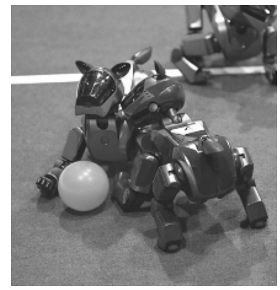
Robotics



RoboCup Challenges

- Simulation League
- Small League
- Medium-sized League (less interest)
- SONY Legged League
- Humanoid League

Small League

- Overhead camera
- Central controlling computer for each team
- Fast and agile
  - Winning teams have had the best hardware

Small League:

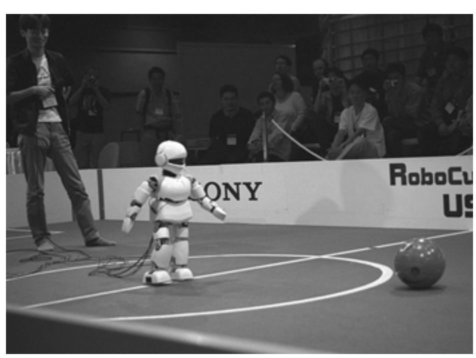
CMU vs. Cornell @ American Open (2003)

Medium-Sized League

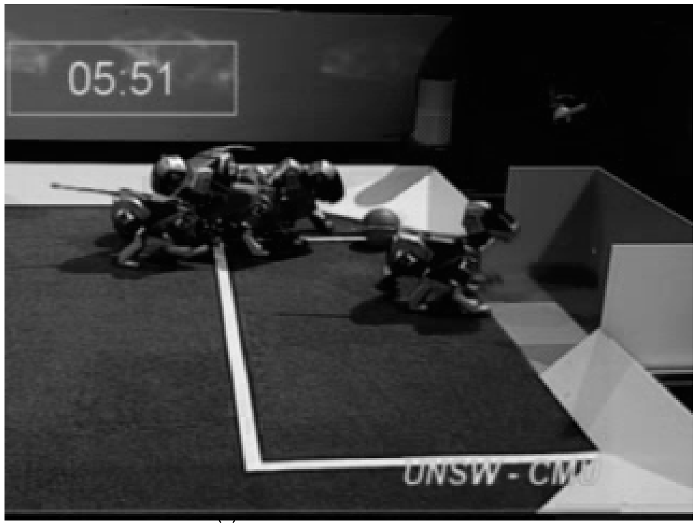
- Largest fully-autonomous robots
- Has been plagued by hardware challenges

Humanoid League

- Still demonstrating technology and skills (kicking, vision, localization)

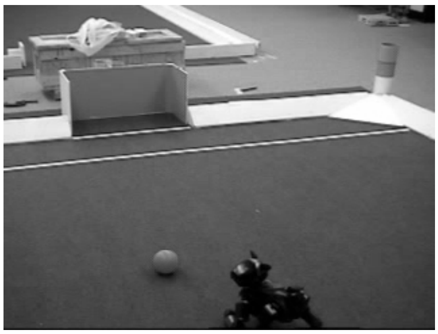


SONY Legged League
CMU vs. New South Wales (1999)



CMU vs. New South Wales (2002)

