Last update: May 6, 2010

Robotics

CMSC 421: Chapter 25

What is a robot?

A machine to perform tasks

◇ Some level of autonomy and flexibility, in some type of environment

◇ Sensory-motor functions
  - Locomotion on wheels, legs, or wings
  - Manipulation with mechanical arms, grippers, and hands

◇ Communication and information-processing capabilities
  - Localization with odomoters, sonars, lasers, inertial sensors, GPS, etc.
  - Scene analysis and environment modeling with a stereovision system on a pan-and-tilt platform

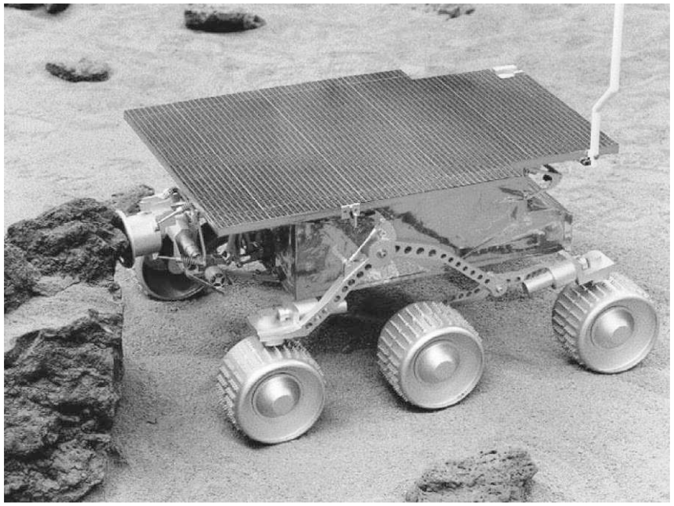
Reasonably mature technology when robots restricted to either

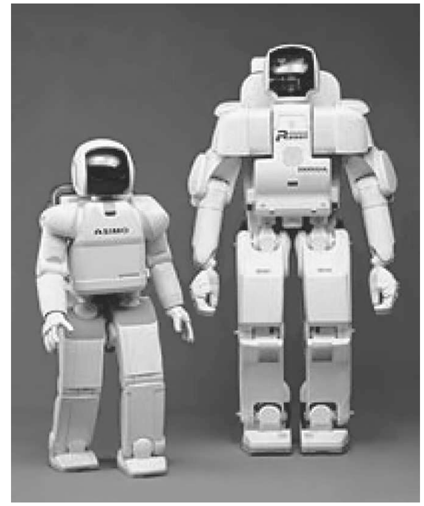
◇ well-known, well-engineered environments (e.g., manufacturing)

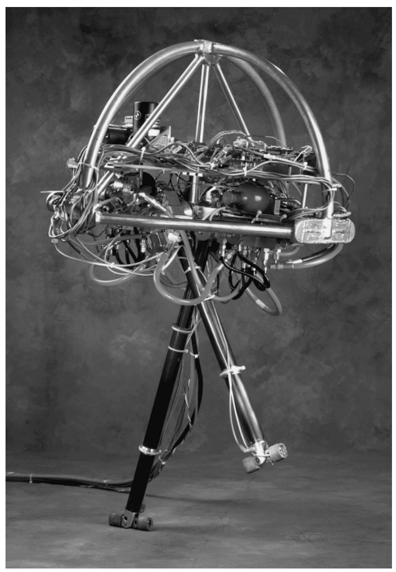
◇ performing single simple tasks (e.g., vacuum cleaning, lawn mowing)

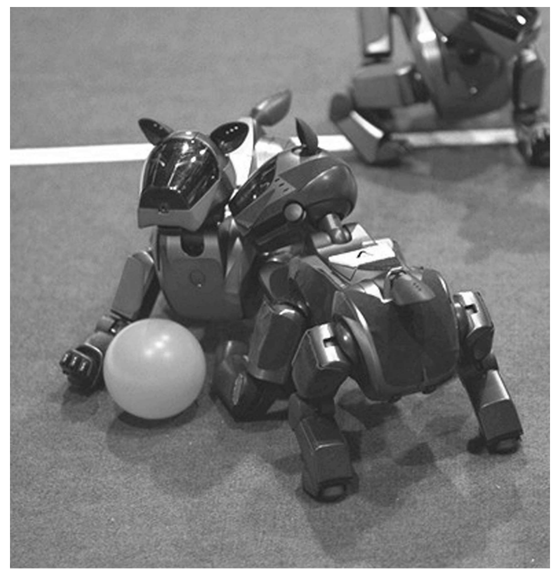
For more diverse tasks and open-ended environments, robotics remains a very active research field

Examples of Mobile Robots

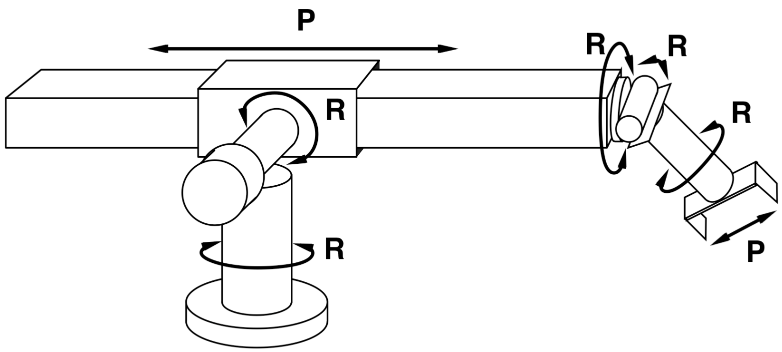








Examples of Manipulators



Examples of Sensors

Range finders: sonar (land, underwater), laser range finder, radar (aircraft), tactile sensors, GPS

