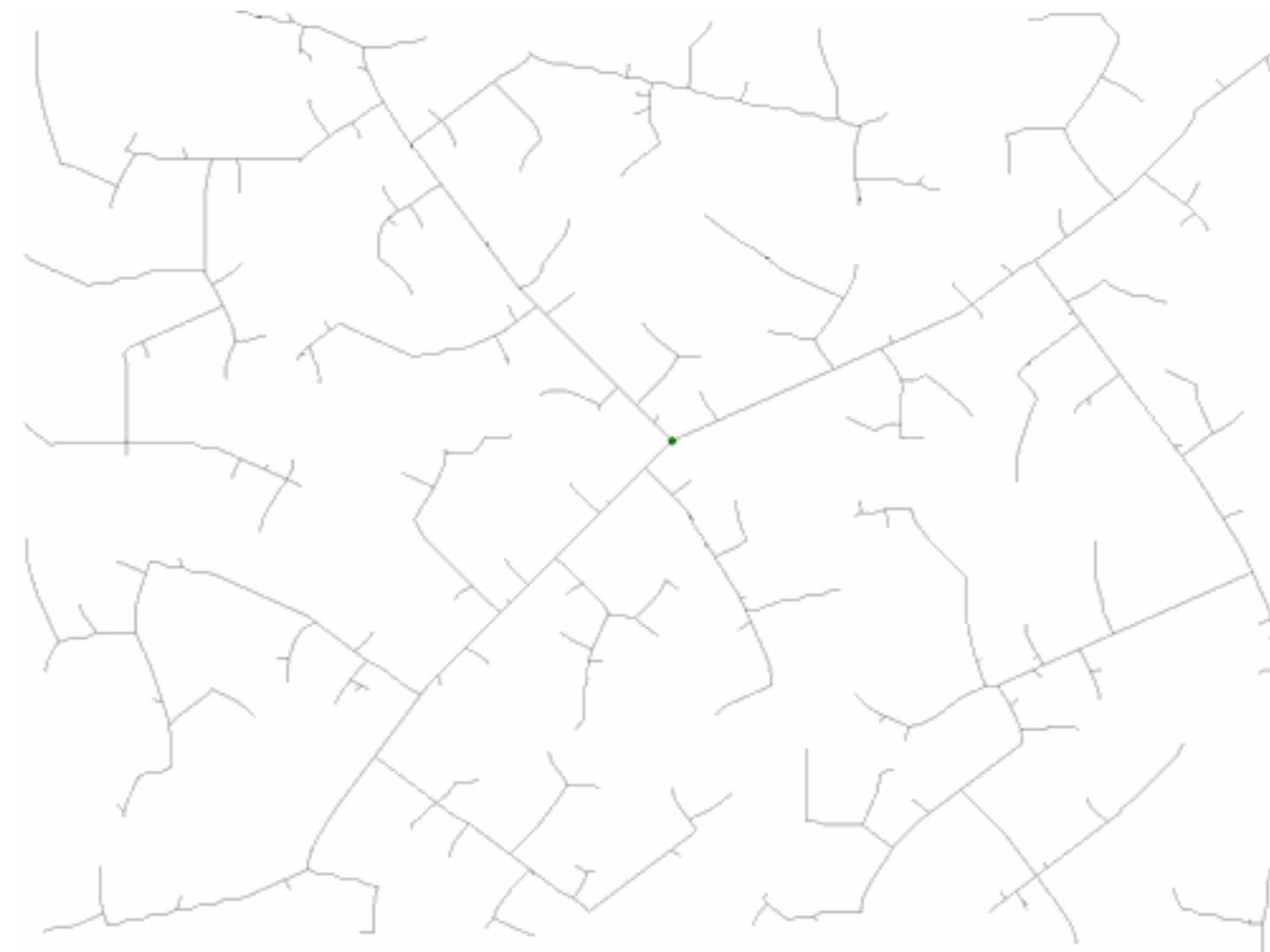


All straight line paths may not be feasible, or a good measure of distance between states.

# Rapidly Exploring Random Trees (RRTs)

## Kinodynamic planning

Build up a tree through generating "next states" in the tree by executing random controls.



# Rapidly Exploring Random Trees (RRTs)

Build up a tree through generating "next states" in the tree by executing random controls.

GENERATE\_RRT( $x_{init}, K, \Delta t$ )

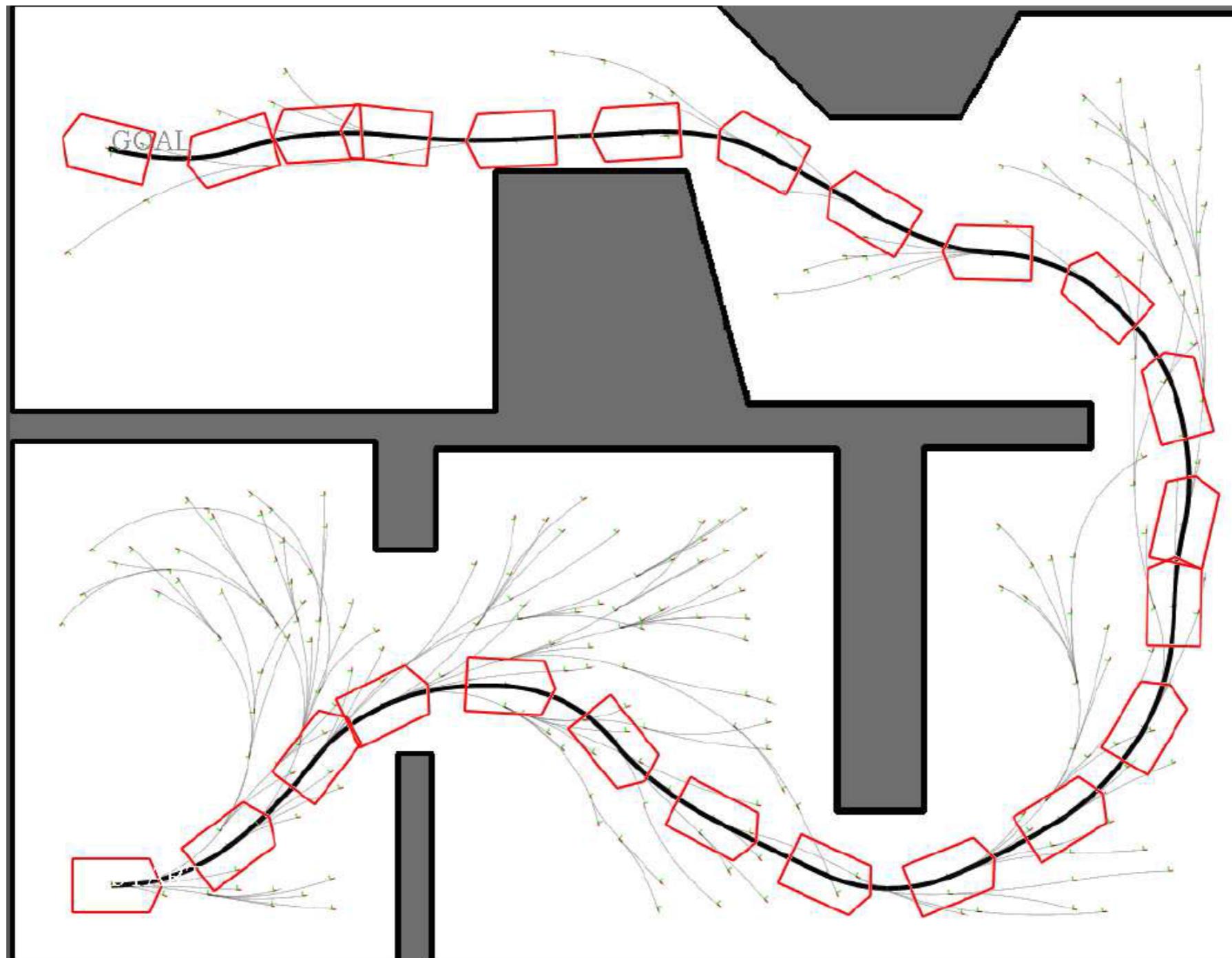
```
1   $\mathcal{T}.\text{init}(x_{init});$ 
2  for  $k = 1$  to  $K$  do
3       $x_{rand} \leftarrow \text{RANDOM\_STATE}();$ 
4       $x_{near} \leftarrow \text{NEAREST\_NEIGHBOR}(x_{rand}, \mathcal{T});$ 
5       $u \leftarrow \text{SELECT\_INPUT}(x_{rand}, x_{near});$ 
6       $x_{new} \leftarrow \text{NEW\_STATE}(x_{near}, u, \Delta t);$ 
7       $\mathcal{T}.\text{add\_vertex}(x_{new});$ 
8       $\mathcal{T}.\text{add\_edge}(x_{near}, x_{new}, u);$ 
9  Return  $\mathcal{T}$ 
```

SELECT\_INPUT( $x_{rand}, x_{near}$ )

- Two point boundary value problem
  - If too hard to solve, often just select best out of a set of control sequences.  
This set could be random, or some well chosen set of primitives.

