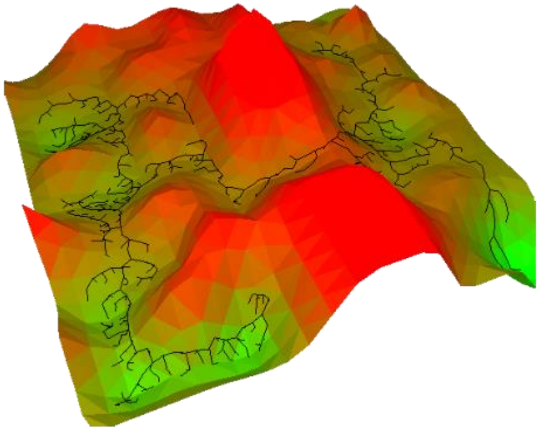
Trajectory Optimization

Jane Li
Assistant Professor
Mechanical Engineering & Robotics Engineering
http://users.wpi.edu/~zli11

Recap

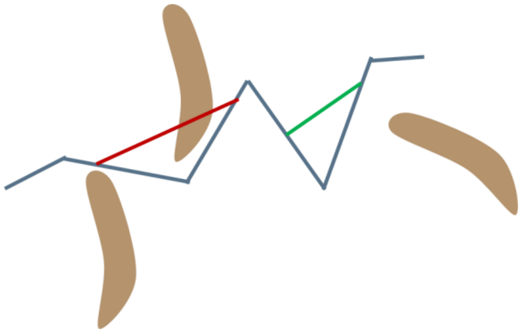
- We heard about RRT*, a sampling-based planning in high-dimensional cost spaces



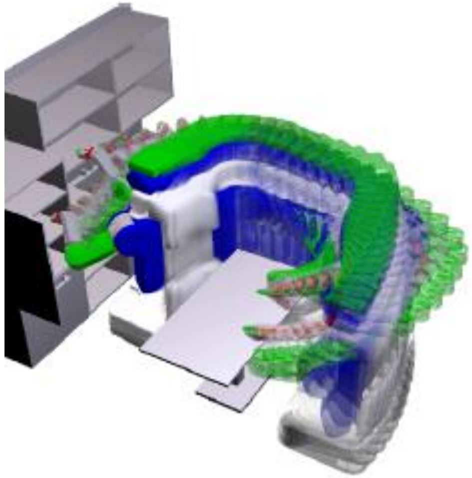
- These kinds of methods can be quite time consuming, what if we need something fast?
- **Trajectory optimization**:
  - Fast ☺
  - Made for optimality ☺
  - Has trouble with feasibility ☹
  - Has trouble with local minima ☹

A good choice if you really need speed and feasibility constraints aren't severe.

Examples



- Naïve optimization
  - Smoothing
- (Arm) motion planning
  - Use straight line segments to connect start, end and via points
  - Optimize the piecewise trajectory for some merits (e.g. min time, energy consumption)
- Other concerns
  - Obstacle avoidance
  - Constraints



Trajectory Optimization

- Trajectory optimization has two roles
  - Smooth and shorten trajectories generated by other methods (e.g. RRT)
  - Plan from scratch – given a initial trajectory that contains collisions and may violate constraints, optimize for a high-quality collision-free trajectory that satisfies constraints
- Key ingredients in trajectory optimization
  - Numerical optimization methods
  - Methods that check and penalize collision

Performance of Trajectory Optimization Algorithms

- Speed
  - How fast to solve a problem of (high dimension)?
- Reliability
  - Can it solve a large fraction of problem?
- Path quality
  - Free of collision? Or even treat collision as hard constraints (e.g. keep a safe distance from the obstacles)?
- Flexibility
  - How easy to add new constraints and cost function terms?

Problem Formulation

- Solve this optimization problem:

$$\arg\min_{\tau} C(\tau)$$

- Uncertainty
- Smoothness
- Distance from obstacles

....

- Subject to these constraints:

$$f(\tau) = 0$$

$$g(\tau) \leq 0$$

$$h(\tau) < 0$$

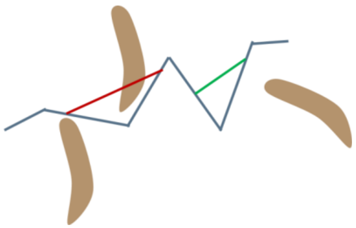
Example?

Example?

Example?

Trajectory Space

- We are no longer searching in C-space, we are searching in trajectory space



- Trajectory space is **infinite dimensional**!
  - We thus optimize on a discretization of a trajectory (still a lot of dimensions!)
- In the next slides we talk about gradients and sampling, we are talking about sampling/gradient **for trajectories, not configurations**.

How do we solve this?

- Many options and strategies.
  - This is an active research area!

$$\arg\min_{\tau} C(\tau)$$

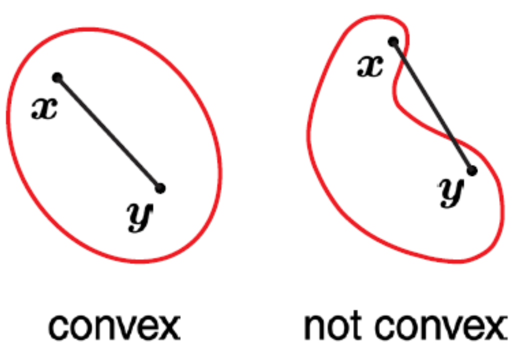
$$f(\tau) = 0$$
$$g(\tau) \leq 0$$
$$h(\tau) < 0$$

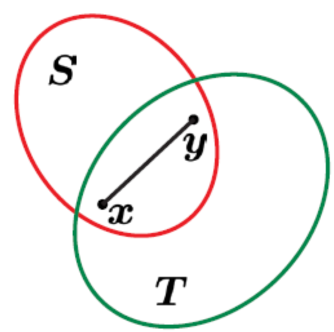
Option 1: Gradient Descent

- Option 1: Put the constraints into the cost function and solve by gradient descent
  - Very fast!
  - What guarantees are there on optimality?
  - What guarantees are there on feasibility?

Option 2: Linear/Quadratic/Convex programming

- Method
  - Formulate the cost function to be linear/quadratic/convex
  - Use linear/quadratic/convex programming to solve
- Pros and Cons
  - Linear/quadratic/convex solver may not be very fast
  - Some constraints/cost functions are hard to represent this way. Examples?
  - What guarantees do we have on optimality?





Option 3: Combine options 1 and 2

- Method
  - Compute linear/quadratic/convex approximation to constraints/cost locally (i.e at the current τ)
  - Use it to get small deformation of trajectory (i.e. the gradient) and iterate
- Pros and Cons
  - Very promising!
  - Local linear/quadratic/convex approximation may still be hard to formulate

Option 4: Sampling Trajectory Space

- Method
  - Sample around current trajectory in trajectory space
  - Compute gradient from the information you learn by sampling, iterate
- Pros and Cons
  - Don't need an approximate model of anything or to compute gradients; easy to implement
  - Requires evaluating cost function on a lot of trajectories (could be slow)
  - Very unclear how to sample well

Summary

- Trajectory optimization is usually **faster** for **high-dimensional cost-space planning** than sampling-based methods

- An active research area with many methods:
  1. Put constraints in cost function and do gradient descent → lagrange multipliers
  2. Linear/quadratic/convex programming
     - Limited application for real-world problems
  3. Combine 1 and 2
  4. Sample in trajectory space, compute gradient from samples

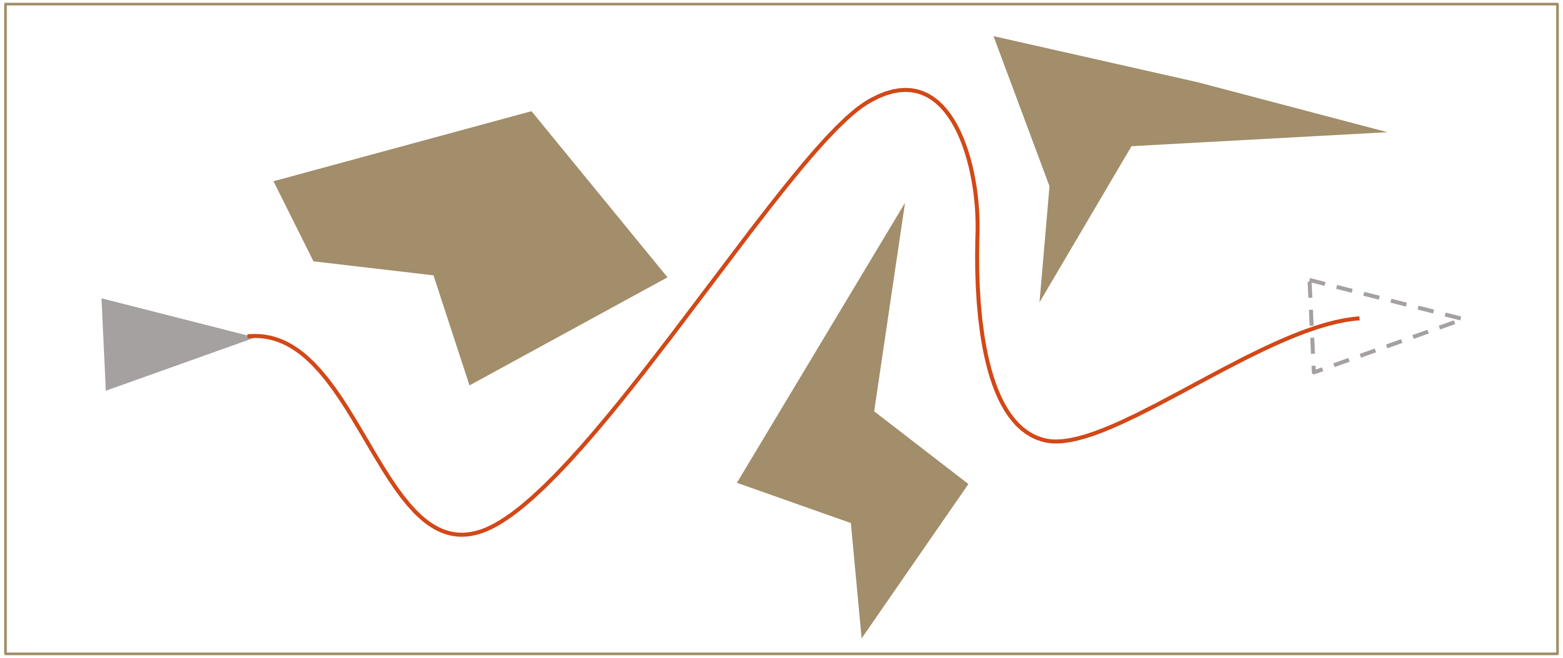
- Trajectory optimizers often have trouble with **obstacles** and **local minima**

Putting it all together

Jane Li
Assistant Professor
Mechanical Engineering & Robotics Engineering
http://users.wpi.edu/~zli11

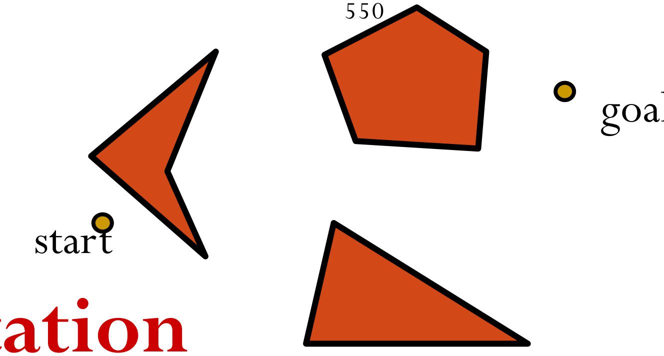
What is motion planning?