Imaging sensors: cameras (visual, infrared)

Proprioceptive sensors:
shaft decoders (joints, wheels),
inertial sensors, force sensors, torque sensors

Hand-coding of robot controllers

Manual development of a robot controller for a specific task

To do hand-coding reliably and inexpensively, need

- ♢ well-structured, stable environment
- ♢ restrictions on the scope and diversity of the robot's tasks
- ♢ only a limited human-robot interaction

Developing the reactive controller

- ♢ Devices to memorize motion of a pantomime
- ♢ Graphical programming interfaces

Automated robot controllers

Integrate planning, acting, sensing, learning
Nead to deal with

- ♢ heterogeneous partial models of the environment and of the robot
- ♢ information acquired through sensors and communication channels
- ♢ noisy and partial knowledge of the state

Specialized algorithms for different types of tasks:

- ♢ Path and motion planning
- ♢ Perception
- ♢ Navigation
- ♢ Motor control

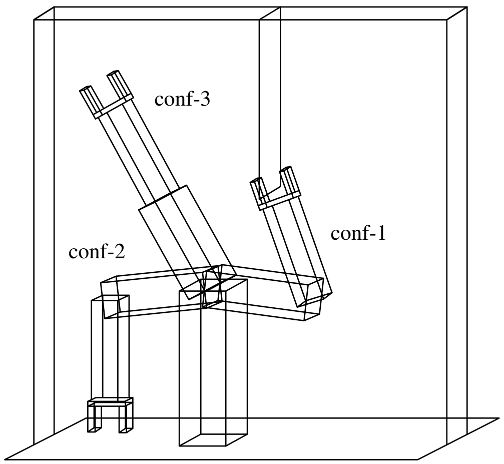
Path Planning

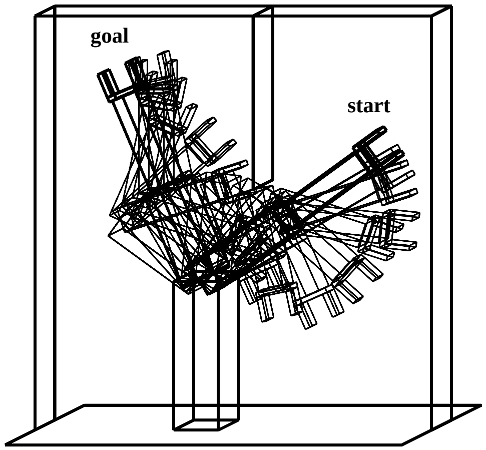
Path planning: find geometric path from starting position to goal position

◇ Input: geometric model of the environment (obstacles, free space)

◇ Solution path must avoid collision with obstacles

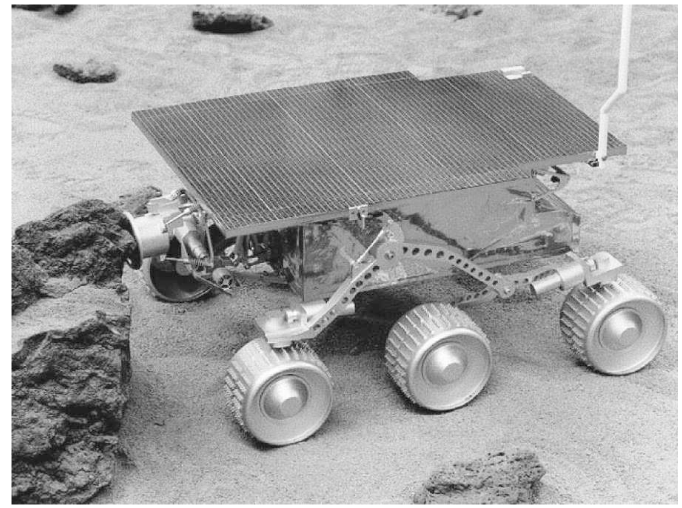
- must also satisfy the robot's *kinematic* (movement) constraints





Motion Planning

Motion planning: find a trajectory that's feasible in both *space* and *time*



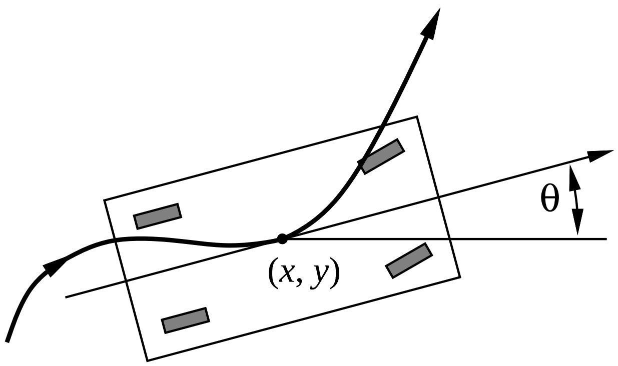
- ◇ Need a feasible path (relies on path planning)
- ◇ Also need a control policy that satisfies the robot's speed and acceleration constraints

Technology for path planning and motion planning is relatively mature

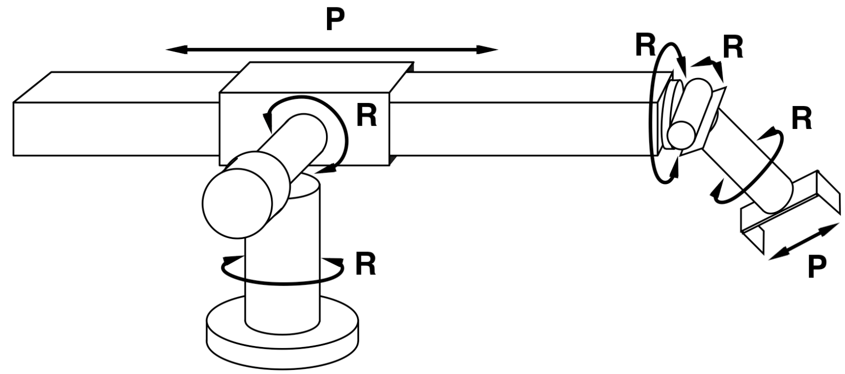
- ◇ Deployed in areas such as CAD and computer animation
- ◇ Computational geometry and probabilistic algorithms