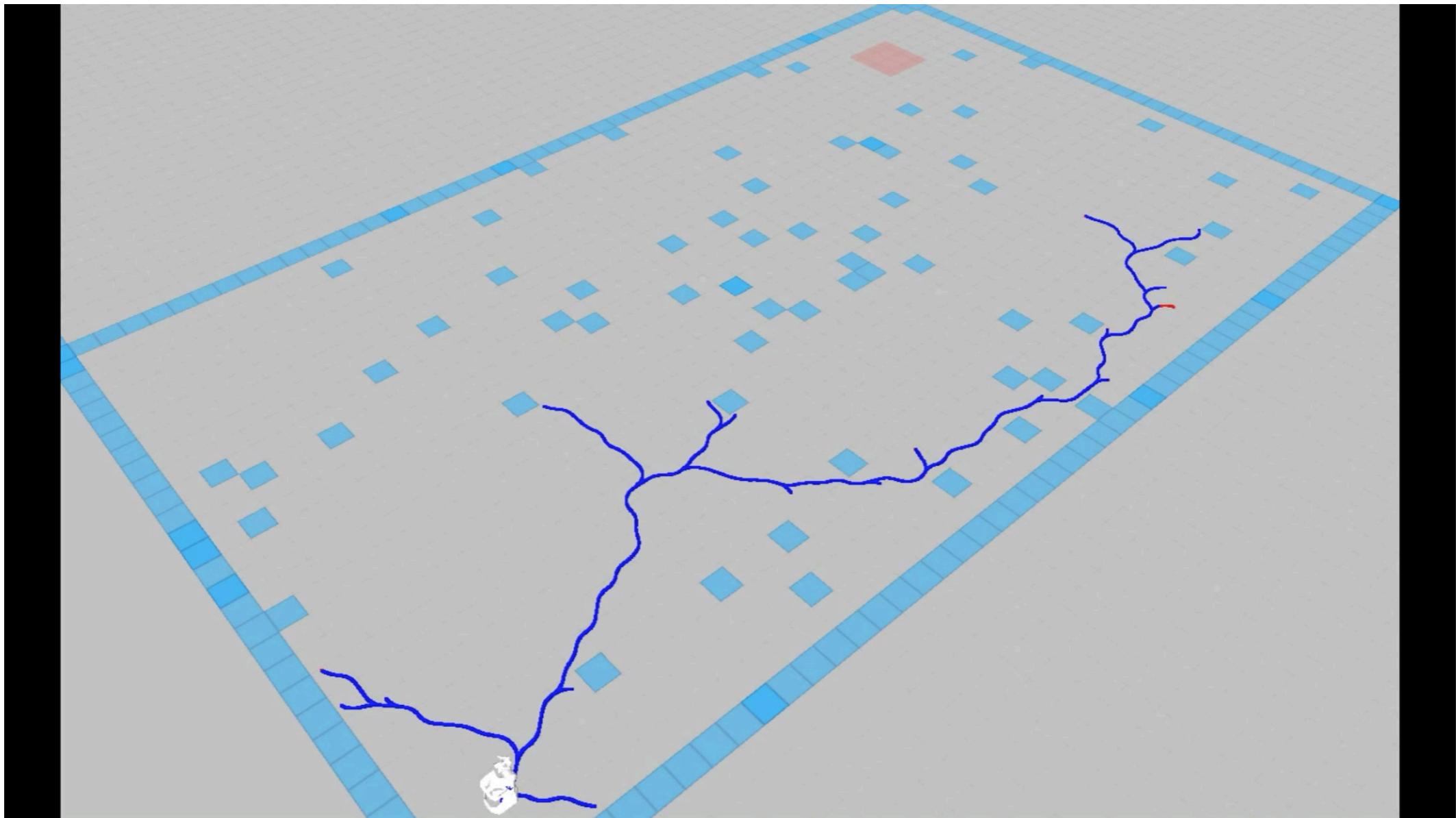
# Rapidly Exploring Random Trees (RRTs)

Build up a tree through generating "next states" in the tree by executing random controls.



# Rapidly Exploring Random Trees (RRTs)

Build up a tree through generating "next states" in the tree by executing random controls.

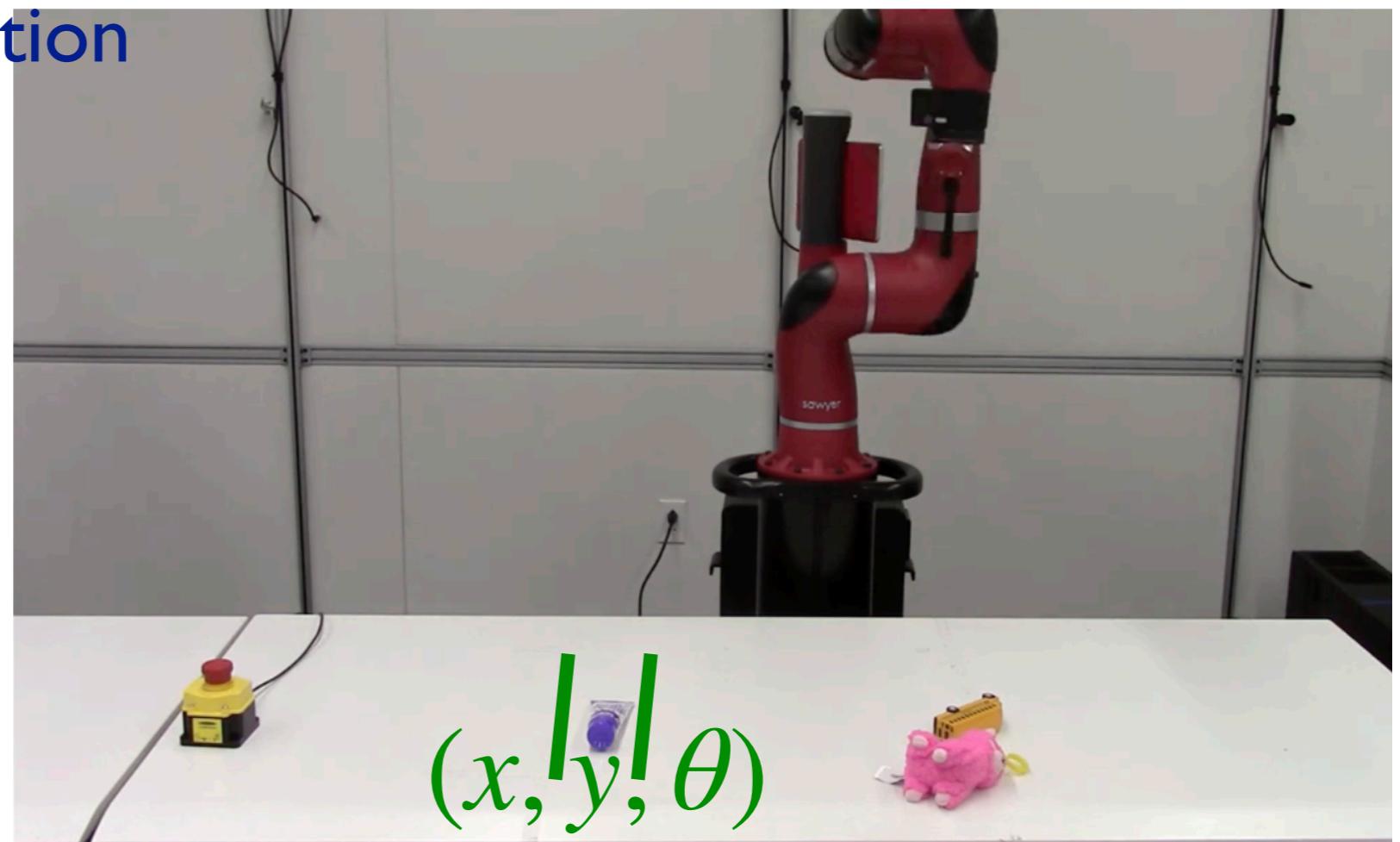
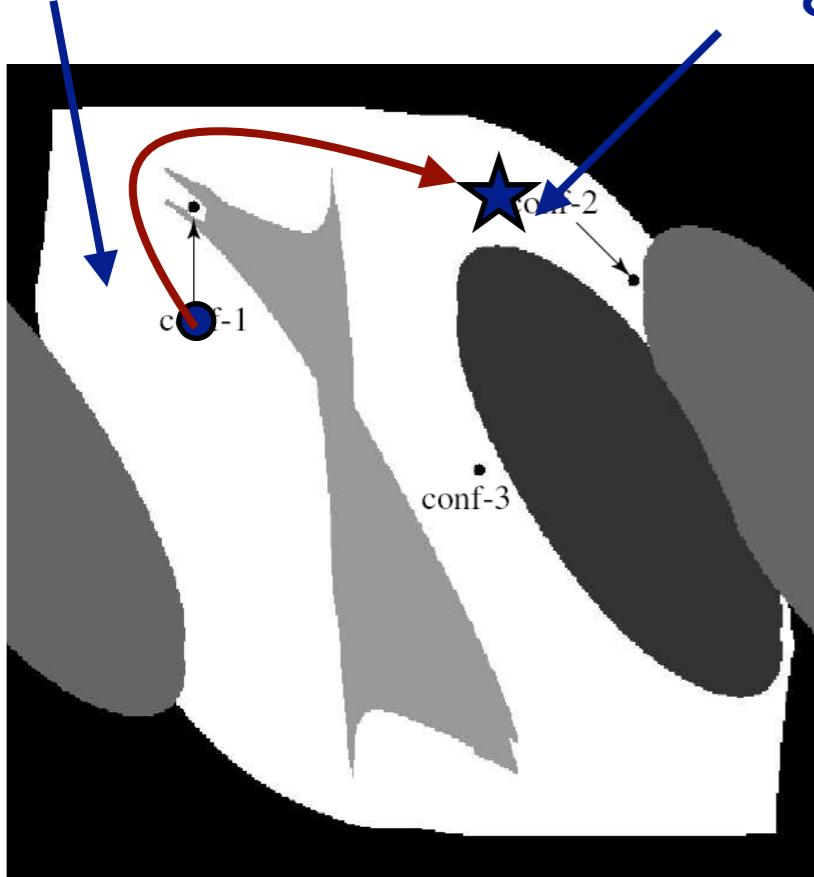