- The automatic generation of motion



- The fundamental concepts you need to know to do this are…

Framework



continuous representation

(configuration space formulation)

↓

discretization

(random sampling, processing critical geometric events)

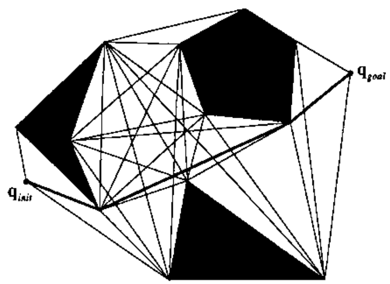
↓

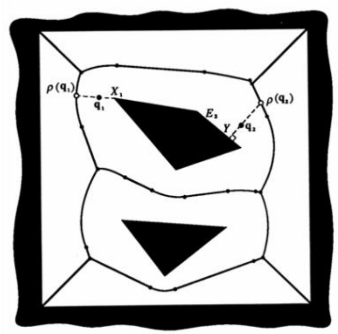
graph searching

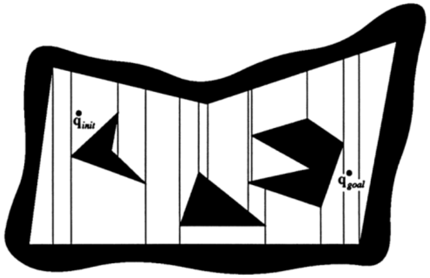
(breadth-first, best-first, A*)

Discretization

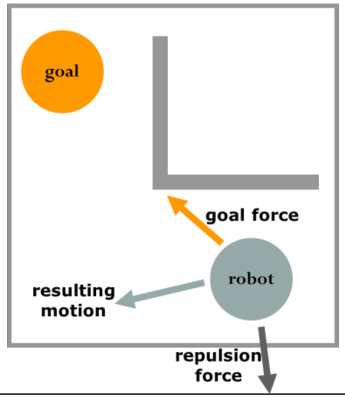
- Sampling (random, with bias)
- Processing critical geometric features
  - (Reduced) Visibility graph, Voronoi graph, cell decomposition