Configuration parameters

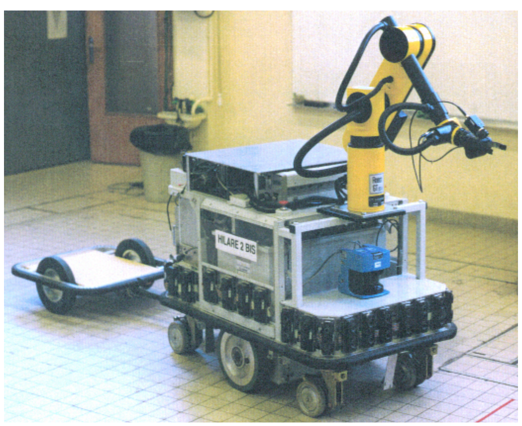
Configuration parameters: the numbers that specify the robot's current state



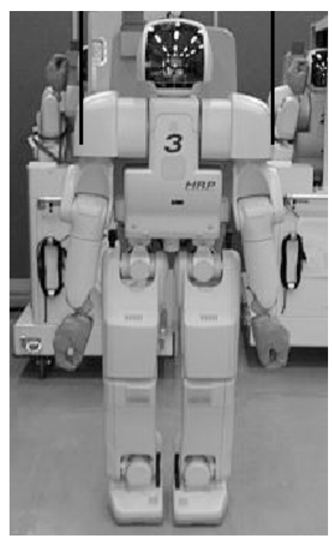
Three parameters: x, y, θ



Seven parameters



10 parameters: 6 for arm,
4 for platform & trailer

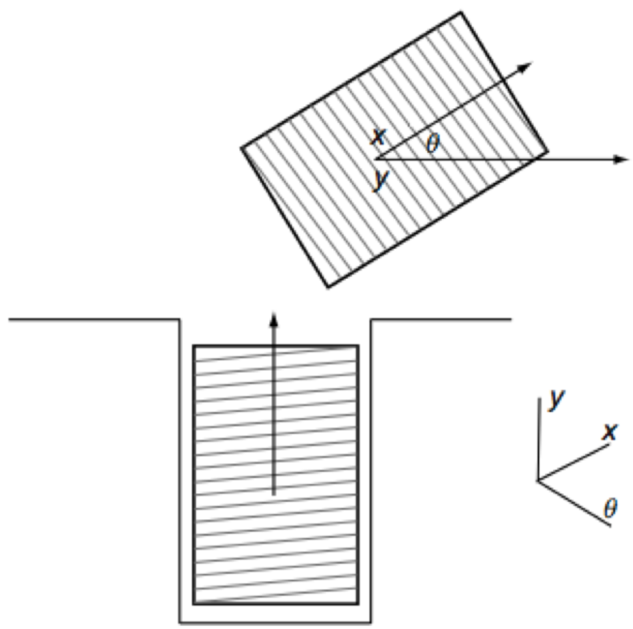


52 parameters:
- 2 for the head
- 7 for each arm
- 6 for each leg
- 12 for each hand

Configuration parameters

q = the *configuration* of the robot = an n-tuple of reals
CS = the robot's *configuration space* = $\{$all possible values for $q\}$

The configuration parameters aren't independent
E.g., can't change θ without changing x and y