# How to move your robot?

## I. Task space to Configuration space

## 2. Configuration space trajectory

Initial configuration      Desired configuration



$(x, y, \theta)$

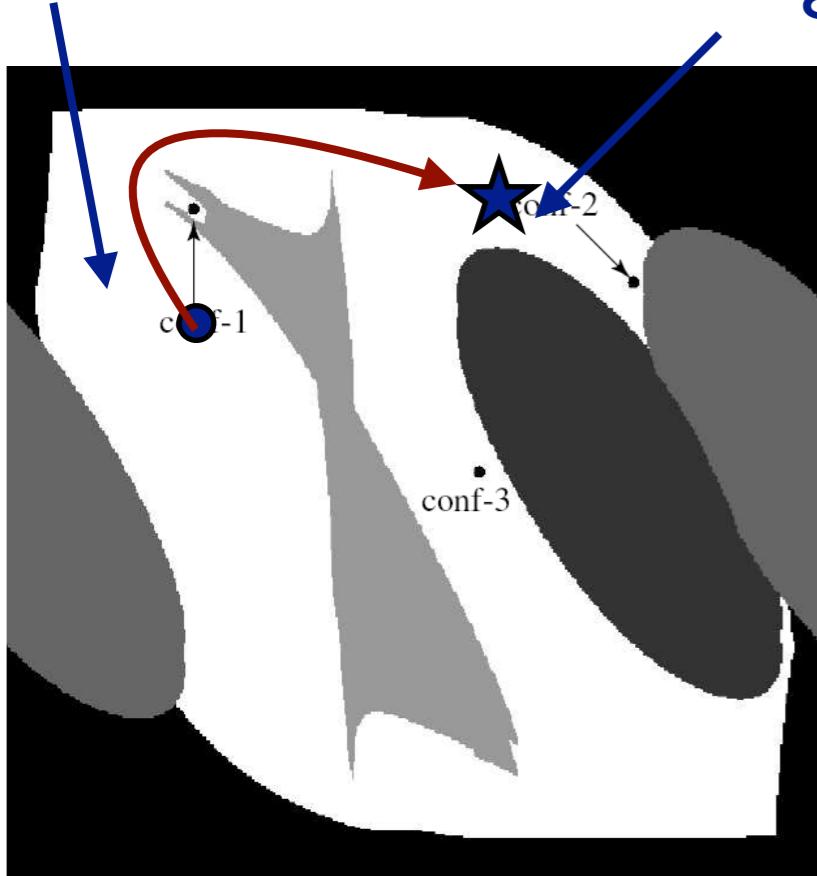
# How to move your robot?

I. Task space to Configuration space

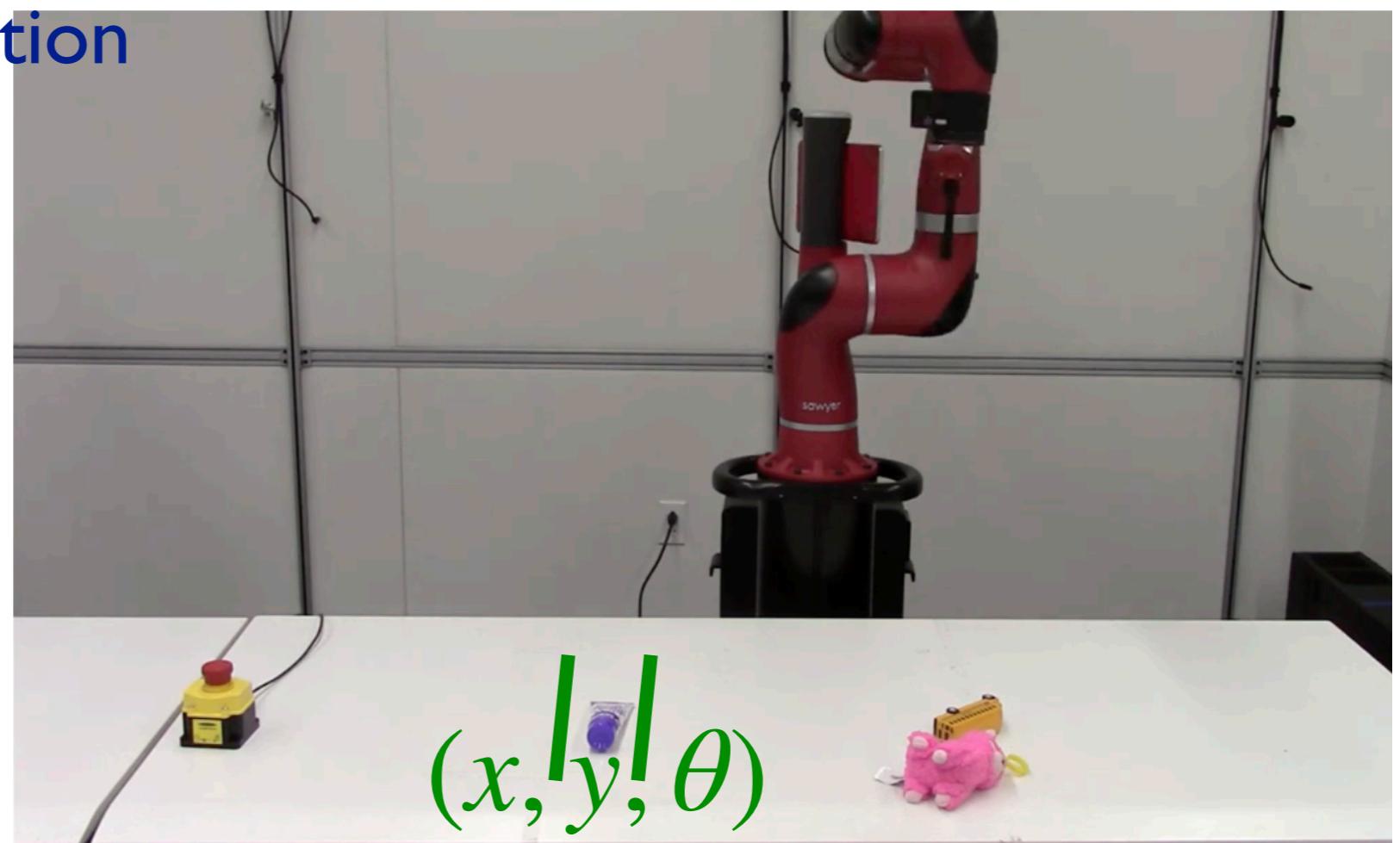
2. Configuration space trajectory

3. Trajectory execution

Initial configuration



Desired configuration



# Trajectory Execution



Dynamically feasible trajectory  $x_t^{ref}$   
from planner



*What control commands should I apply in order  
to get the robot to robustly track this trajectory?*

Robot state  $x_t$

Robot location, or joint angles.

Control sequence  $u_t$

Velocities, torques.

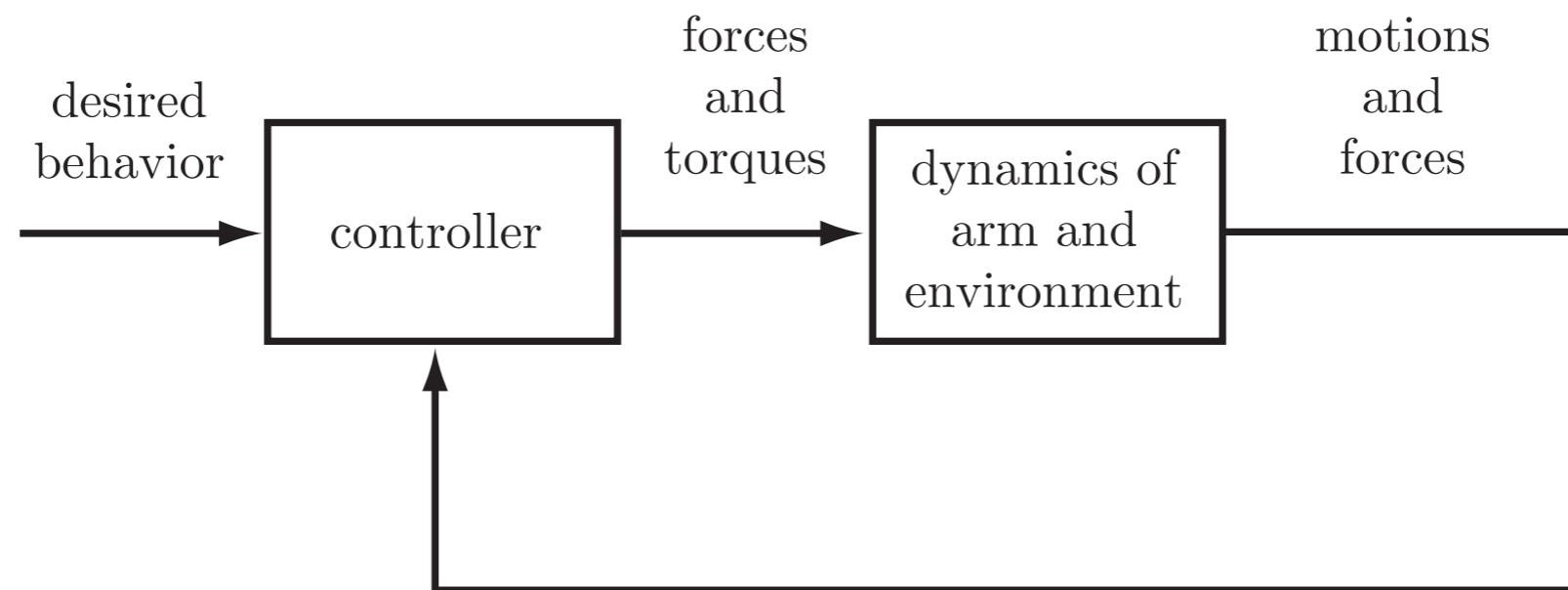
Dynamics function  $x_{t+1} = f(x_t, u_t)$  State evolution as we apply control.

Cost function  $\sum_t \|x_t - x_t^{ref}\|$

*Low-level control can be formulated as an optimization problem.*

# Trajectory Execution

## Feedback Control



# Low-level Control

Simplifying assumptions:

Linear dynamics, quadratic cost.



*Linear Quadratic Regulator*

Exactly solved using dynamic programming.

The LQR setting assumes a linear dynamical system:

$$x_{t+1} = Ax_t + Bu_t,$$

$x_t$ : state at time  $t$

$u_t$ : input at time  $t$

It assumes a quadratic cost function:

$$g(x_t, u_t) = x_t^\top Q x_t + u_t^\top R u_t$$

with  $Q \succ 0, R \succ 0$ .

For a square matrix  $X$  we have  $X \succ 0$  if and only if for all vectors  $z$  we have  $z^\top X z > 0$ . Hence there is a non-zero cost for any state different from the all-zeros state, and any input different from the all-zeros input.