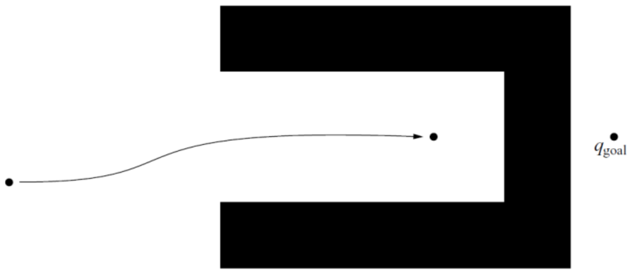




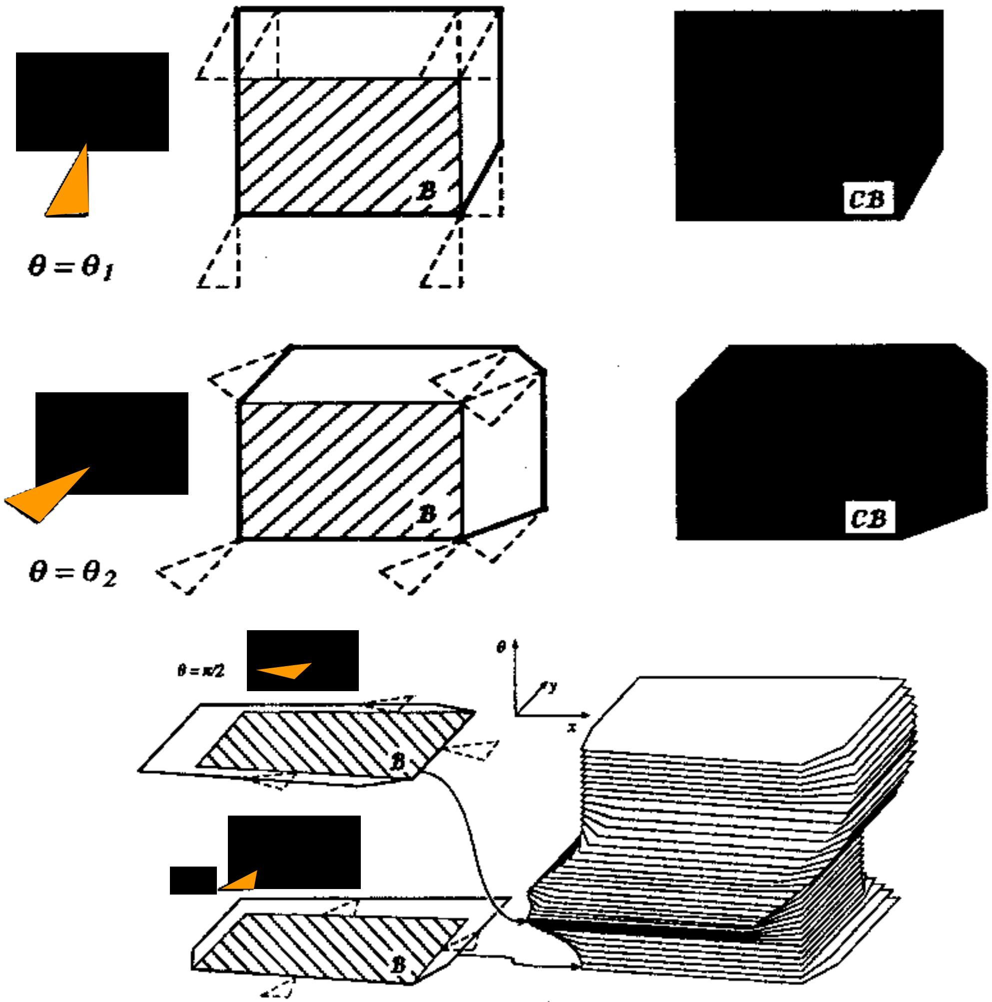


  - Potential field

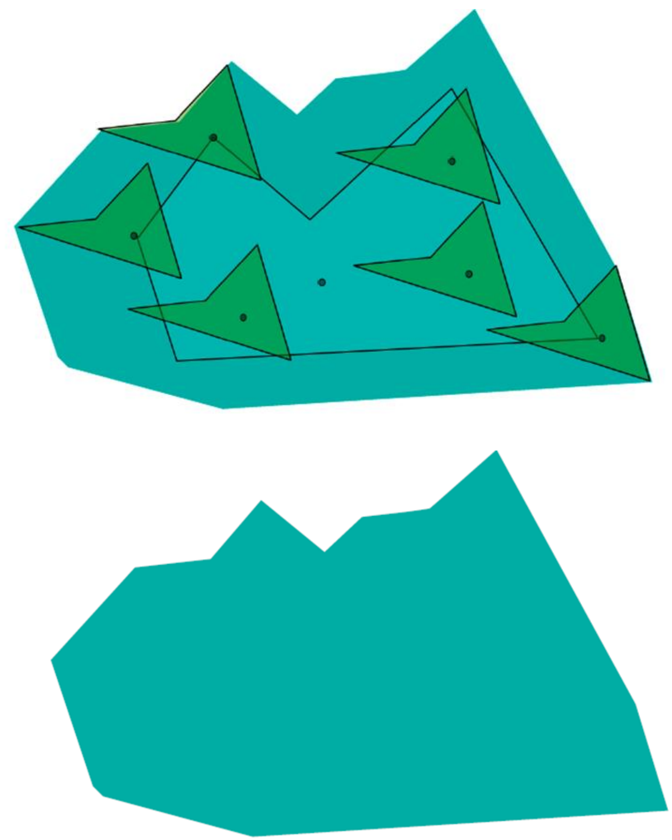




Not a Point Robot? – Configuration Space

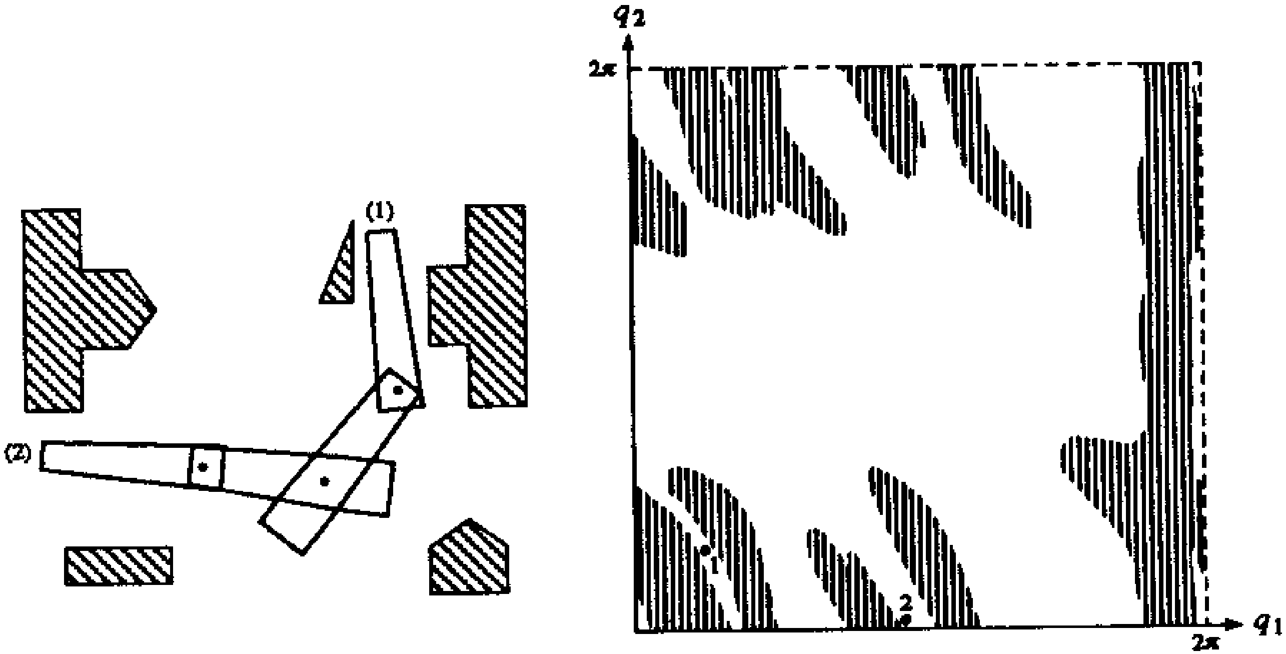


Minkowski Sum



Configuration Space

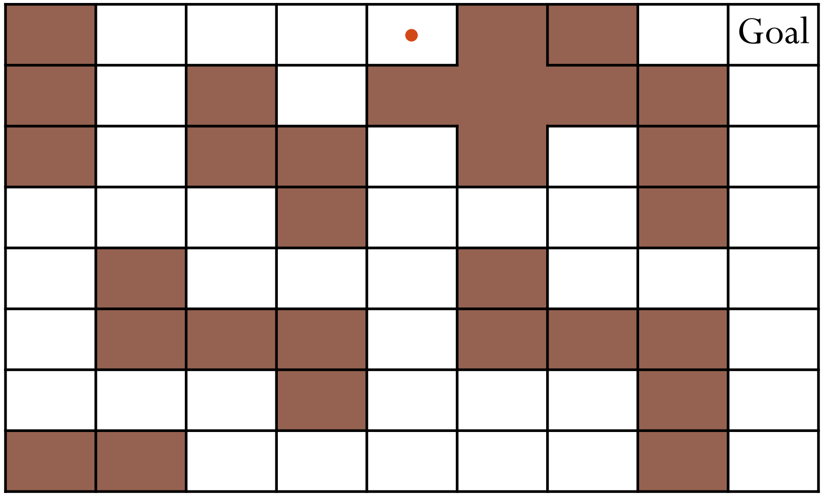
Usually No Analytical Solution
Sampling

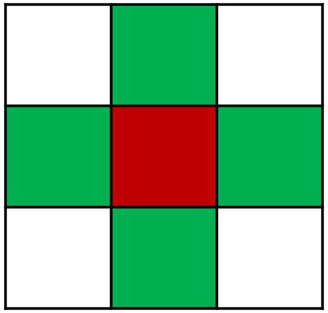


How to compute C_{obs} for articulated bodies?

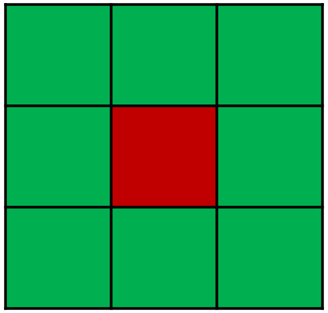
Discrete Search Problem Formulation

1. State Space
   - The space of cells, usually in x,y coordinates
2. Successor Function
   - A cell's successors are its neighbors
   - 4 connected vs. 8 connected
3. Actions
   - Move to a neighboring cell
4. Cost Function
   - Distance between cells traversed
   - Are costs the same for 4 vs 8 connected?
5. Goal Test
   - Check if at goal cell
   - Multiple cells can be marked as goals





4-connected



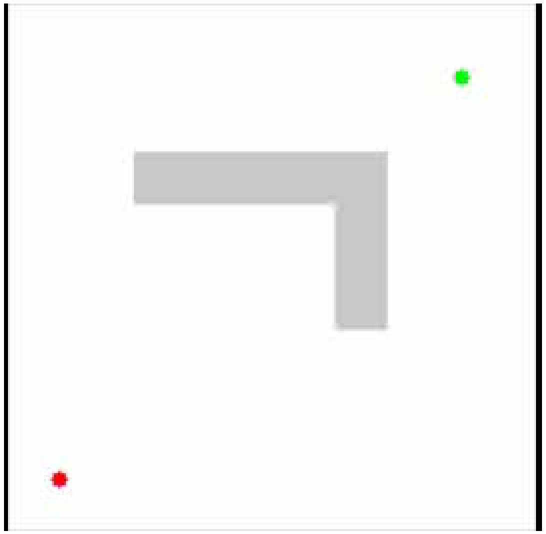
8-connected

A* Search

- Main idea: Select nodes based on cost-to-come and heuristic:

$$f(x) = g(x) + h(x)$$

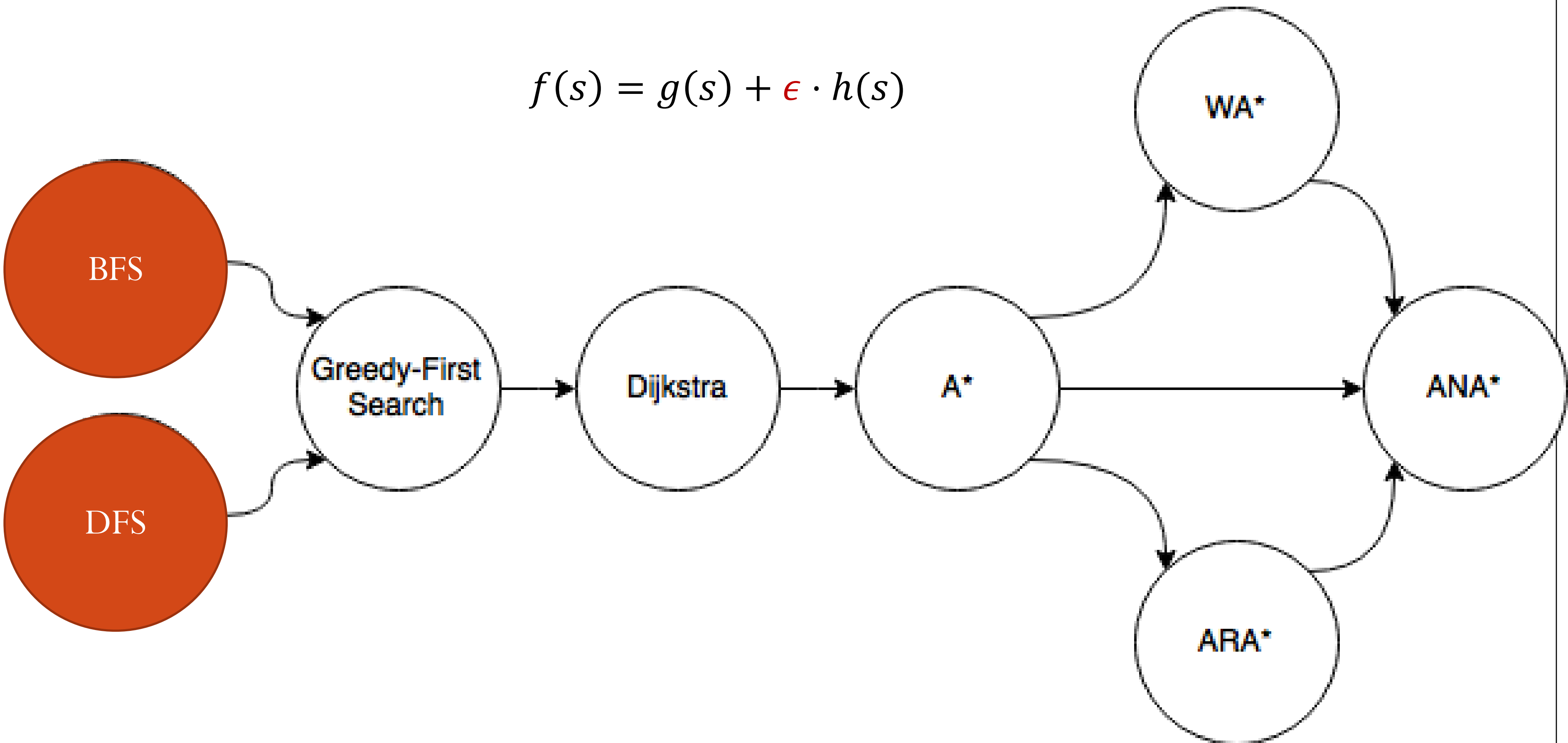
- Open list is a *priority queue*, nodes are sorted according to f(x)
- g(x) is sum of edge costs from root node to x



- This algorithm is the basis for MANY variants (D*, ARA*, ANA*, etc.)

Other Search Algorithms

$$f(s) = g(s) + \epsilon \cdot h(s)$$



From Presentation by **Abhishek Kulkarni**: Anytime Non-Parametric A*

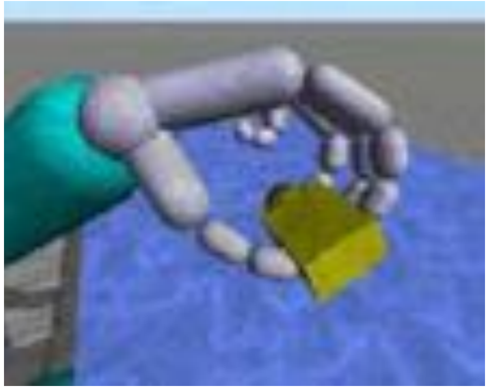
Representations of Rotation

- **Euler angles** are simplest but have singularity problems
  - Besides you usually convert to rotation matrices to actually use them
- **Quaternions** are beautiful math but can be annoying to manipulate
  - But, quaternions are perfect for some problems. For example: sampling 3D rotations
- Use **Homogenous Transforms** whenever possible.
  - Easy to compose and manipulate (just linear algebra)
  - Rotation and translation in one consistent package
  - Not super-simple, but can "read" the coordinate axes by looking at columns
- Many software frameworks provide methods to convert between these representations (ROS, openrave)

Collision Checking

- Many representations, most popular for motion planning are
  - Triangle meshes
  - Composites of primitives (box, cylinder, sphere)
  - Voxel grids

Triangle meshes



Composite of primitives

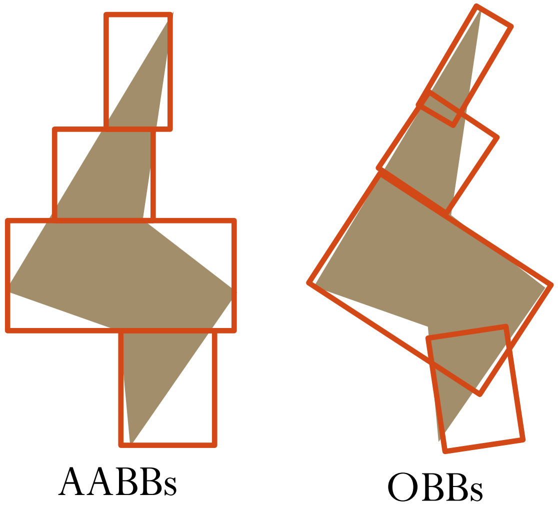
Voxel

Collision Checking

- How to make it faster:

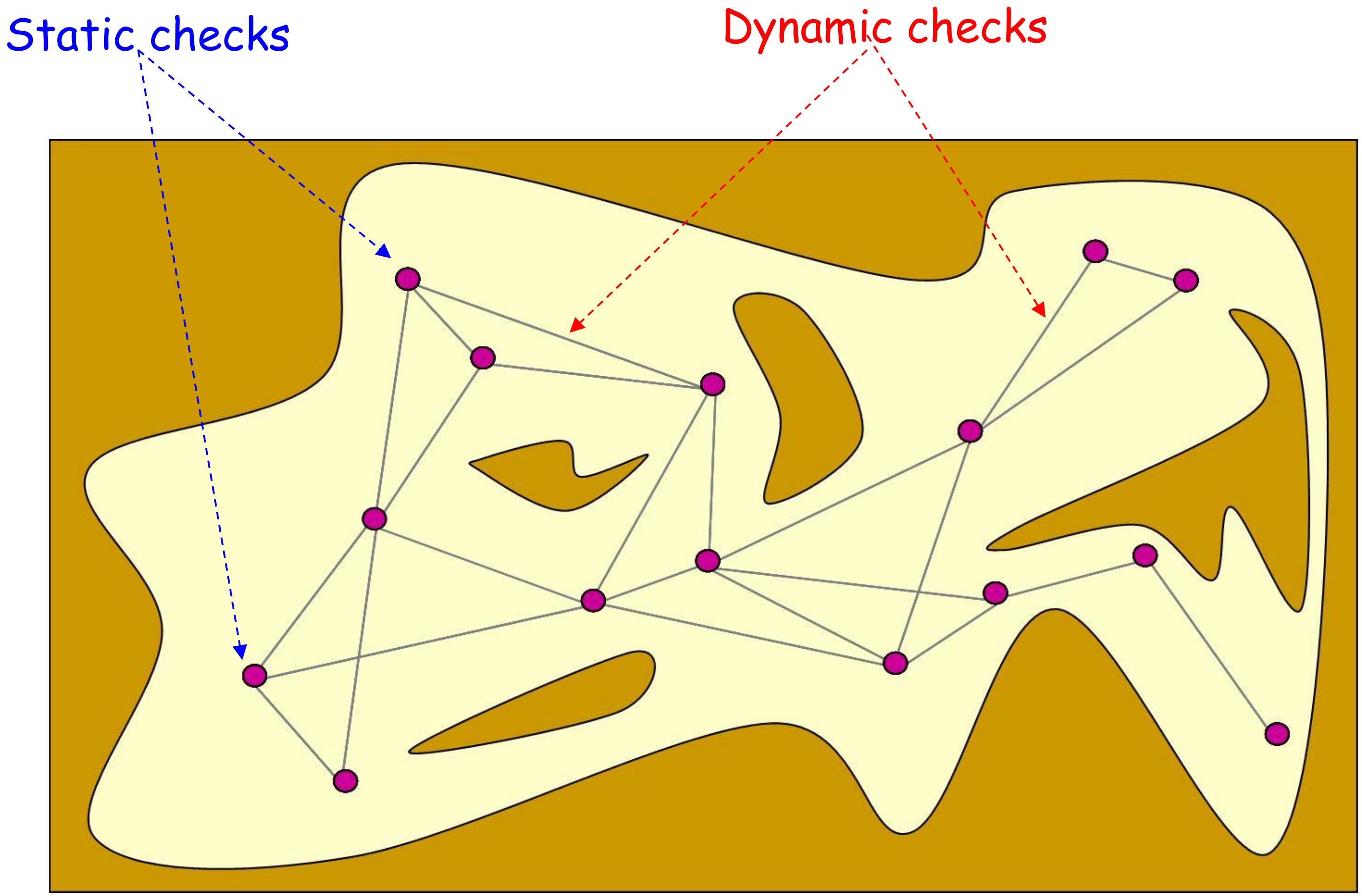
1. Simplify your meshes

2. Use Bounding Volumes



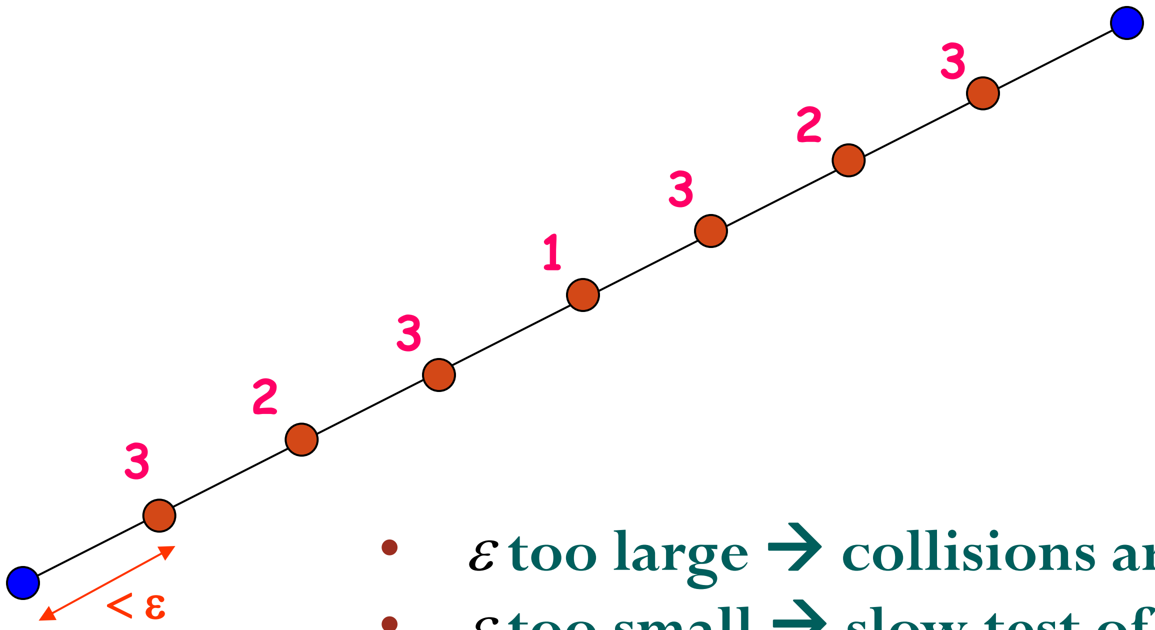
Hierarchical Bounding Volumes

Static vs. Dynamic Collision Detection



Usual Approach to Dynamic Checking

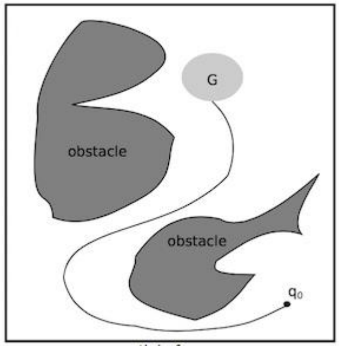
1) Discretize path at some fine resolution ε
2) Test statically each intermediate configuration

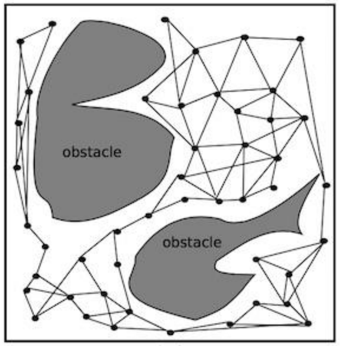


- *ε* **too large → collisions are missed**
- *ε* **too small → slow test of local paths**

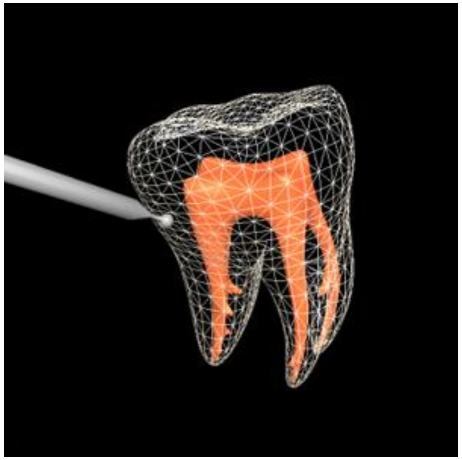
Testing Path Segment vs. Finding First Collision

- PRM planning
  - Detect collision as quickly as possible → Bisection strategy

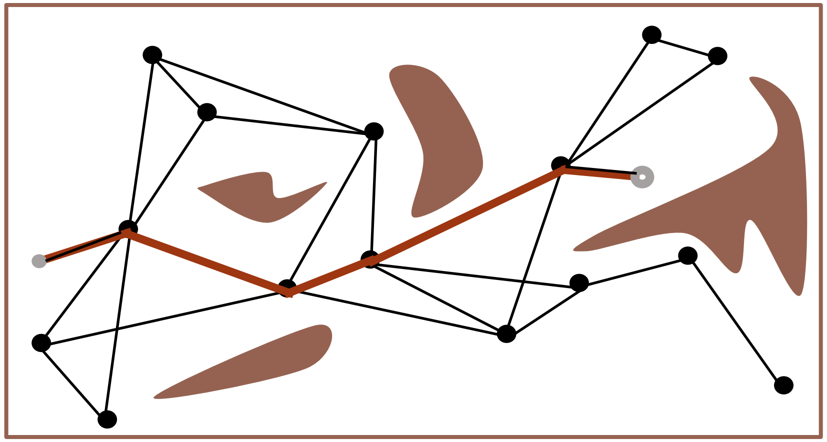




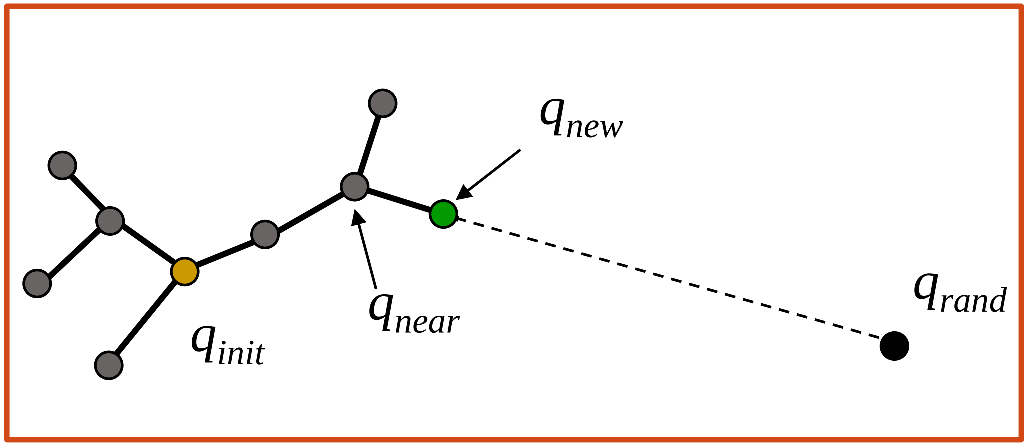
- Physical simulation, haptic interaction
  - Find first collision → Sequential strategy