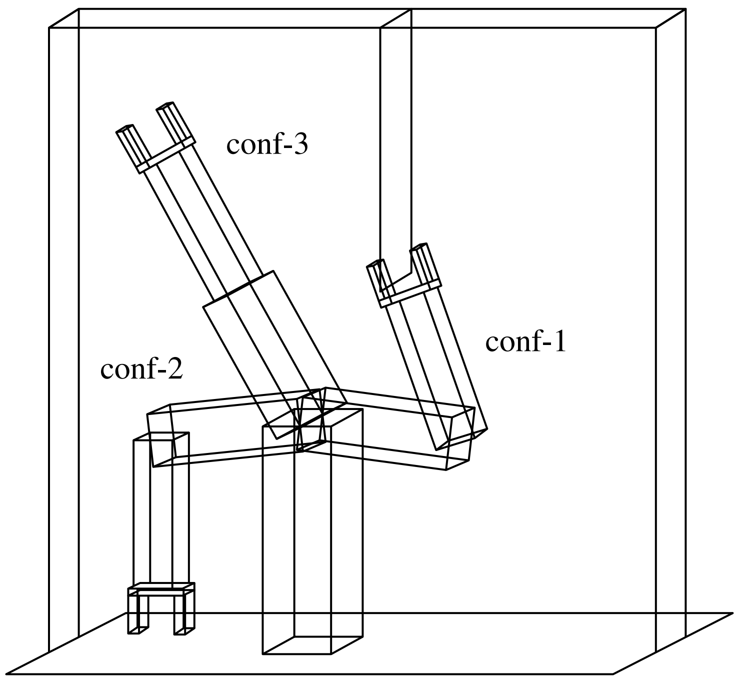


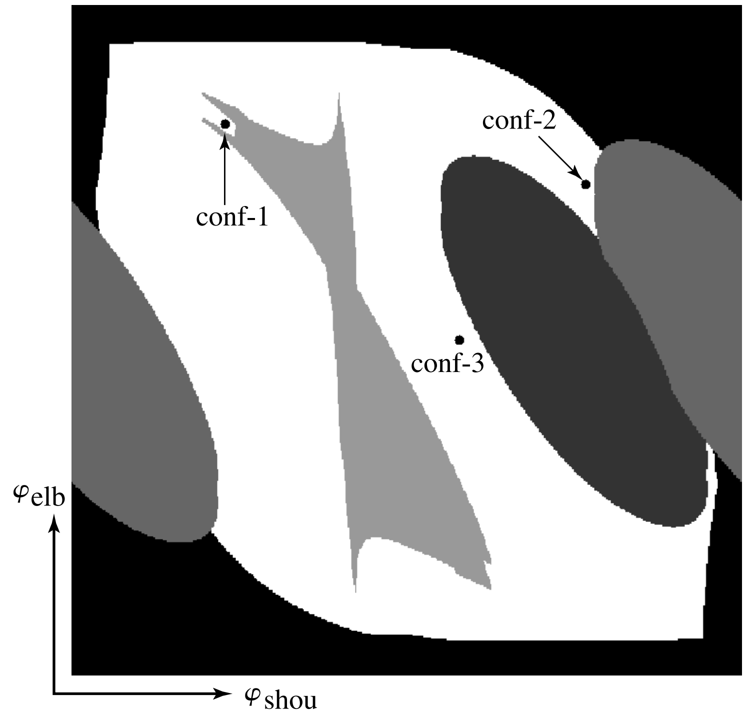
CSfree = *free* configuration space (configs that don't collide with obstacles)
CSfree can be quite complicated

Motion Planning

In ordinary geometric space, the robot occupies a region
In configuration space, it occupies a **point**

Idea: do path planning in configuration space
  Find a path in CSfree from an initial config q_i to a final config q_g





Dealing with the configuration space

n-dimensional space, where n = number of configuration parameters
Each parameter is a real number $\Rightarrow$ ∞^n possible states

For state-space search, convert to a finite state space.

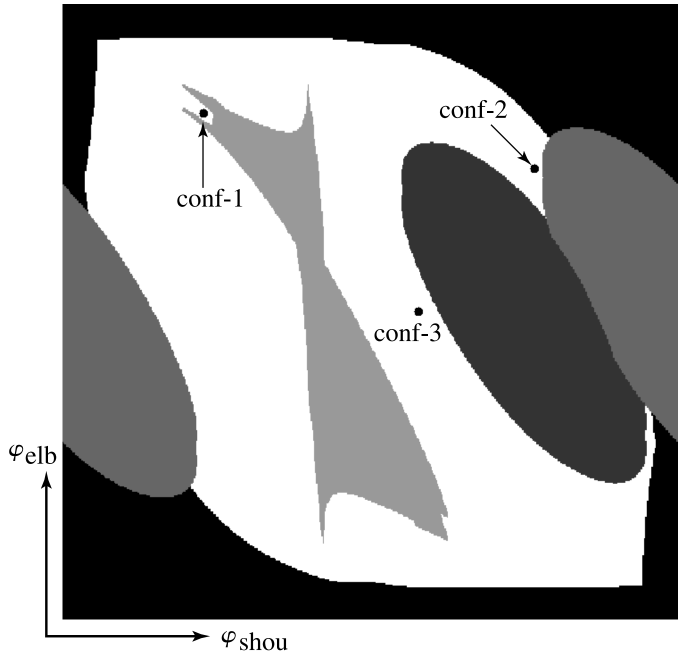
Cell decomposition:

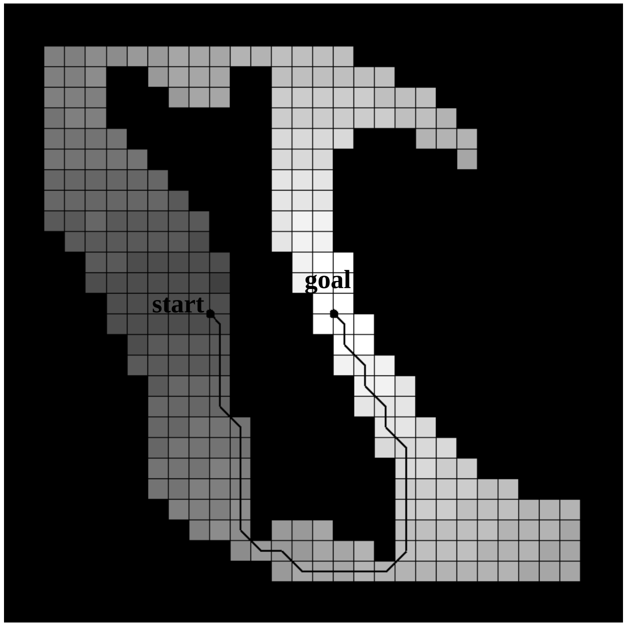
◇ Divide up space into simple *cells*, such that each of them can be traversed "easily" (e.g., convex)

◇ Find a path through the *pure freespace* cells (the ones that don't contain any part of an obstacle)

Skeletonization: identify a finite number points/lines that form a graph
Want a graph such that any two points can easily be connected by following a path on the graph

Cell decomposition example





How many cells, how large?