# Linear Quadratic Regulator

Cost if the system is in state  $x$ ,  
and we have  $i$  steps to go.

$$J_i(x)$$

Cost if the system is in state  $x$ ,  
and we have  $i+1$  steps to go.

$$J_{i+1}(x)$$

$$= \min_u x^T Q x + u^T R u + J_i(Ax + Bu)$$

# Linear Quadratic Regulator

$$J_{i+1}(x) \leftarrow \min_u [x^\top Qx + u^\top Ru + J_i(Ax + Bu)]$$

Initialize  $J_0(x) = x^\top P_0 x$ .

$$\begin{aligned} J_1(x) &= \min_u [x^\top Qx + u^\top Ru + J_0(Ax + Bu)] \\ &= \min_u [x^\top Qx + u^\top Ru + (Ax + Bu)^\top P_0(Ax + Bu)] \end{aligned} \quad (1)$$

To find the minimum over  $u$ , we set the gradient w.r.t.  $u$  equal to zero:

$$\nabla_u [\dots] = 2Ru + 2B^\top P_0(Ax + Bu) = 0,$$

$$\text{hence: } u = -(R + B^\top P_0 B)^{-1} B^\top P_0 A x \quad (2)$$

$$\begin{aligned} (2) \text{ into (1): } J_1(x) &= x^\top P_1 x \\ \text{for: } P_1 &= Q + K_1^\top R K_1 + (A + B K_1)^\top P_0 (A + B K_1) \\ K_1 &= -(R + B^\top P_0 B)^{-1} B^\top P_0 A. \end{aligned}$$

# Linear Quadratic Regulator

In summary:

$$J_0(x) = x^\top P_0 x$$

$$x_{t+1} = Ax_t + Bu_t$$

$$g(x, u) = u^\top Ru + x^\top Qx$$

$$J_1(x) = x^\top P_1 x$$

$$\text{for: } P_1 = Q + K_1^\top R K_1 + (A + B K_1)^\top P_0 (A + B K_1)$$

$$K_1 = -(R + B^\top P_0 B)^{-1} B^\top P_0 A.$$

$J_1(x)$  is quadratic, just like  $J_0(x)$ .

Update is the same for all times and can be done in closed form for this particular continuous state-space system and cost!

$$J_2(x) = x^\top P_2 x$$

$$\text{for: } P_2 = Q + K_2^\top R K_2 + (A + B K_2)^\top P_1 (A + B K_2)$$

$$K_2 = -(R + B^\top P_1 B)^{-1} B^\top P_1 A.$$

# Linear Quadratic Regulator

Set  $P_0 = 0$ .

for  $i = 1, 2, 3, \dots$

$$\begin{aligned} K_i &= -(R + B^\top P_{i-1} B)^{-1} B^\top P_{i-1} A \\ P_i &= Q + K_i^\top R K_i + (A + B K_i)^\top P_{i-1} (A + B K_i) \end{aligned}$$

The optimal policy for a  $i$ -step horizon is given by:

$$\pi(x) = K_i x$$

The cost-to-go function for a  $i$ -step horizon is given by:

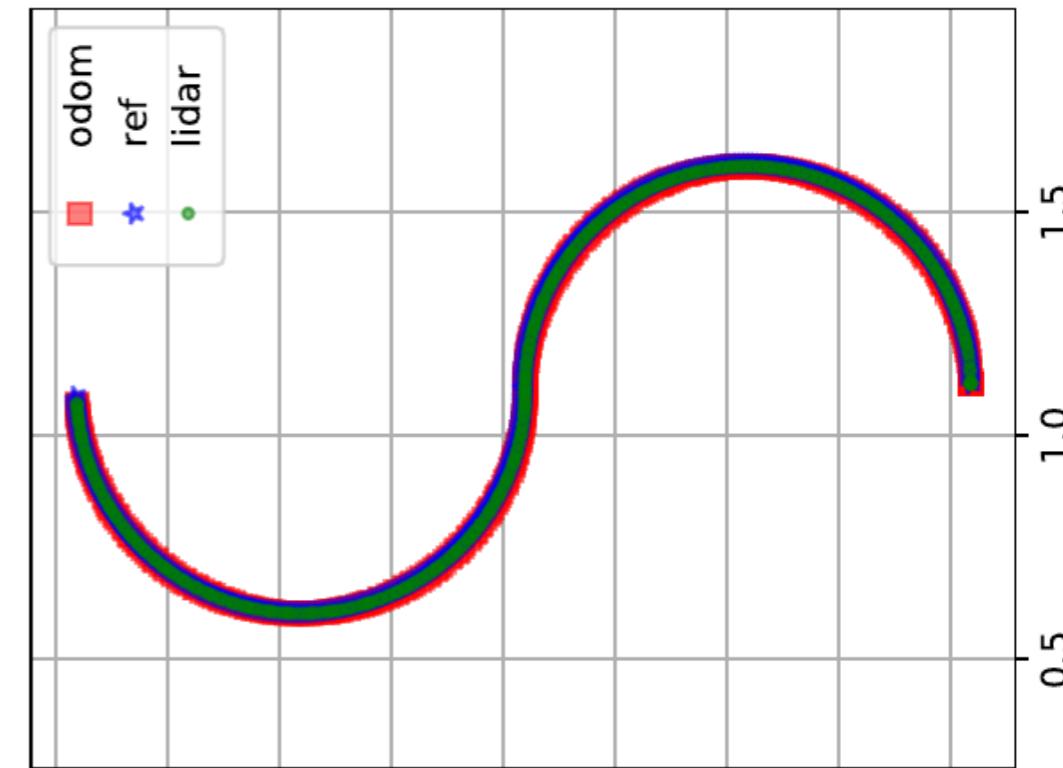
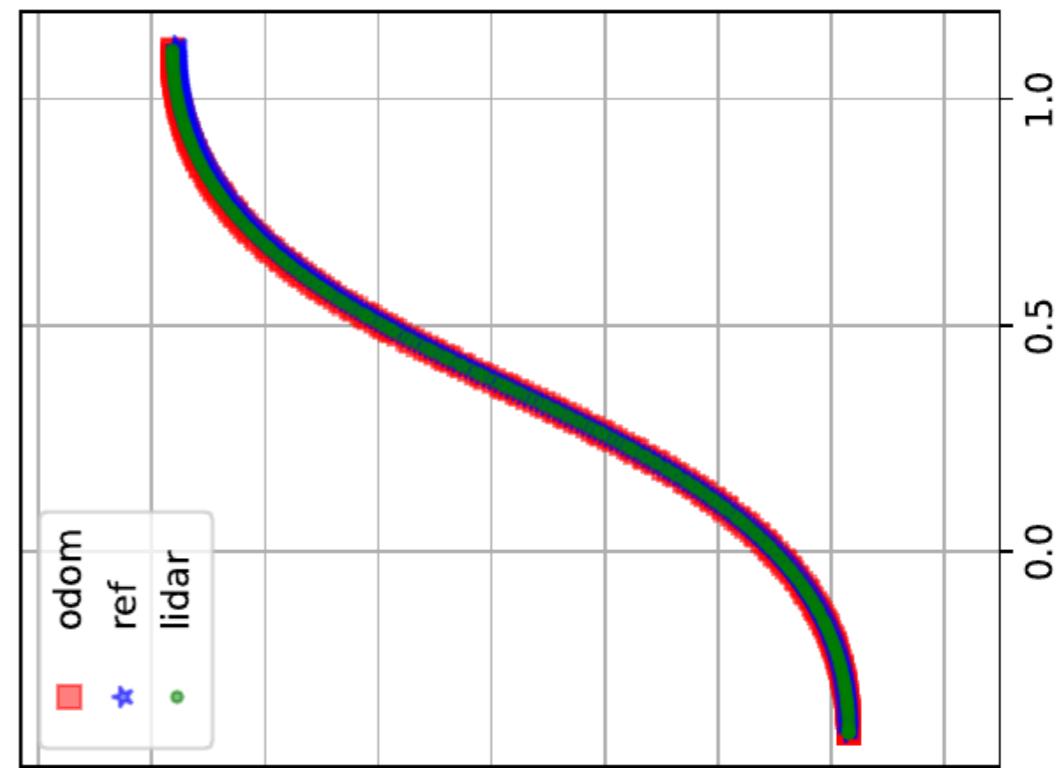
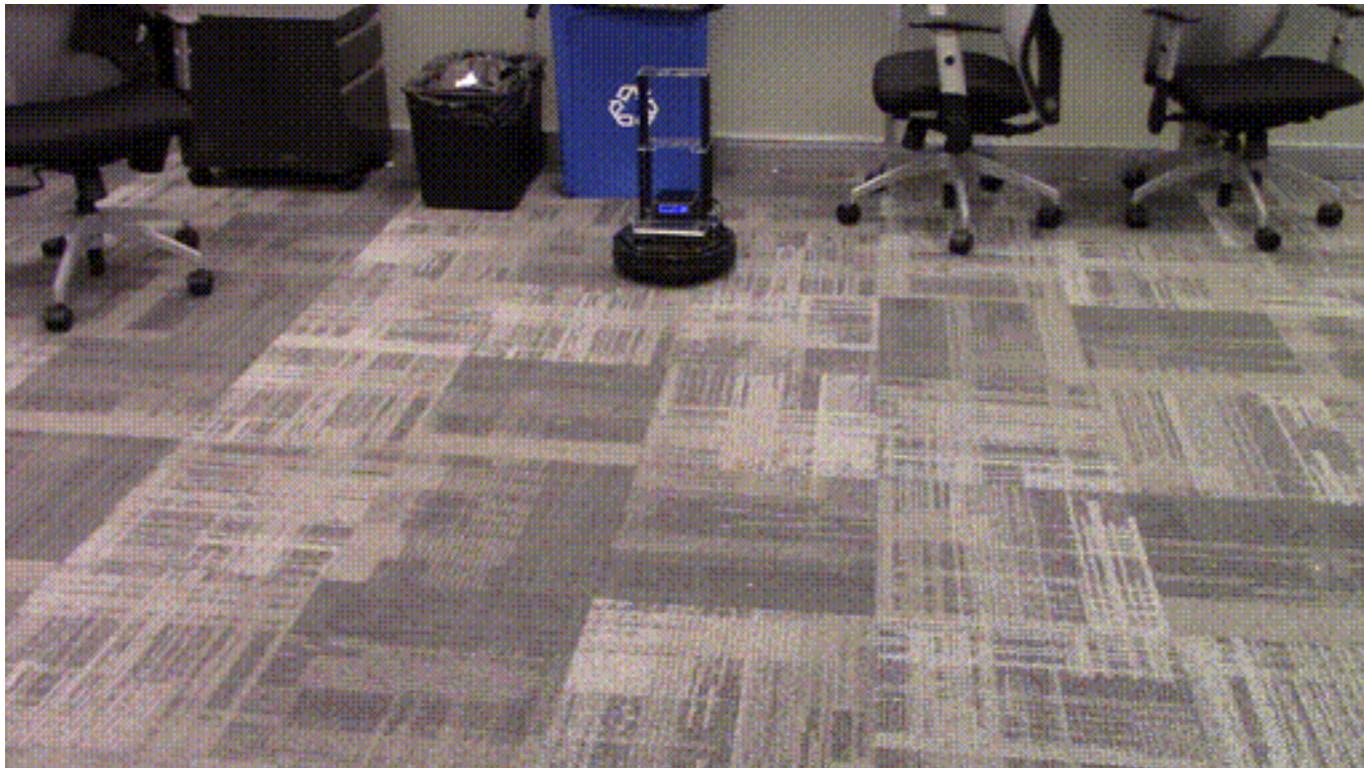
$$J_i(x) = x^\top P_i x.$$