Sampling-based Planning



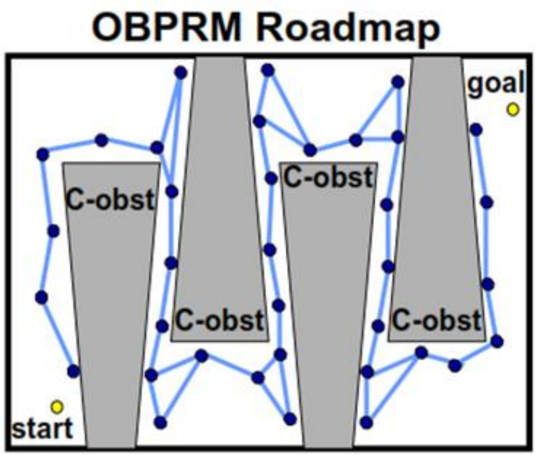
PRM



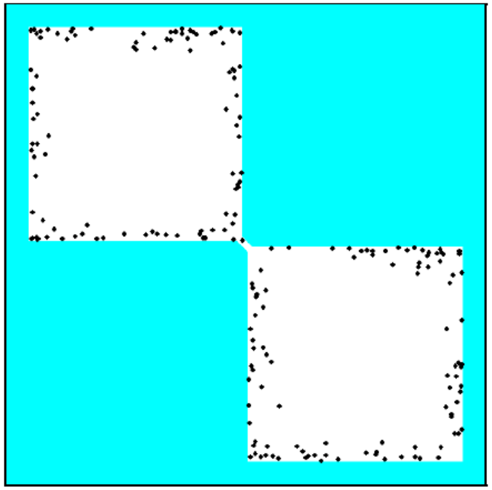
RRT

- **The good**:
  - Provides fast *feasible* solution
  - Popular methods have few parameters
  - Works on practical problems
  - Works in high-dimensions
  - Works even with the wrong distance metric
- **The bad**:
  - No quality guarantees on paths*
    - In practice: smooth/optimize path afterwards
  - No termination when there is no solution
    - In practice: set an arbitrary timeout
  - Probabilistic completeness is a weak property
    - Completeness in high-dimensions is impractical

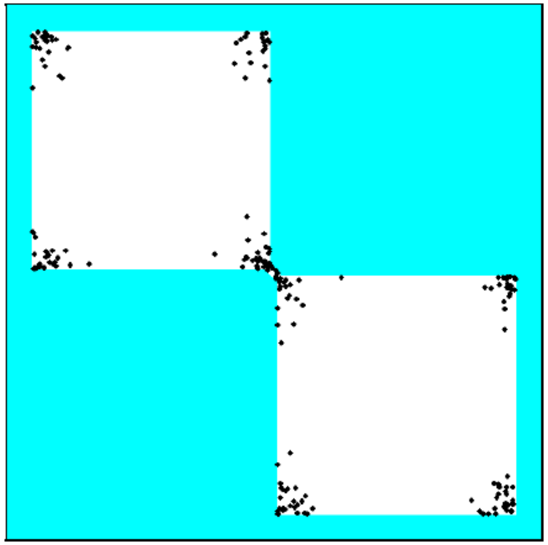
Sampling Strategies



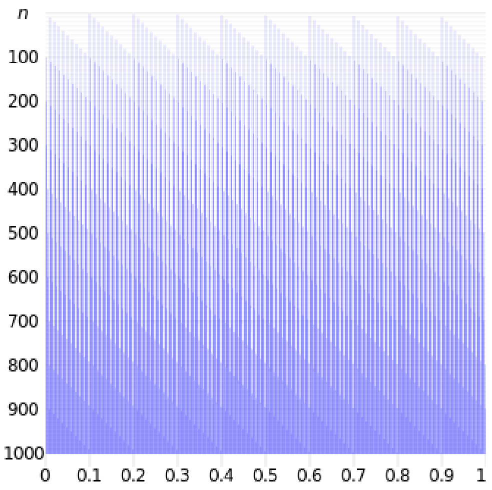
Obstacle-based PRM



Gaussian Sampling

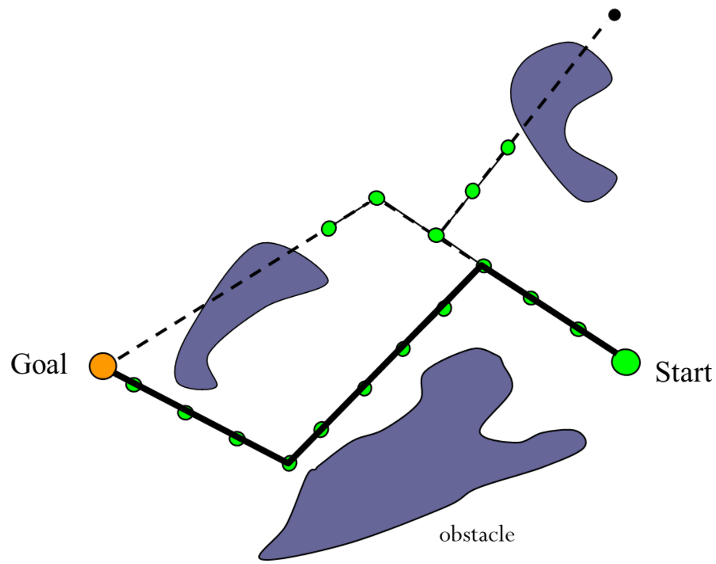


Bridge Sampling

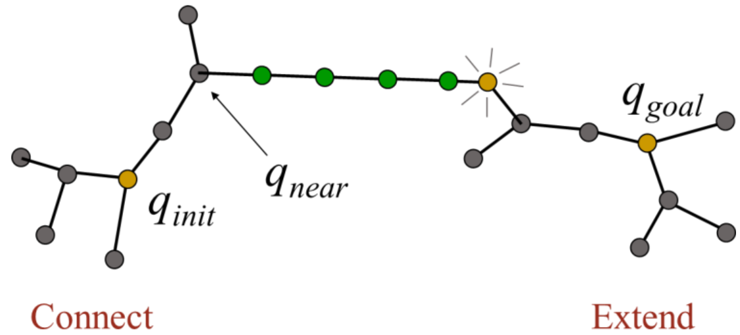


quasi-random sampling

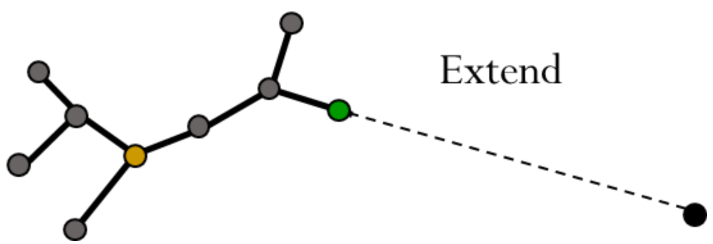
Variants of RRT

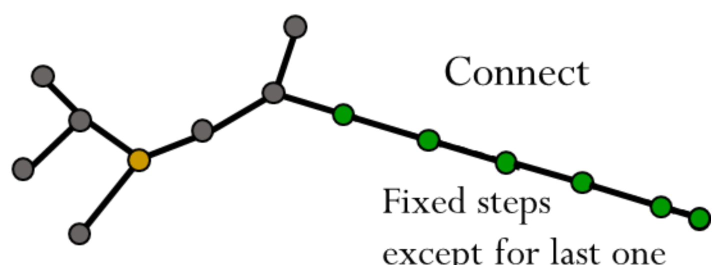


RRT with Obstacles and Goal Bias

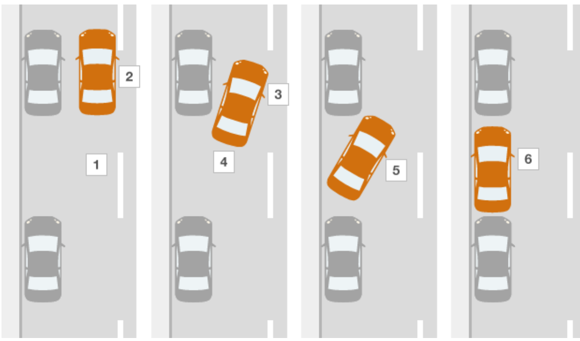


BiDirectional RRT

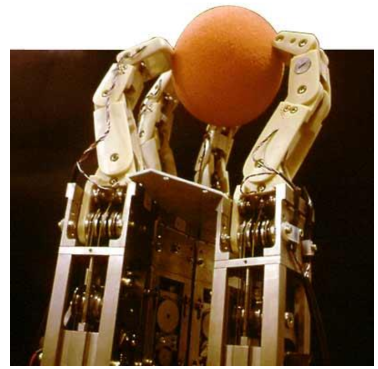




Non-holonomic Constraints

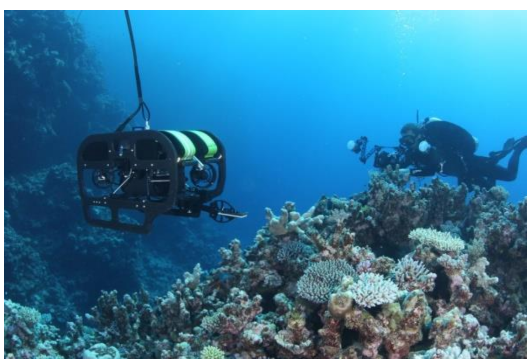


Parallel Parking



Manipulation with a robotic hand

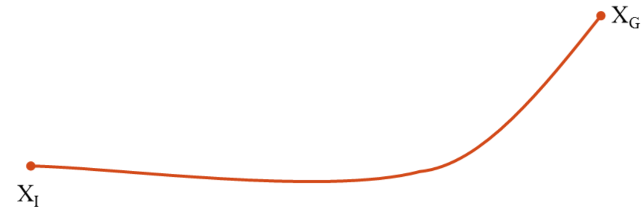
Untethered space robots



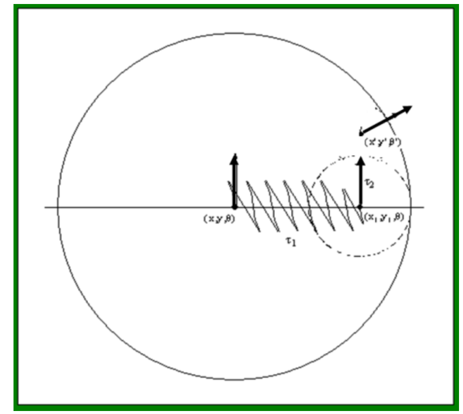
Underwater robot
Forward propulsion is allowed only in the pointing direction

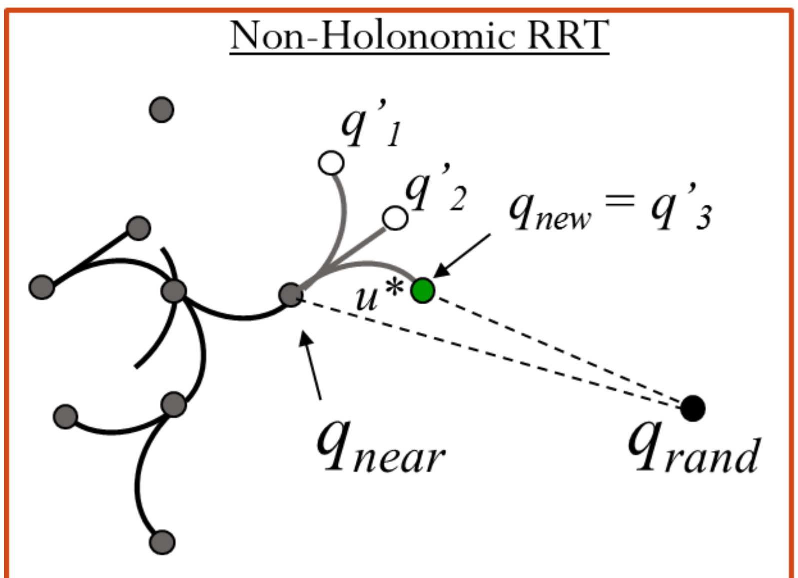
- **Rolling without contact**
- **Conservation of angular momentum**
- **A Chosen actuation strategy**

How to Plan?