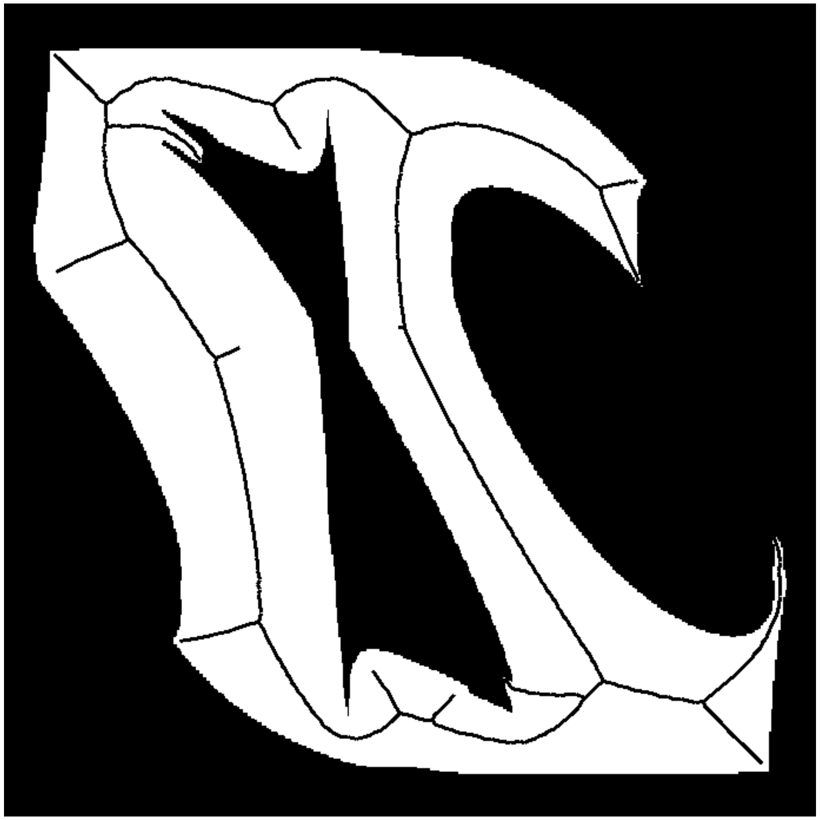
♢ Large number of small cells ⇒ large computation time

♢ Small number of large cells ⇒ no path through pure freespace

Solution: recursively decompose mixed (free+obstacle) cells into smaller cells

♢ quadtrees

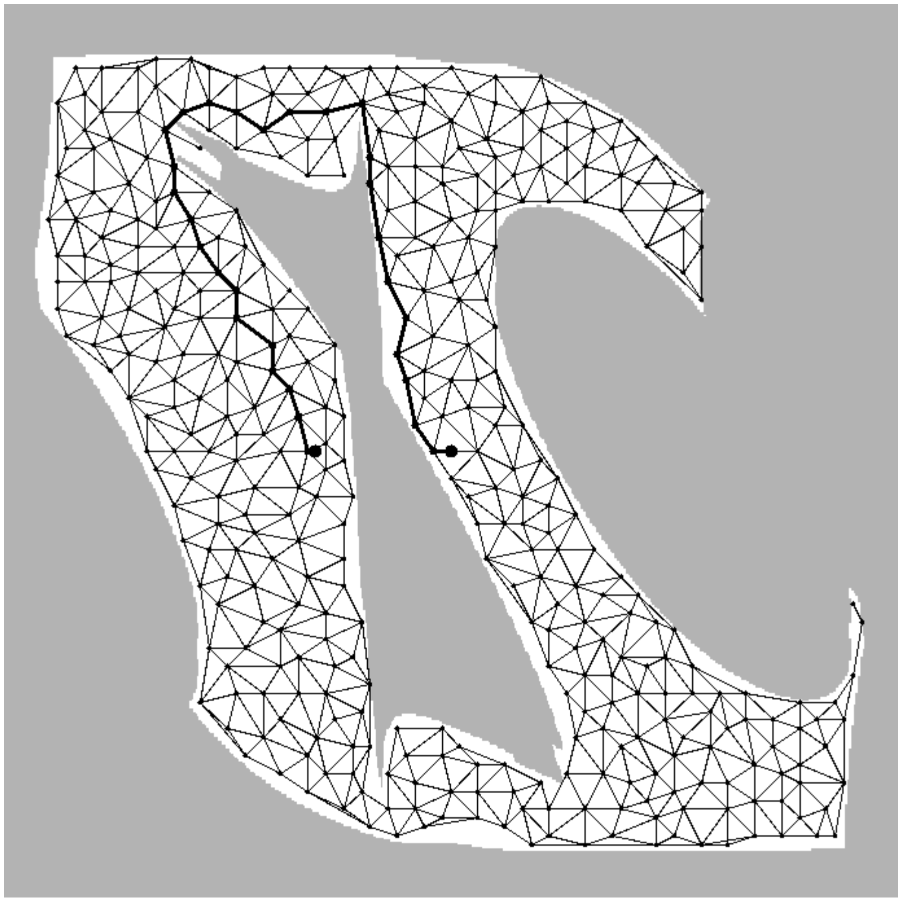
Skeletonization: Voronoi diagram



Voronoi diagram: locus of points equidistant from obstacles

Problem: doesn't scale well to higher dimensions

Skeletonization: Probabilistic Roadmap



Probabilistic roadmap R:

1. generate random points in CS, and keep the ones in CSfree
2. create graph by joining each adjacent pair p_1, p_2 by a line $L(p_1, p_2)$

To keep things simple, we'll use straight lines

More generally, the lines might be curved, in order to satisfy the robot's kinematic constraints

R is *adequate* if it contains enough points to ensure that every start/goal pair is connected through the graph

Path planning with roadmaps

Given an adequate roadmap for CSfree and two configurations q_i and q_g in CSfree, a feasible path from q_i to q_g can be found as follows:

◇ Find configuration q_i' in R such that $L(q_i, q_i')$ is in CSfree

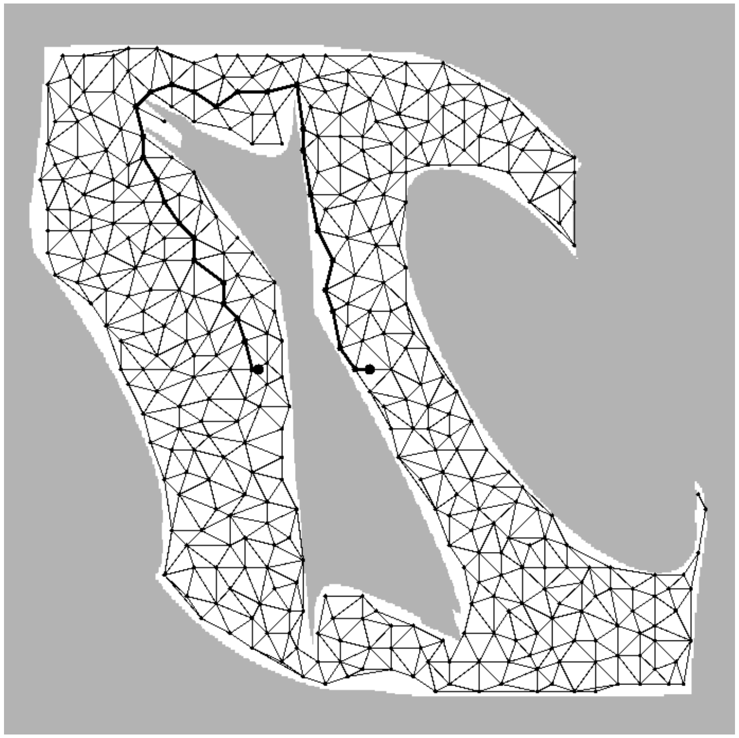
◇ Find configuration q_g' in R such that $L(q_g, q_g')$ is in CSfree

◇ In R, find a path from q_i' to q_g'

◇ The planned path is the finite sequence of line segments between them
Do postprocessing to optimize and smooth the path

This reduces path planning to a simple graph-search problem,
plus collision checking and kinematic steering

When is a roadmap adequate?



The property we want: whenever there's a path in CSfree from q_i to q_g, the roadmap contains a path from q_i' to q_g'

The *coverage* of a configuration q is

$D(q) = \{q' \in \text{CSfree} \mid L(q, q') \subseteq \text{CSfree}\}$

i.e., every point that can be reached by a straight line from q

The coverage of a set of configurations $Q = \{q_1, q_2, \ldots, q_n\}$ is

$$D(Q) = D(q_1) \cup D(q_2) \cup \ldots \cup D(q_n)$$

R is *adequate* if R is connected and $D(\text{vertices}(R)) = \text{CSfree}$

Generating an adequate roadmap

◇ Easier to use probabilistic techniques than to compute CSfree explicitly

Start with an empty roadmap R

Until (termination condition), do
    Randomly generate a configuration $q \in$ CSfree
    Add q to R iff
        either q extends R's coverage, i.e., $q \notin D(R)$
        or q extends R's connectivity, i.e.,
            q connects two unconnected subgraphs of R

Termination

Termination condition:

- ◇ Let k = number of random draws since the last time a configuration was added to the roadmap
- ◇ Stop when k reaches some value k_{max}

$1/k_{max}$ is a probabilistic estimate of the ratio between the part of CSfree not covered by R and the total CSfree

For kmax = 1000, the algorithm generates a roadmap that covers CSfree with probability 0.999