# Linear Quadratic Regulator

Extensions which make it more generally applicable:

- Affine systems System with stochasticity
- Regulation around non-zero fixed point for non-linear systems
- Penalization for change in control inputs
- Linear time varying (LTV) systems
- Trajectory following for non-linear systems

# Linear Quadratic Regulator



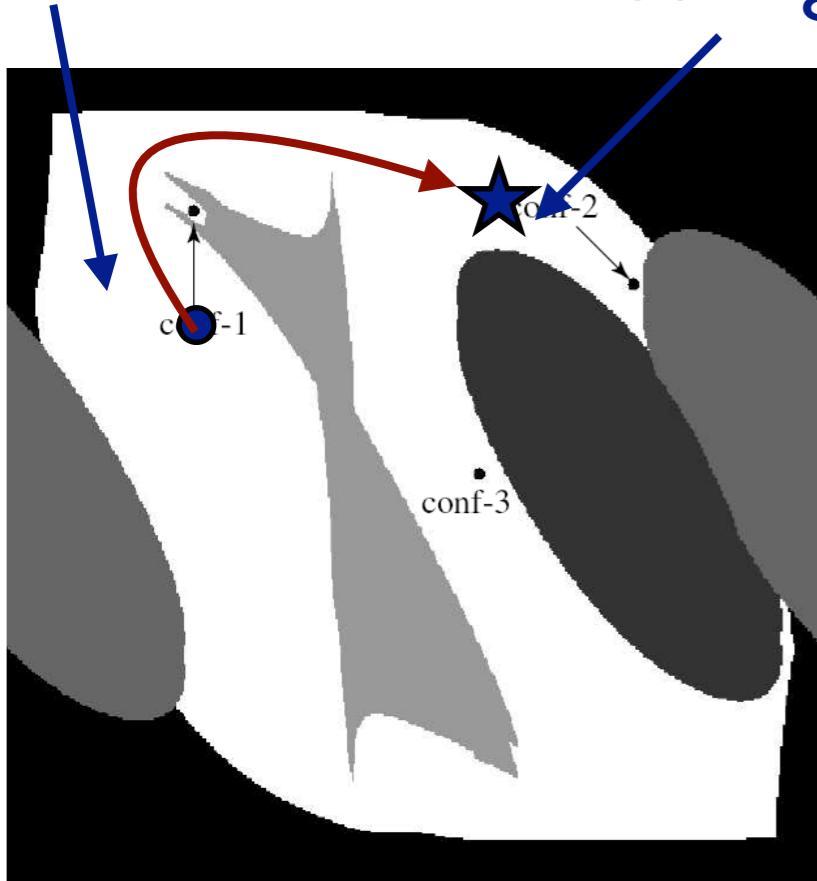
# How to move your robot?