- Exact methods
  - Two-Point Boundary Value Problem (BVP)
  - Apply sequence of allowed maneuvers
  - Apply Sequences of Primitives
- Sampling-based methods

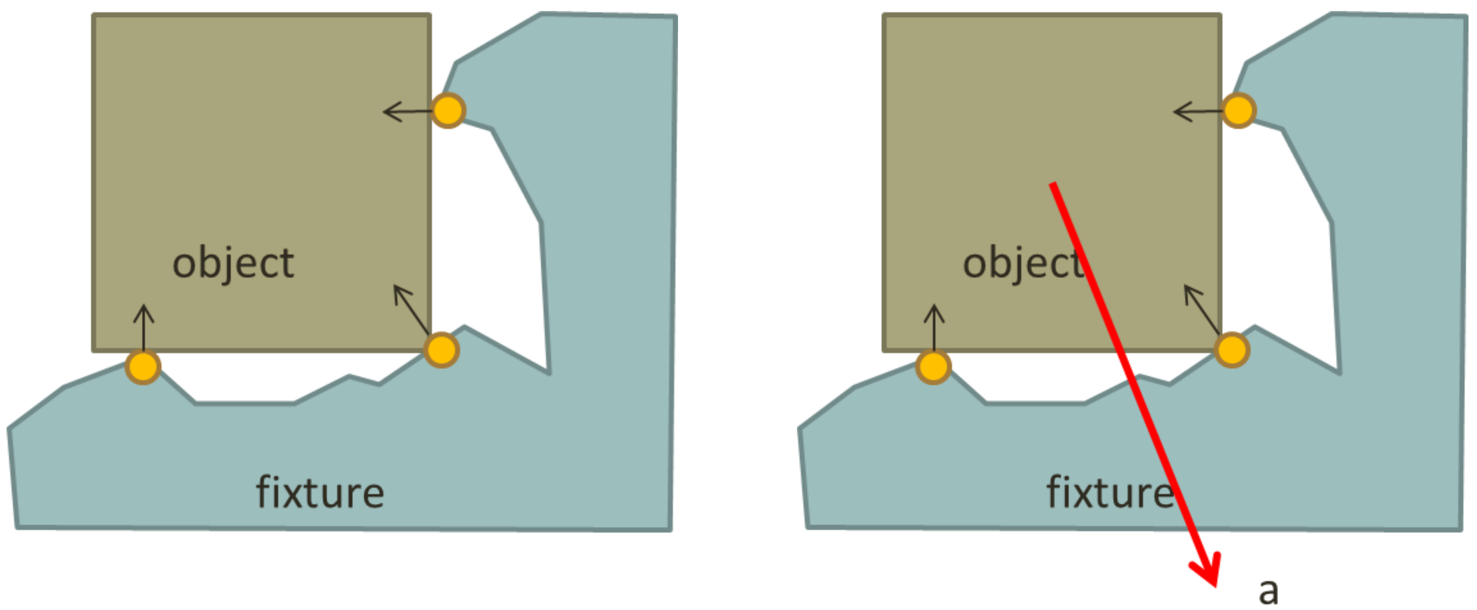




[Knepper and Mason, ICRA 2009]

Grasp Restraint

- Form closure
- Force closure

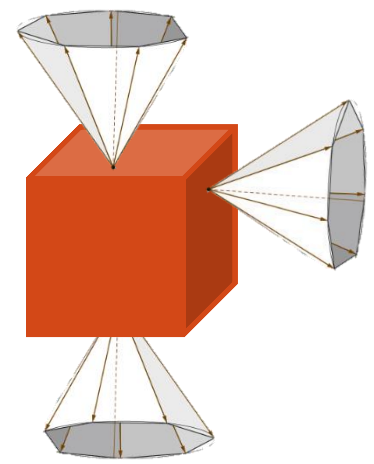


Force-Closure Grasp Metric

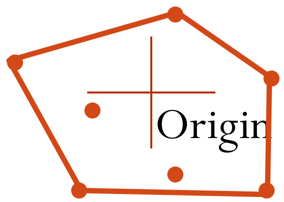
- Many algorithms exist to test for force closure, here is one:

Input: Contact locations
Output: Is the grasp in Force-Closure? (Yes or No)

1. Approximate the friction cone at each contact with a set of wrenches
2. Combine wrenches from all cones into a set of points S in wrench space
3. Compute the *convex hull* of S
4. If the origin is inside the convex hull, return YES. If not, return NO.

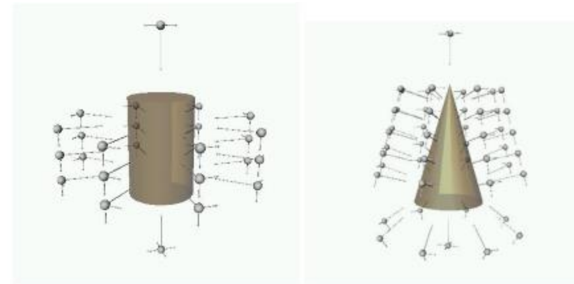


Friction cones at contacts

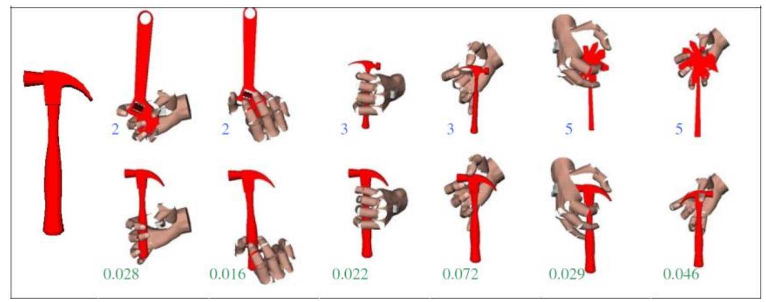


Wrench space
(6 dimensional)

How to Efficiently Plan Grasp?



- Search
  - Contact points on the surface of an object
  - Hand pose relative to object with fingers in a pre-shape → Search Force Closure Grasps
  - Pre-computed grasp sets



- Integrating Grasping and Manipulation Planning
  - Environment Clearance Score
  - Grasp quality
  - Reachability Score

Inverse Kinematics

- If N=M,
  - Jacobian is square → Standard matrix inverse
- If N>M ,
  - Pseudo-Inverse — Minimize $\frac{1}{2}\dot{\theta}^T\dot{\theta}$ given that $\dot{x} = J(\theta)\dot{\theta}$
  - Weighted Pseudo-Inverse

Minimize $\frac{1}{2}||\dot{q}||_w^2 = \frac{1}{2}\dot{q}^T W \dot{q}$ **given that** $\dot{x} = J(\theta)\dot{\theta}$

  - Damped least squares

unconstrained minimization of a **suitable** objective function

$$\min_{\dot{q}} \frac{\mu^2}{2}||\dot{q}||^2 + \frac{1}{2}||\dot{x} - J\dot{q}||^2 = H(\dot{q})$$

compromise between large joint velocity and task accuracy

  - Iterative Jacobian Pseudo-Inverse

Iterative Jacobian Pseudo-Inverse Inverse Kinematics

While true

$X_{current} = FK(q_{current})$

$\dot{x} = (x_{target} - x_{current})$

error = $|| \dot{x} ||$

If error < threshold

return Success

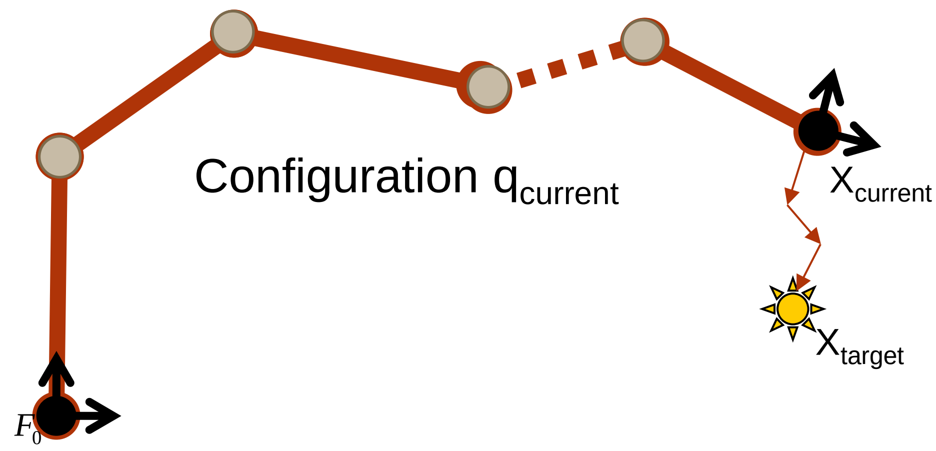
$\dot{q} = J(q)^{+} \dot{x}$

If$(||\dot{q}|| > \alpha)$

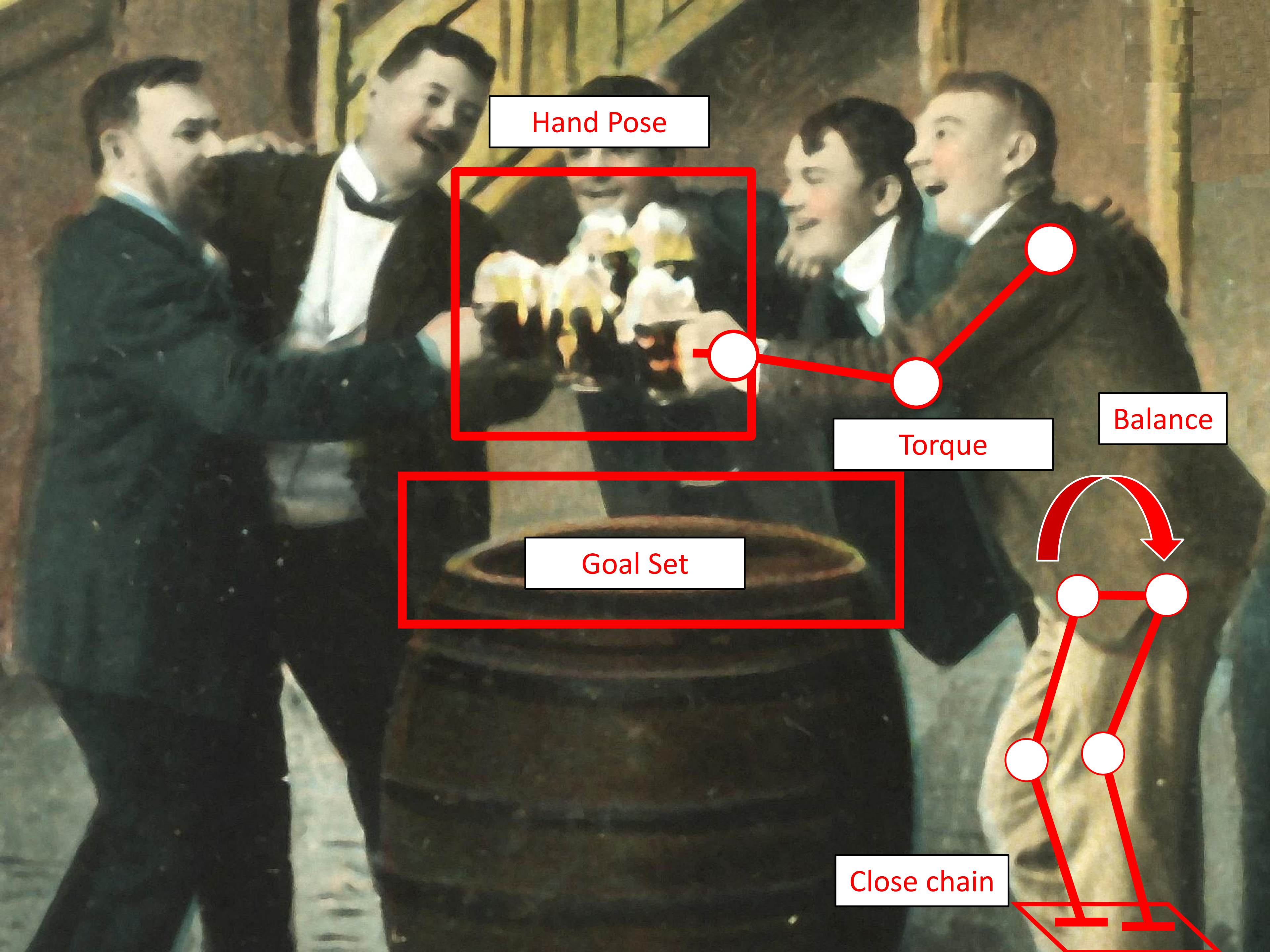
$\dot{q} = \alpha(\dot{q} / ||\dot{q}||)$