Implementation

Very efficient implementations

Marketed products used in

◇ Robotics

◇ Computer animation

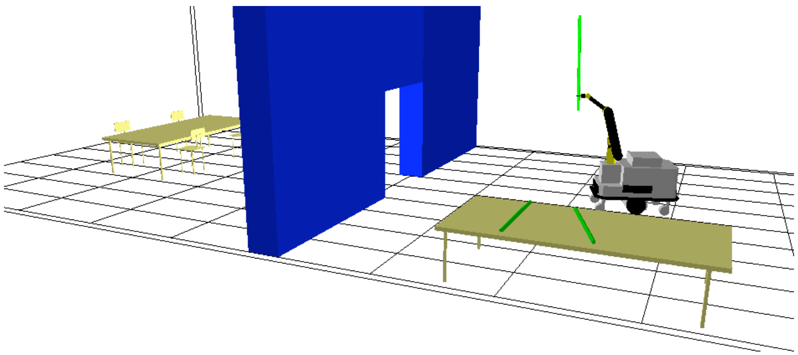
◇ CAD

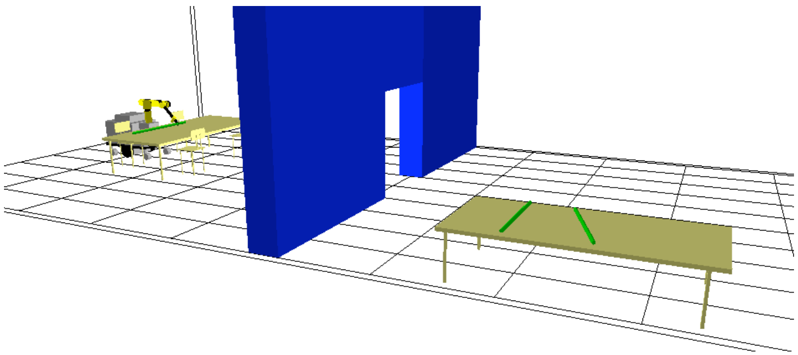
◇ Manufacturing

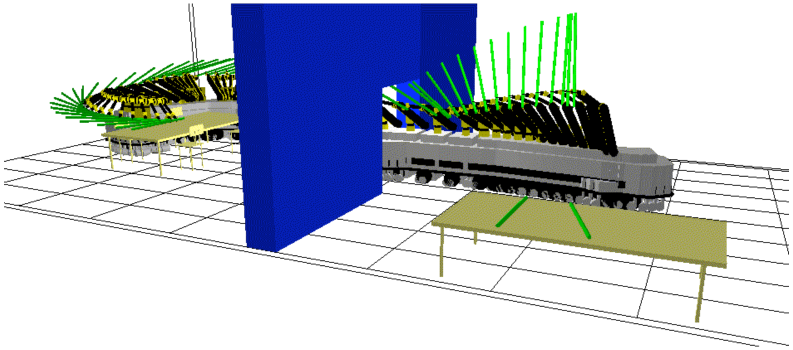
Example

Task: carry a long rod through the door

- ◇ Roadmap: about 100 vertices in 9-dimensional space
- ◇ Generated in less than 1 minute on a normal desktop machine







Motor control

Can view the motor control problem as a search problem
in the dynamic rather than kinematic state space:

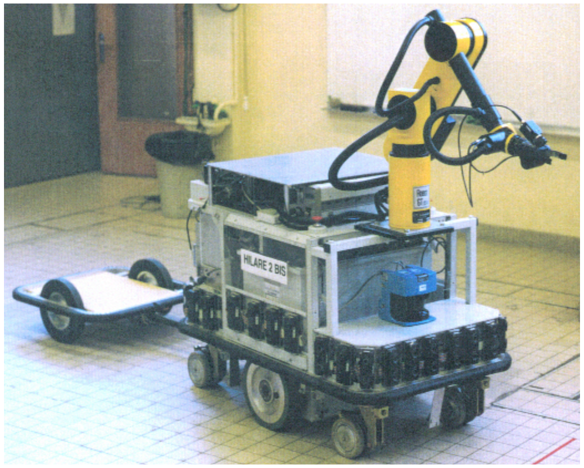
- state space defined by $x_1, x_2, \ldots, \dot{x}_1, \dot{x}_2, \ldots$
- continuous, high-dimensional (Sarcos humanoid: 162 dimensions)

Deterministic control: many problems are exactly solvable
esp. if linear, low-dimensional, exactly known, observable

Simple *regulatory control* laws are effective for specified motions

Stochastic *optimal control*: very few problems exactly solvable
$\Rightarrow$ approximate/adaptive methods

Robust robot control



Hilare, a robot at LAAS
(a French research institute)

- ◇ Sensors: sonar, laser, vision
- ◇ Motor functions: actuators, arm

Several redundant software modules for each sensory-motor function

- ◇ Localization, map building and updating, motion planning and control

Redundancy needed for robustness

- ◇ No single method or sensor has universal coverage
- ◇ Each has weak points and drawbacks

Sensory-Motor Functions

Localization

◇ Laser range data
has problems with obstacles, long corridors

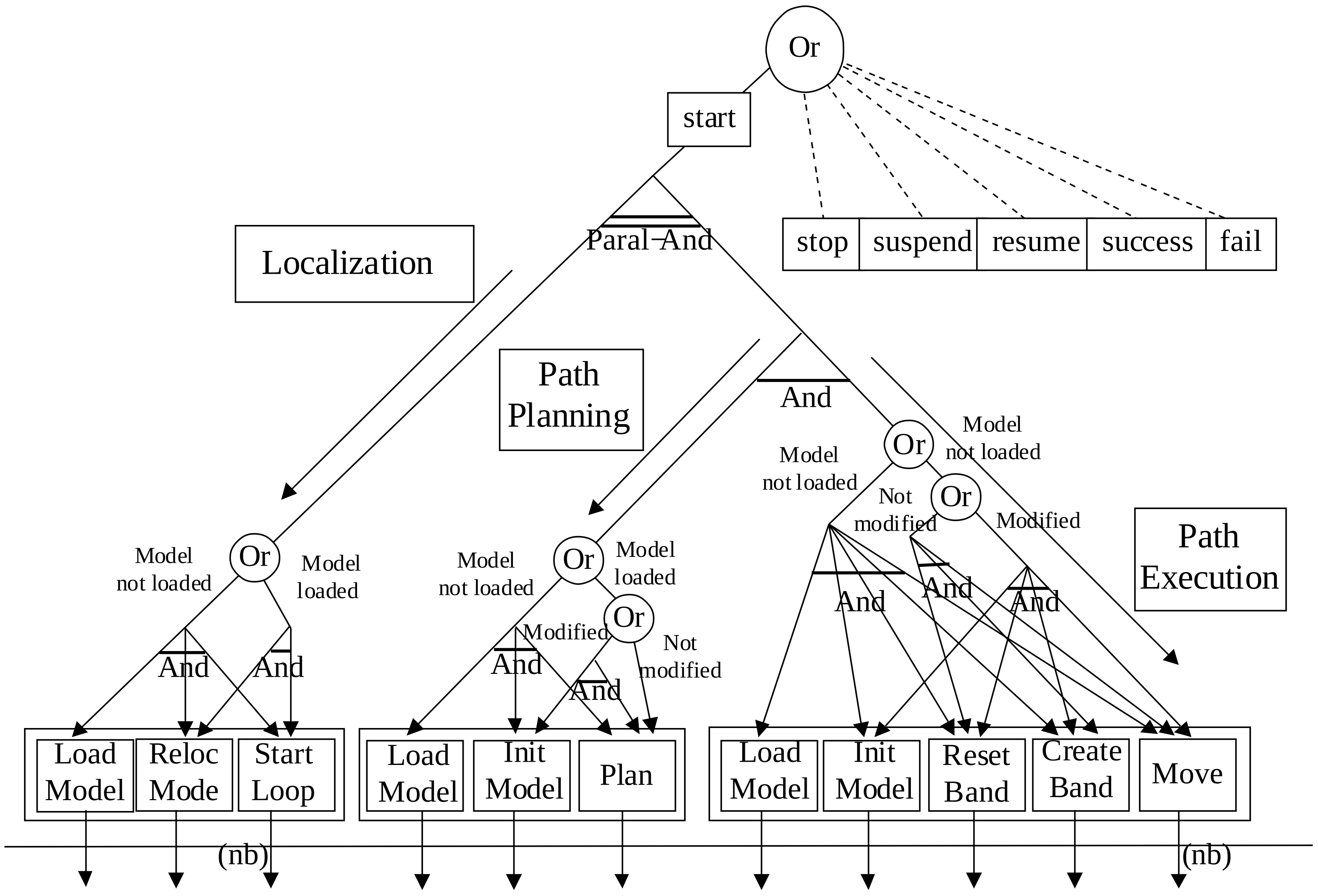
◇ Infrared reflectors, wall-mountedcameras, GPS
only work when in areas covered by the device