I. Task space to Configuration space

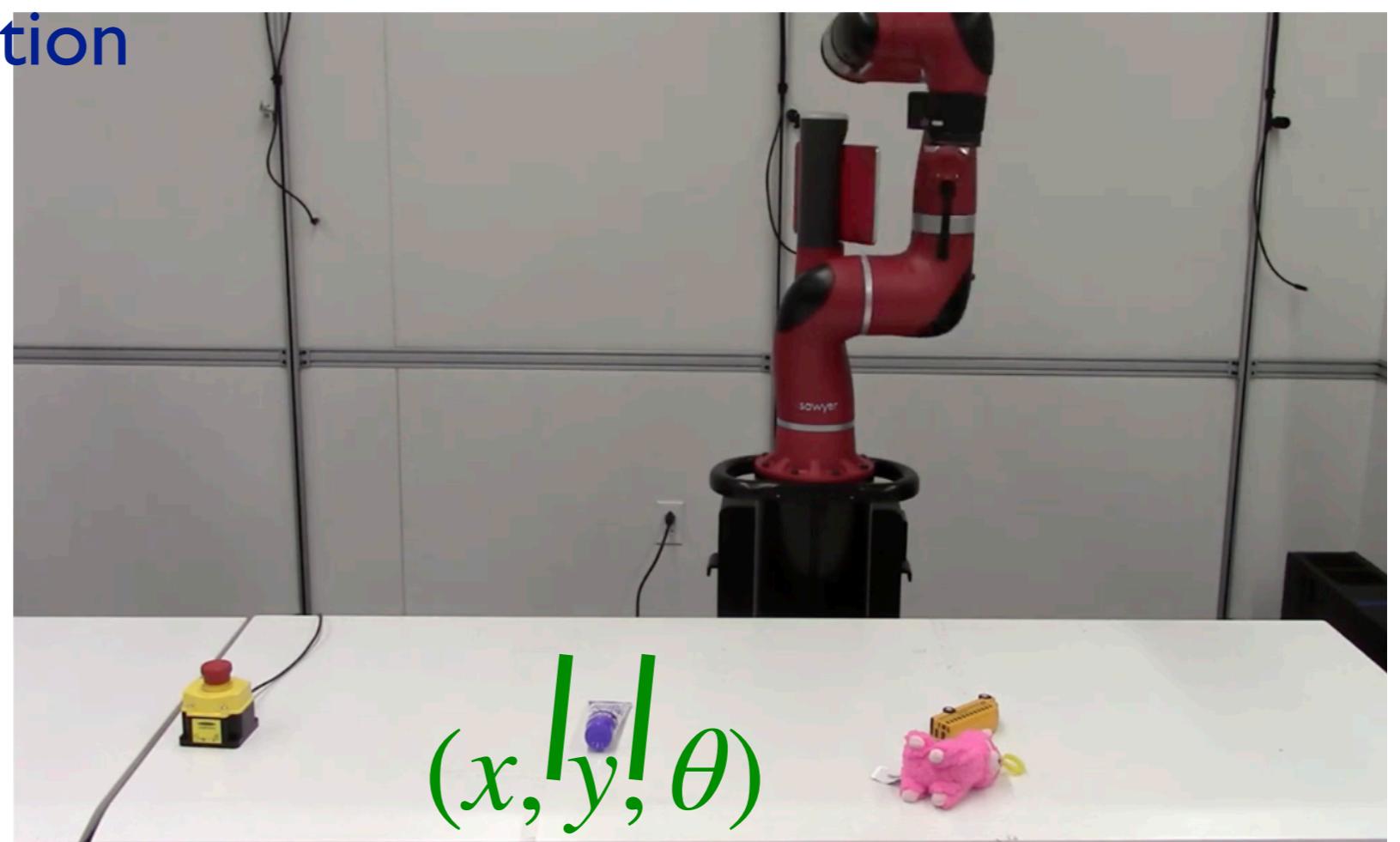
2. Configuration space trajectory

3. Trajectory tracking

Initial configuration

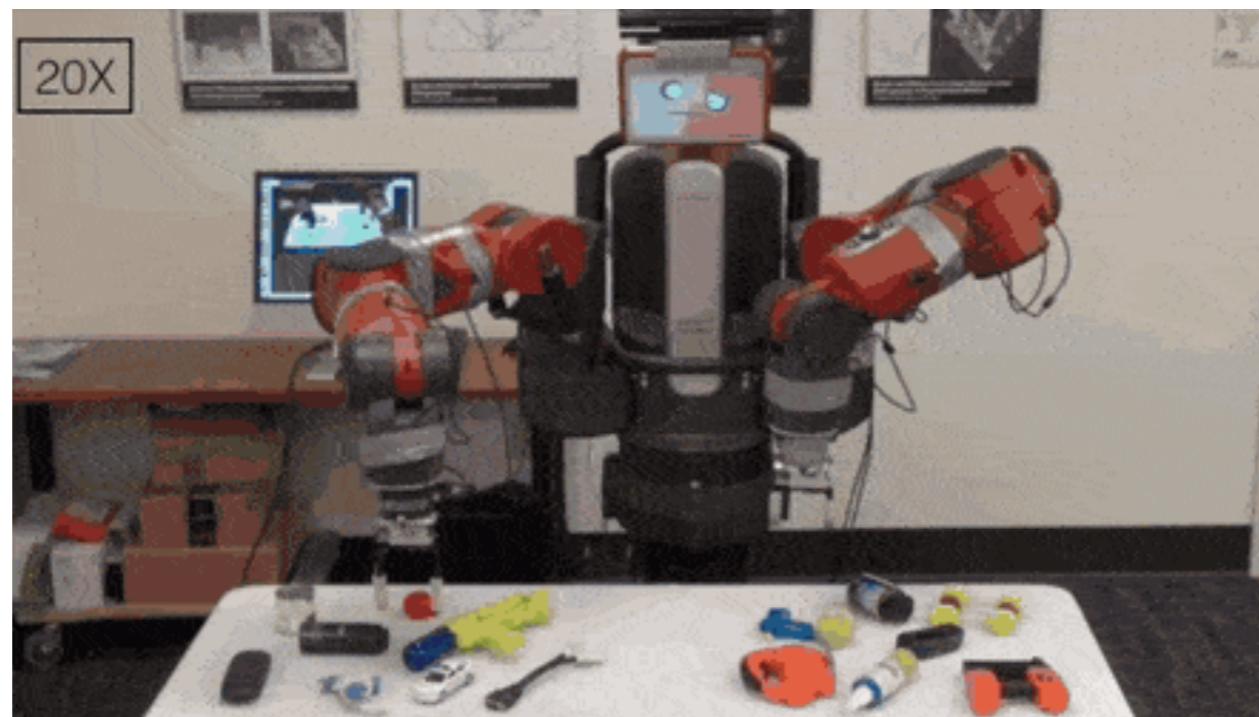


Desired configuration

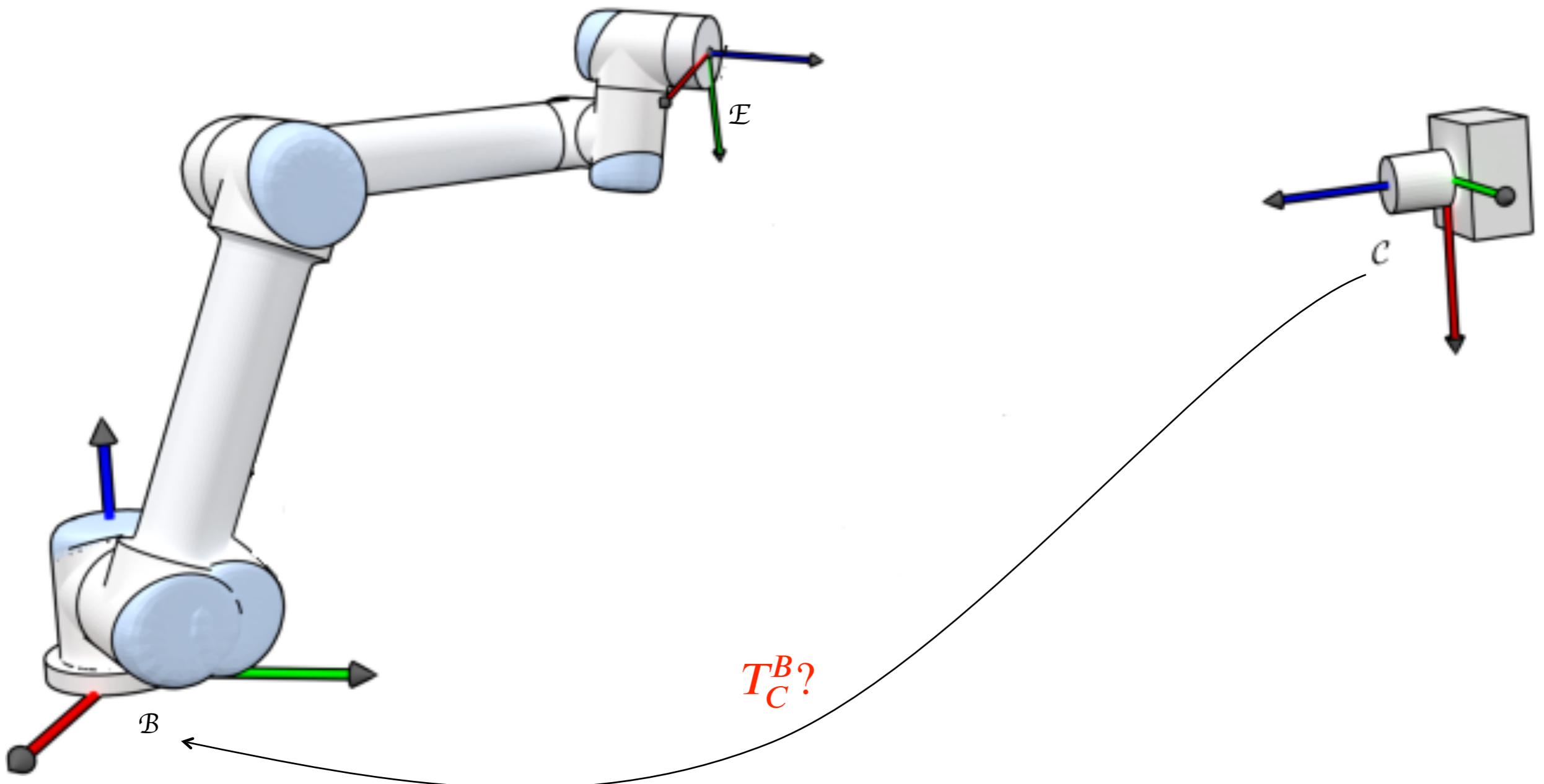


# “Minor” Detail

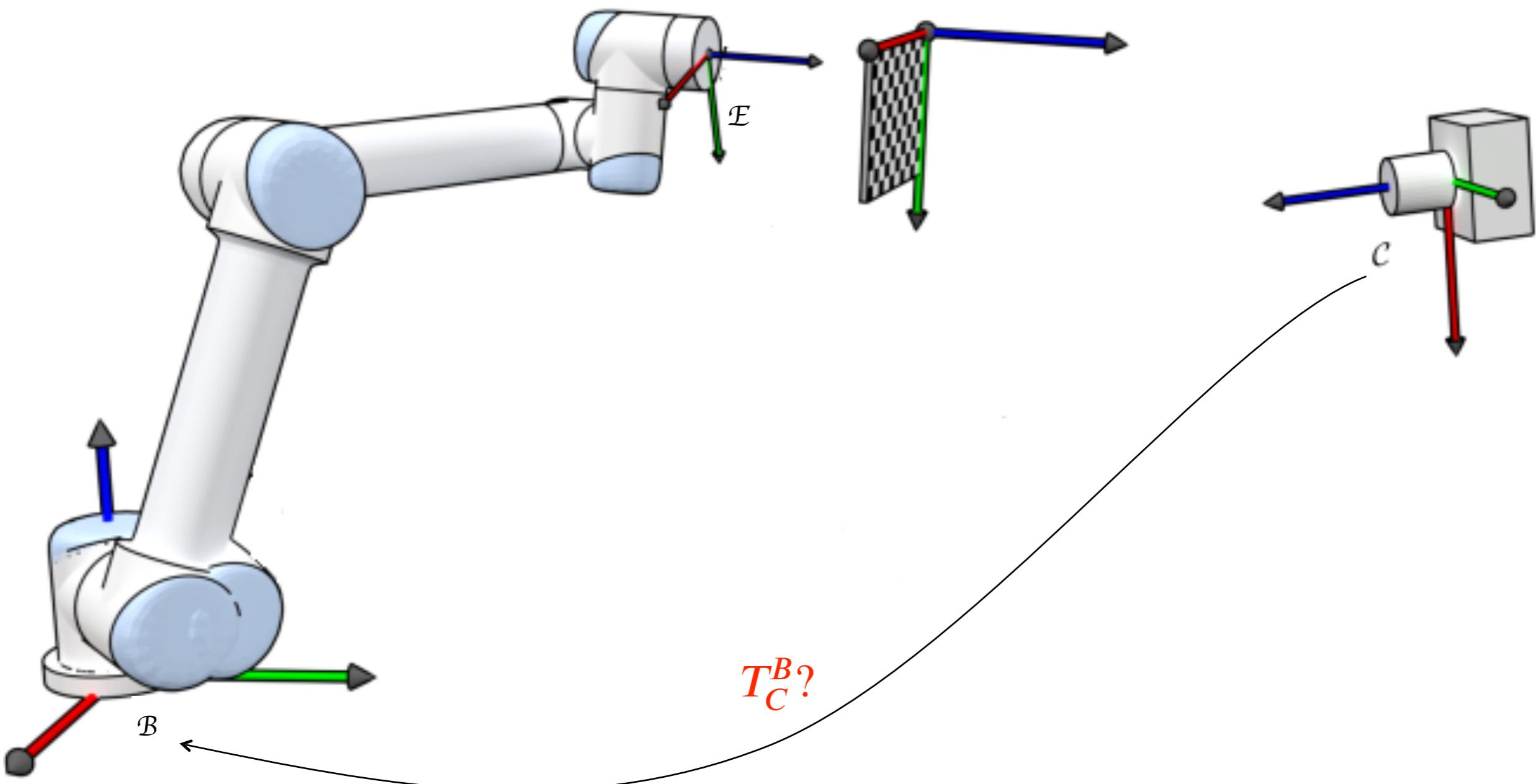
## Camera Calibration



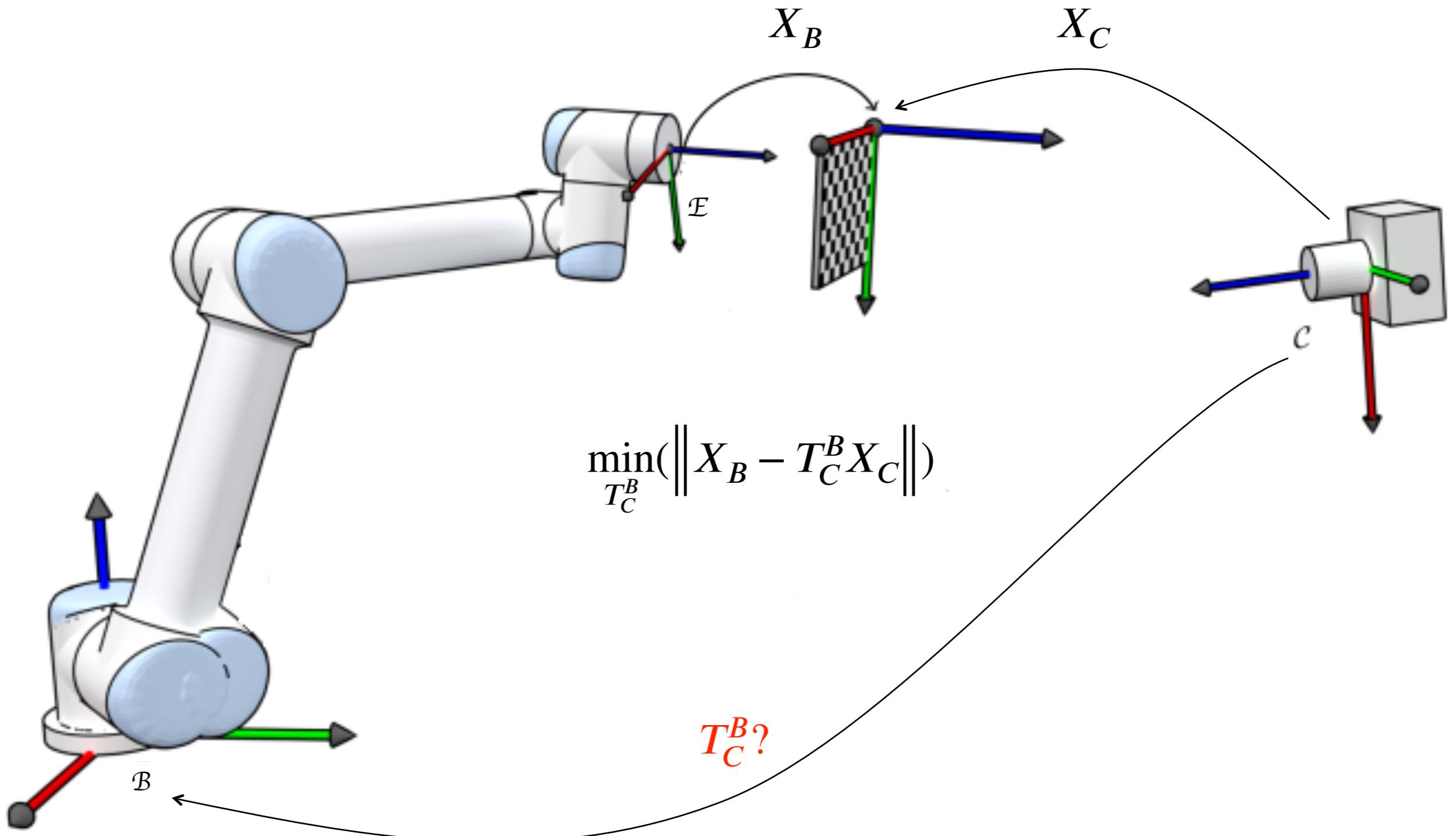
# Camera Calibration



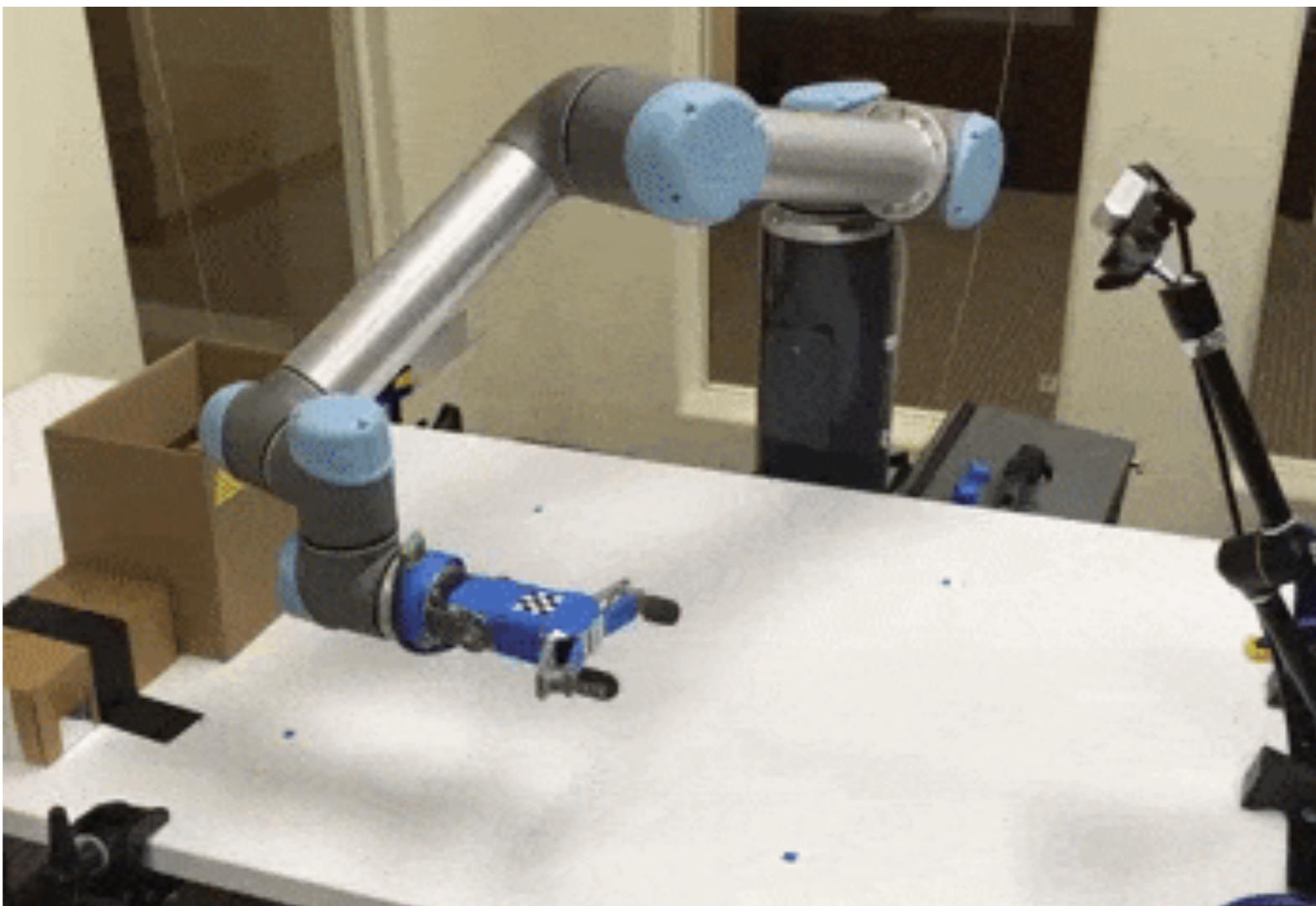
# Camera Calibration



# Camera Calibration



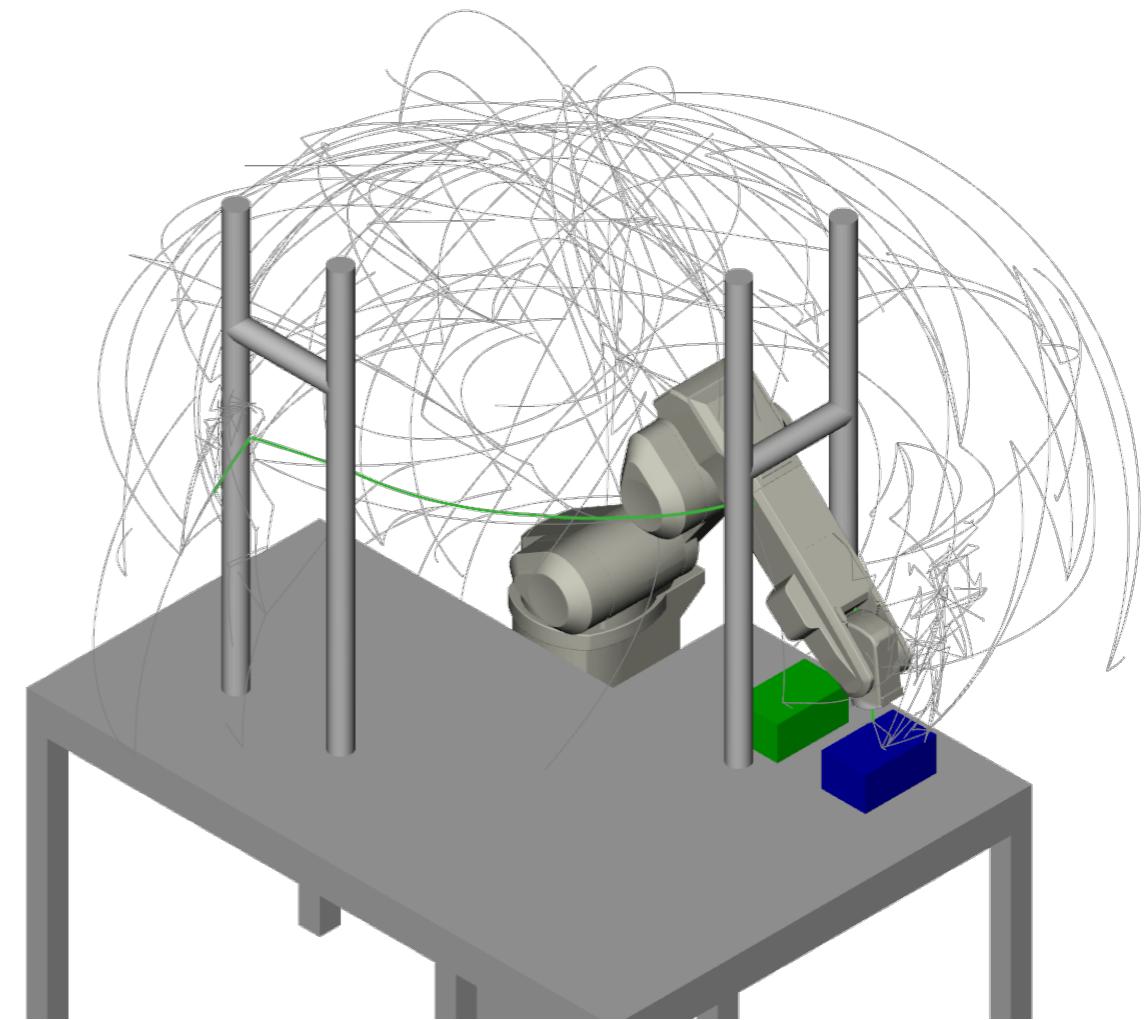
# Camera Calibration



$$\min_{T_C^B} \sum_{i=1}^n \|X_B^i - T_C^B X_C^i\|$$

# Good Softwares

> **MoveIt!**



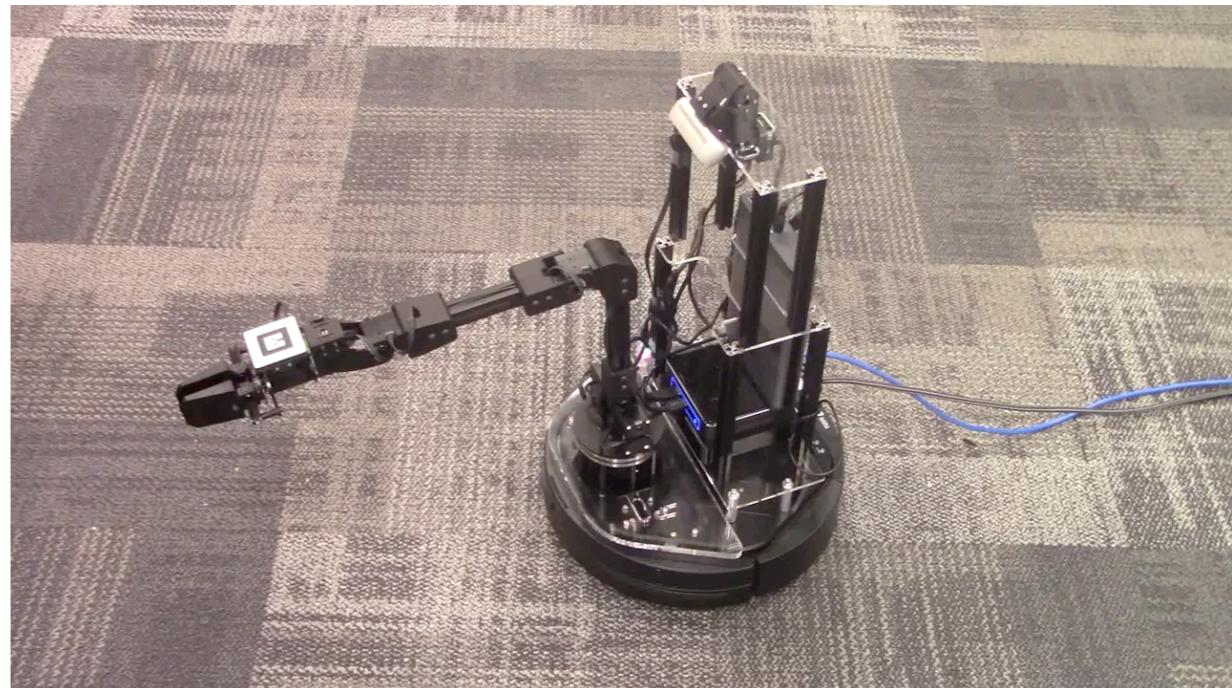
```
<robot name="baxter">
  <link name="base">
    </link>