$q_{current} = q_{current} - \dot{q}$

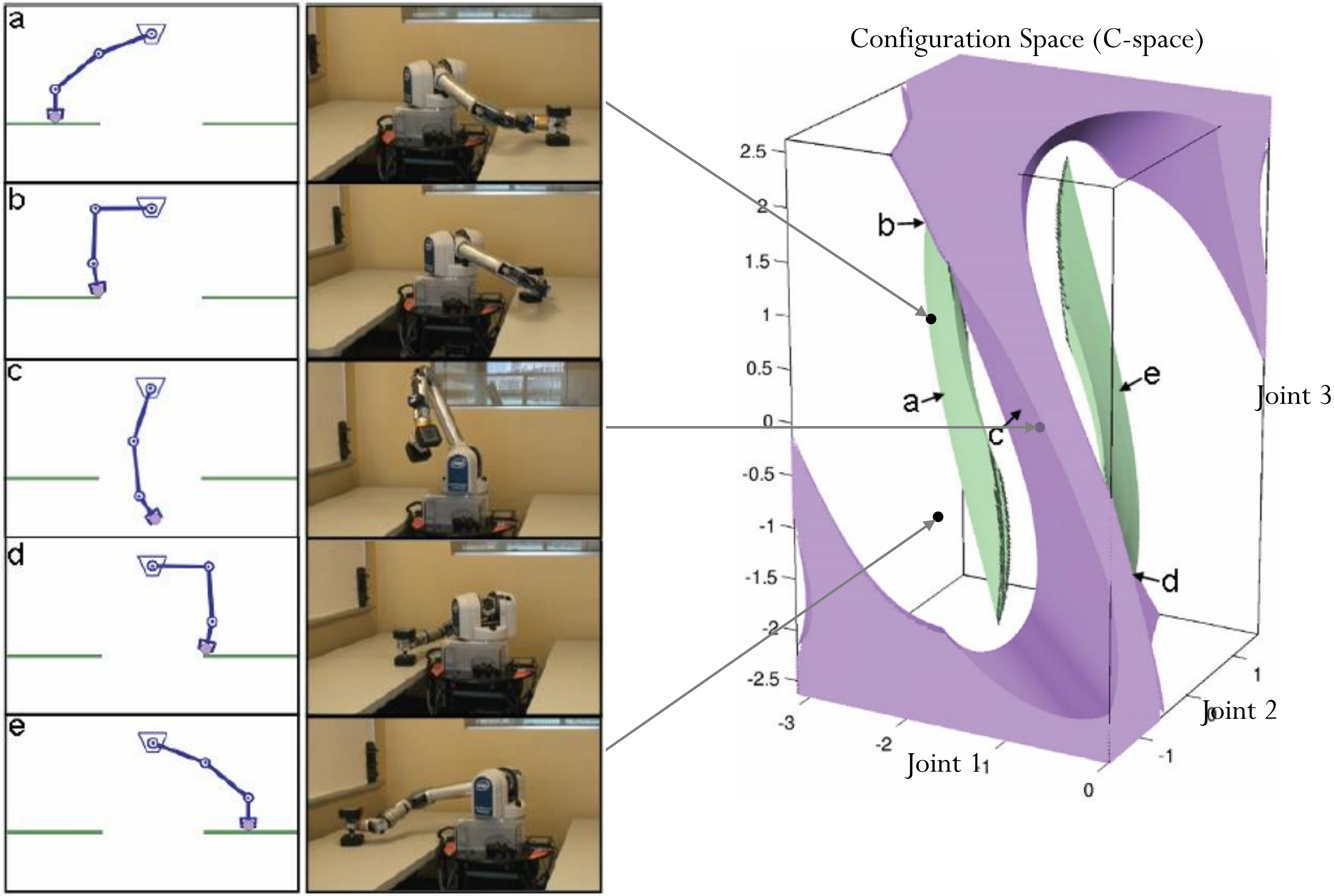
end



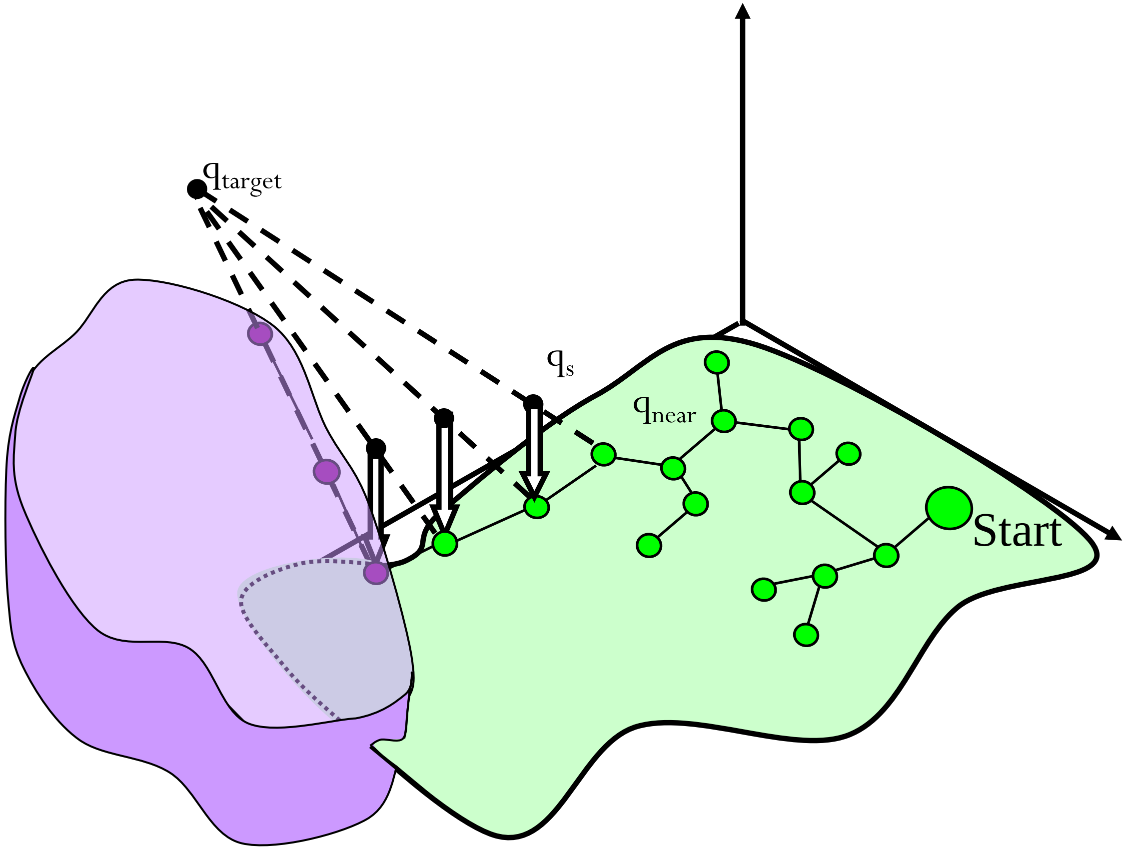
- This is a local method, it will get stuck in local minima (i.e. joint limits)!!!
- α is the step size
- Error handling not shown
- A correction matrix has to be applied to the angular velocity components to map them into the target frame (not shown)


Hand Pose
Torque
Balance
Goal Set
Close chain

Configuration Space (C-space)



Constrained BiDirectional RRT (CBiRRT)



Trajectory Optimization

- Solve this optimization problem:

$$\arg\min_{\tau} C(\tau)$$

trajectory (τ)

$C(\tau)$:
-Uncertainty
-Smoothness
-Distance from obstacles
....

- Subject to these constraints:

$$f(\tau) = 0$$
$$g(\tau) \leq 0$$
$$h(\tau) < 0$$

e.g. pose constraints

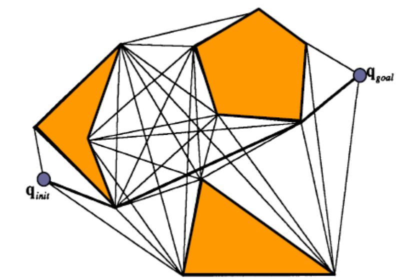
e.g. joint limits

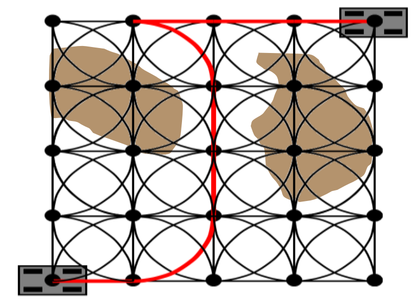
e.g. collision constraints

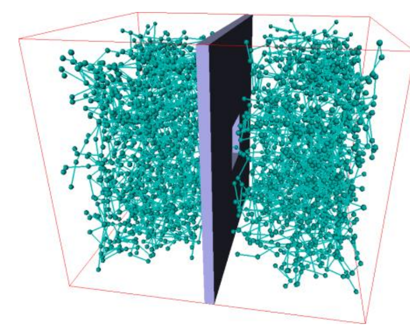
- An active research area with many methods:
  1. Put constraints in cost function and do gradient descent
  2. Linear/quadratic/convex programming
  3. Combine 1 and 2
  4. Sample in trajectory space, compute gradient from samples
- Trajectory optimizers often have trouble with complex obstacles and local minima

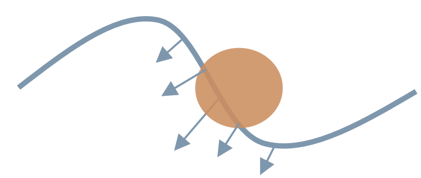
Planner Types

- Exact algorithms
  - Either find a solution or prove none exists
  - Very computationally expensive
  - Unsuitable for high-dimensional spaces
- Discrete Search
  - Divide space into a grid, use A* to search
  - Good for vehicle planning
  - Unsuitable for high-dimensional spaces
- Sampling-based Planning
  - Sample the C-space, construct path from samples
  - Good for high-dimensional spaces
  - Weak completeness and optimality guarantees
- Trajectory Optimization
  - Fast local planning using optimization algorithms
  - Can converge to infeasible local minima when feasibility constraints are difficult



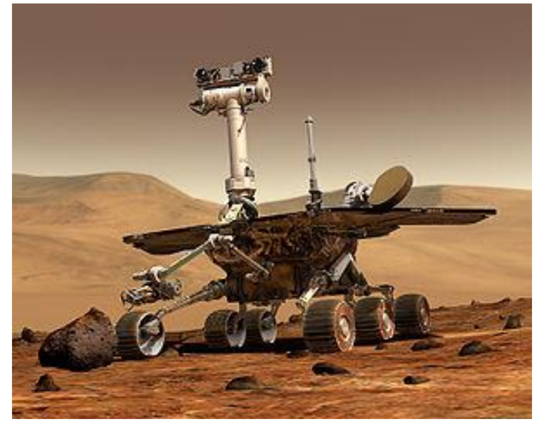






What planning algorithms should we use?