Elastic Band for Plan Execution
◇ Dynamically update and maintain a flexible trajectory

No single method or sensor works well in all cases
Instead, Hilare has several Modes of Behavior (or Modalities)

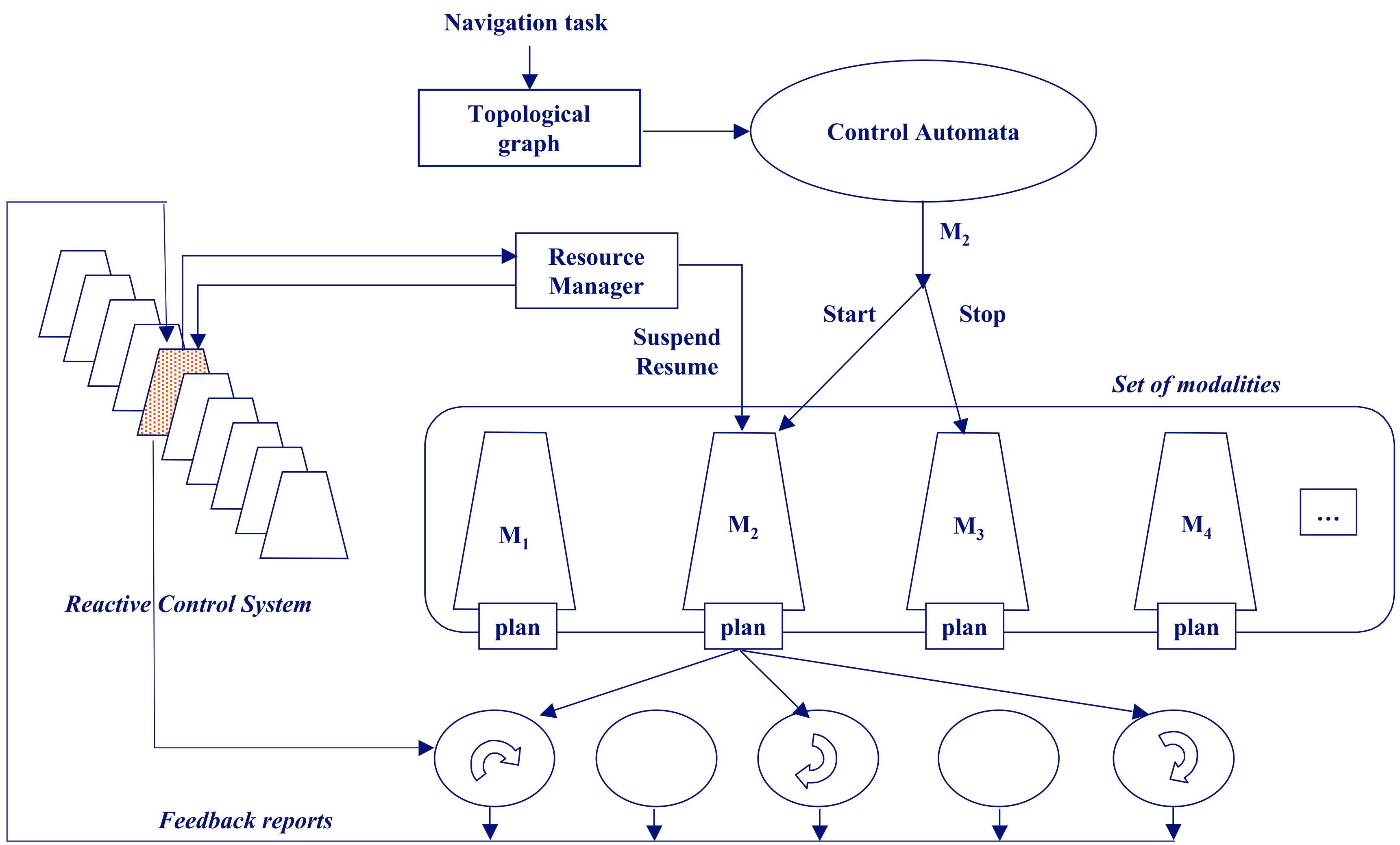
Each modality is an HTN whose primitives are sensory-motor functions
◇ Tells how to combine functions to achieve a desired task

Example of a modality



Which modality to use when?

Hilare uses an MDP to decide which modality to use under which conditions



Summary

Mobile robots and manipulators

Configuration parameters, configuration space

Path planning with roadmaps

Example of robust control