  <link name="torso">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://baxter_description/meshes/torso/base_link.DAE"
      </geometry>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://baxter_description/meshes/torso/base_link_col
      </geometry>
    </collision>
  </link>
</robot>
```

# Movel! Example

```
target_poses = [
    {'position': np.array([0.28, 0.17, 0.22]),
     'pitch': 0.5,
     'numerical': False},
    {'position': np.array([0.28, -0.17, 0.22]),
     'pitch': 0.5,
     'roll': 0.5,
     'numerical': False}
]

robot.arm.go_home()

for pose in target_poses:
    robot.arm.set_ee_pose_pitch_roll(**pose)
    time.sleep(1)
robot.arm.go_home()
```

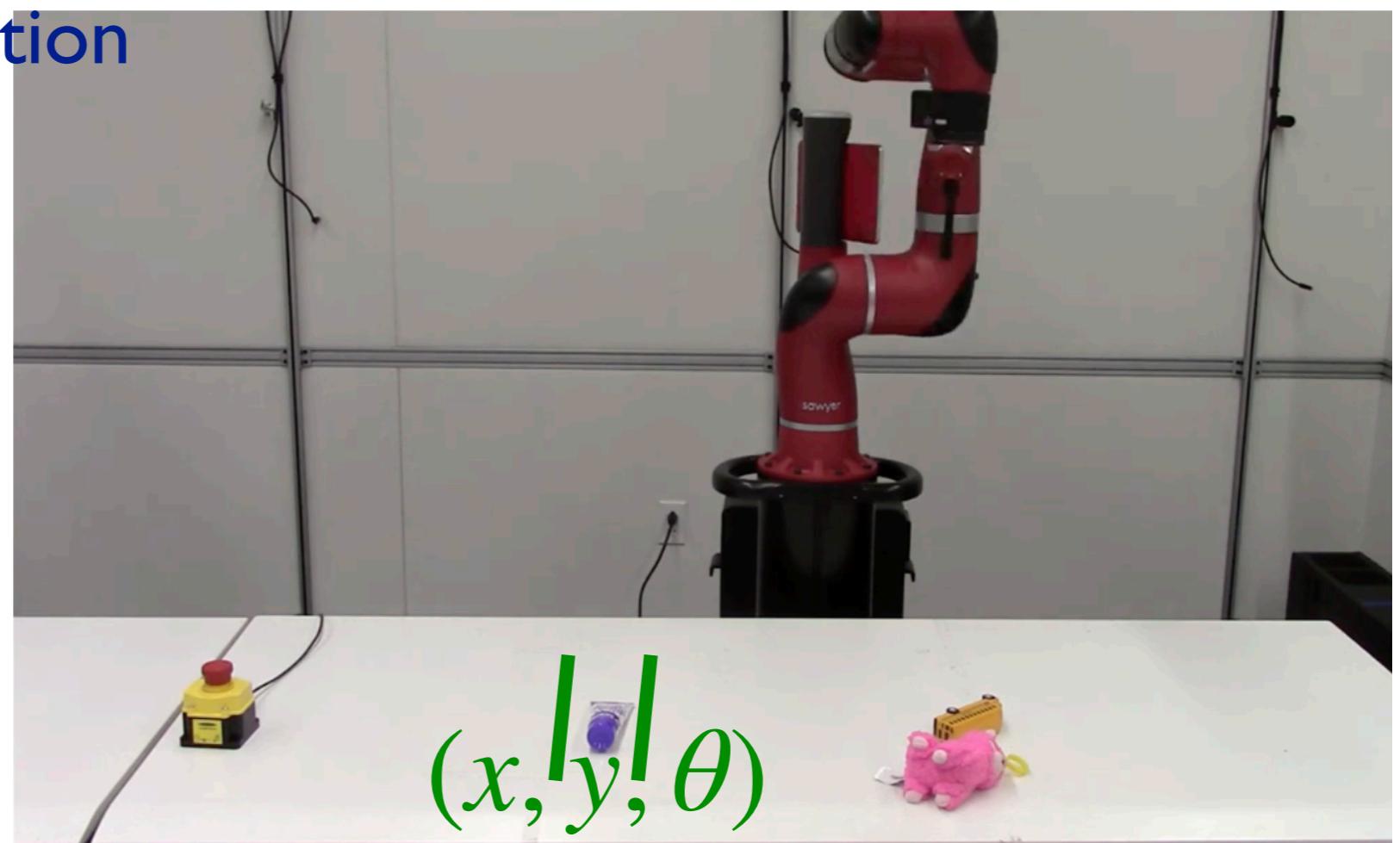
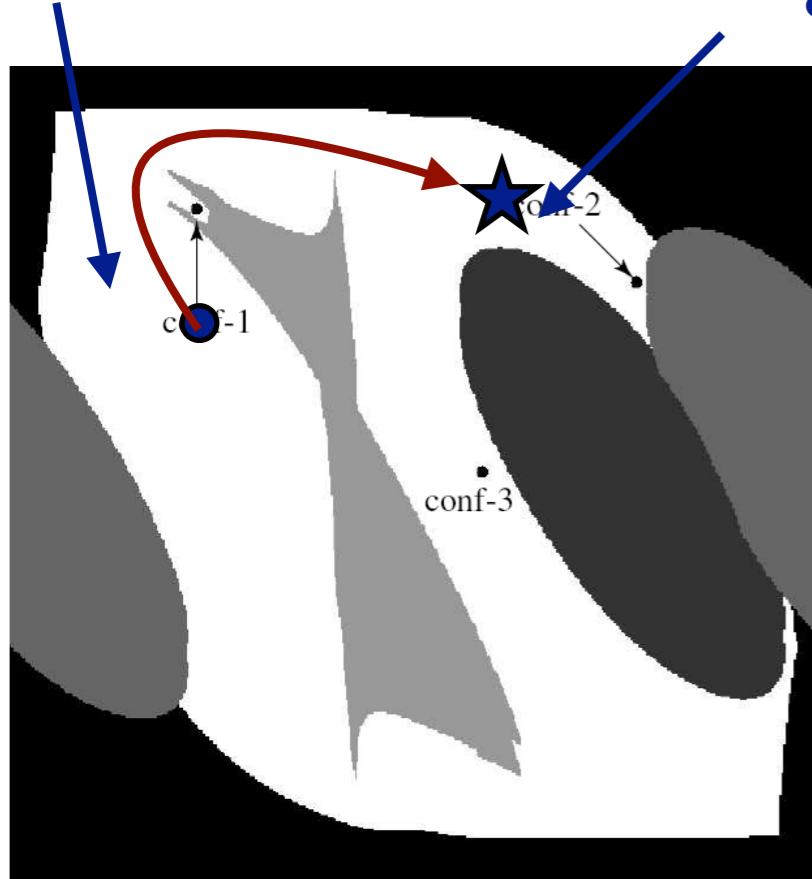


# Robotics Review: How to move your robot?

1. Task space to Configuration space
2. Configuration space trajectory
3. Trajectory execution

Configuration Space  
Forward / Inverse Kinematics  
Motion Planning  
Optimal Control

Initial configuration      Desired configuration



$$(x, |y|, \theta)$$

# Resources

Kris Hauser's Robotic Systems Book

<http://motion.cs.illinois.edu/RoboticSystems/>

Pieter Abbeel's Advanced Robotics Course at Berkeley

<https://people.eecs.berkeley.edu/~pabbeel/cs287-fa19/>

Howie Choset's Robotic Motion Planning Course at CMU

<https://www.cs.cmu.edu/~motionplanning/>

Control Systems

<https://courses.engr.illinois.edu/ece486/fa2021/>

Intro to Robotics

<https://publish.illinois.edu/ece470-intro-robotics/>