Roomba iCreate



Mars Rovers

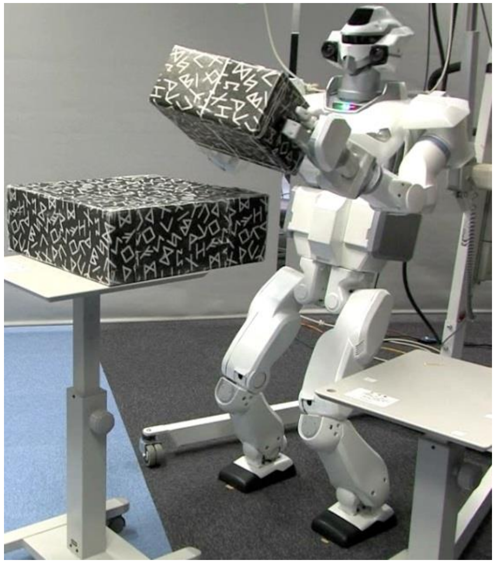
DARPA Urban Challenge

Google Self-Driving Car

What planning algorithms should we use?

Factory Automation

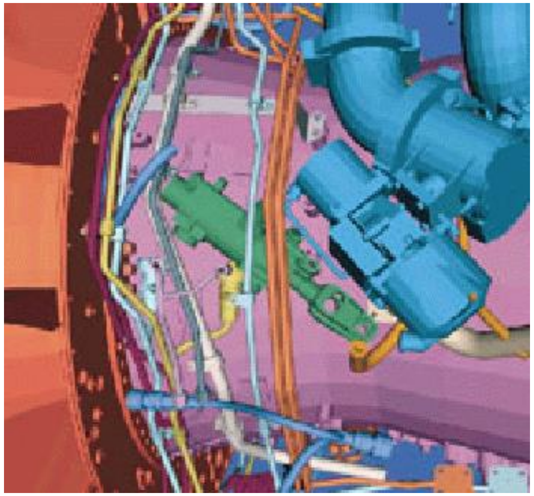
Rigid Object Manipulation

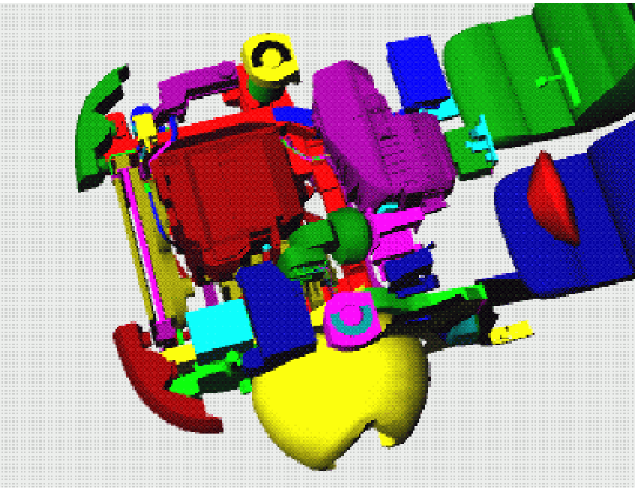


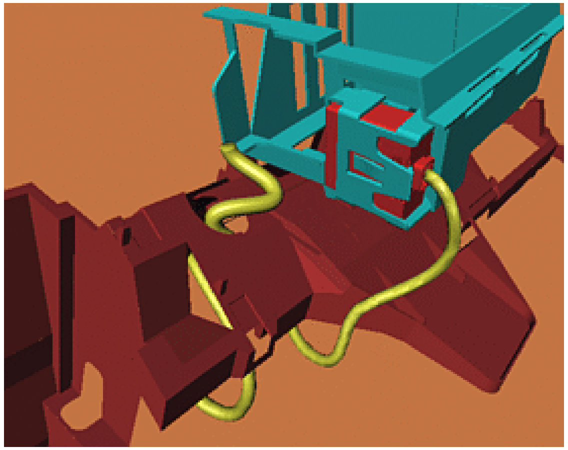
Humanoid Manipulation

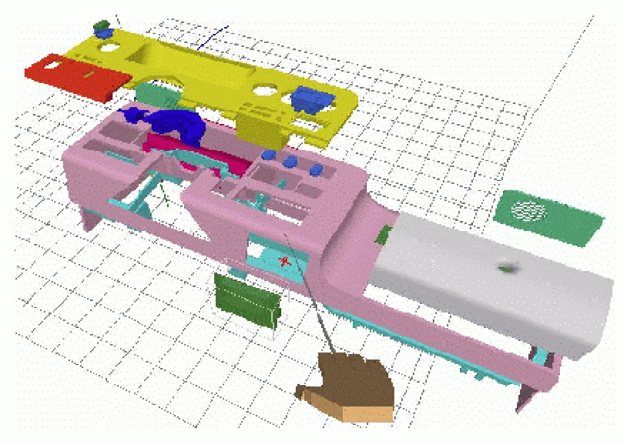
Deformable Object Manipulation

What planning algorithms should we use for assembly/disassembly planning?

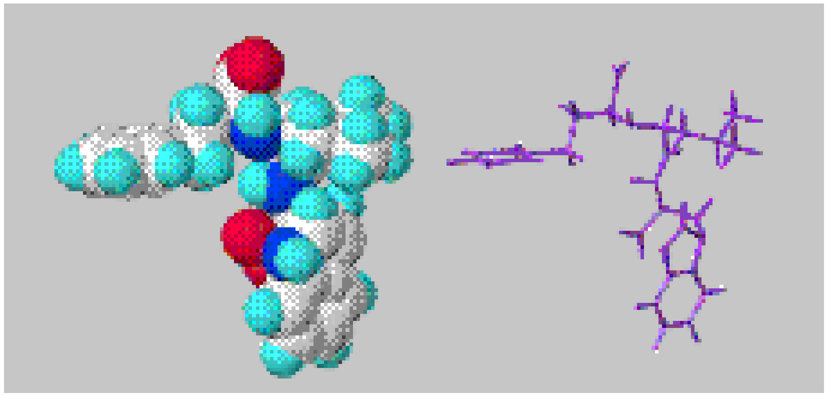


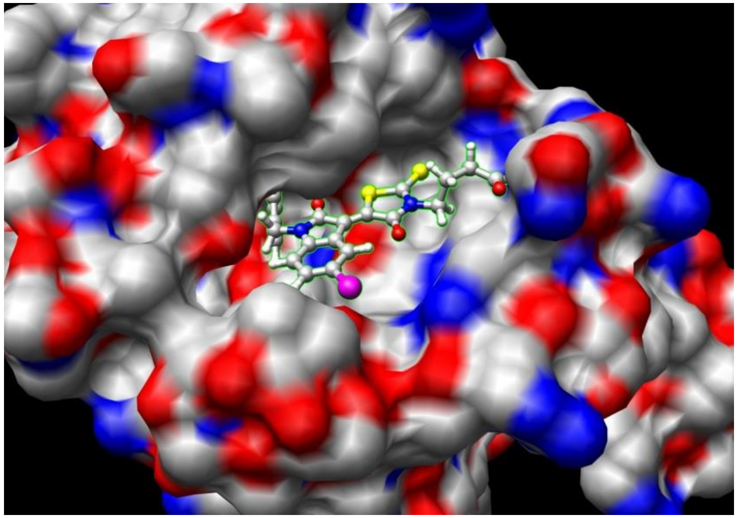


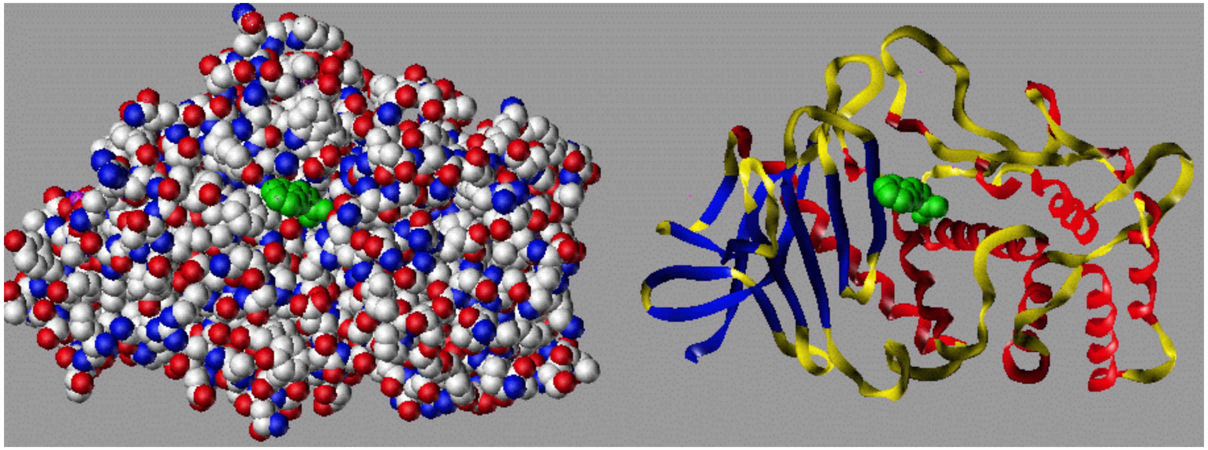




What planning algorithms should we use for protein docking?







What problems would you like to solve?

Integrating sensing

Manipulation of deformable objects

Dynamic constraints

Planning with uncertainty

Efficient optimal planning in high dimensions

Collaborating with humans

Moving obstacles