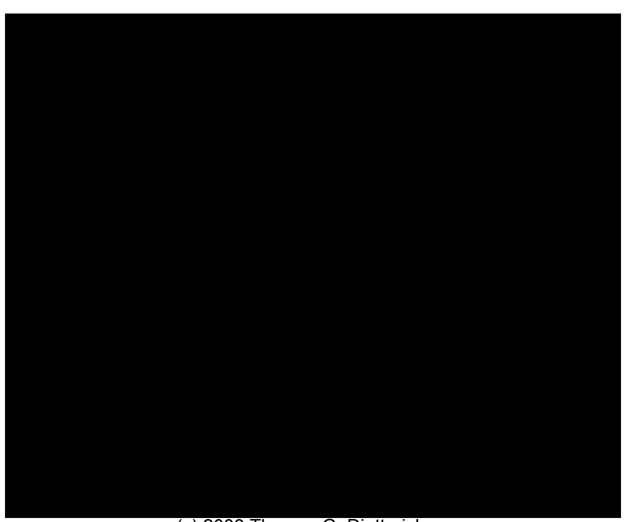
Special Purpose Vision



Bright Light

Dim Light

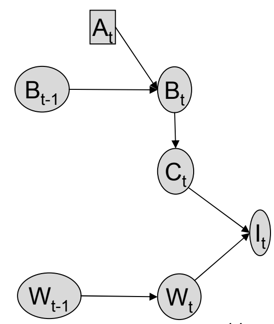
What the Dog Sees



Making Sense of Sensing

- $P(\text{Image}_t \mid \text{CameraPose}_t, \text{World}_t)$
- $P(\text{CameraPose}_t \mid \text{BodyPose}_t)$
- $P(\text{BodyPose}_t \mid \text{BodyPose}_{t-1}, \text{Action}_t)$
- $P(\text{World}_t \mid \text{World}_{t-1})$

- $\text{argmax}_{Wt}\ P(W_t \mid I_t, A_t,) =$
  $\text{argmax}_{Wt} \sum_{Wt-1,Ct,Ct-1,Bt,Bt-1} P(I_t|W_t,C_t) \cdot P(W_t|W_{t-1}) \cdot P(C_t \mid B_t) \cdot P(B_t \mid B_{t-1},A_t) =$
- $\text{argmax}_{Wt} \sum_{Ct} P(I_t|W_t,C_t) \cdot \sum_{Wt-1} P(W_t|W_{t-1}) \cdot \sum_{Bt} P(C_t \mid B_t) \cdot \sum_{Bt-1} P(B_t \mid B_{t-1},A_t)$



The World

- Locations (and orientations and velocities) of
  - self
  - ball
  - other players on same team
  - players on other team

Actions

- Actions can be described at many levels of detail
  - low level actions: moving body joints
  - intermediate level actions: walking gaits, shooting and passing motions, localization motions, celebration dances
    - learned or programmed prior to the game
  - higher-level actions: “shoot on goal”, “pass to X”, “keep away from Y”
    - decisions are made at this level during the game

Choosing Actions to Maximize Utility

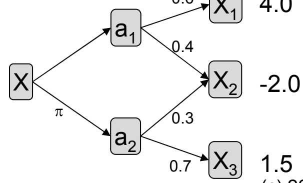
- Markov Decision Process
  - Set of states X
  - Set of actions A
  - State transition function: $P(X_t \mid X_{t-1}, A_t)$
  - Reward function: $R(X_{t-1}, A_t, X_t)$
  - Discount factor γ
  - Policy: $\pi: X \mapsto A$
    - maps from states to actions
- Value of a policy:
  - $E[R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots]$

The Reward Function

- Overall Reward function R(X)
  - reward received when entering state X
  - example: scoring goal R = +1
  - example: opponent scores: R = -1
  - reward is zero most of the time. We say that reward is “delayed”

The Value Function

- $V^{\pi}(X)$ is the expected discounted reward of being in state X and executing policy π.
- $V^{\pi}(X) = \sum_{X'} P(X'|X,\pi(X)) \cdot [R(X,\pi(X),X') + \gamma V^{\pi}(X')]$



$V^{\pi}(X) = 0.3 \cdot \gamma\, (-2) + 0.7 \cdot \gamma\, (1.5)$

$= 0.405$

$(\gamma = 0.9)$

Computing the Optimal Policy by Computing its Value Function

- Let $V^*(X)$ denote the expected discounted reward of following the optimal policy, π^*, starting in state X.

  $V^*(X) = \max_a \sum_{X'} P(X'|X,a) [R(X,a,X') + \gamma V^*(X')]$

Value Iteration:

Initialize $V(X) = 0$ in all states X

repeat until V converges:

for each state X, compute

$V^*(X) := \max_a \sum_{S'} P(X'|X,a) [R(X,a,X') + \gamma V^*(X')]$

Computing the Optimal Policy from V*

$\pi^*(X) := \text{argmax}_a \sum_{X'} P(X'|X,a) [R(X,a,X') + \gamma V^*(X')]$

Perform a one-step lookahead, evaluate the resulting states X' using V*, and choose the best action

Scale-up Problems

- Value Iteration
  - Requires O(|X| |A| B) time, where B is the branching factor (number of states resulting from an action)
  - Not practical for more than 30,000 states
  - Not practical for continuous state spaces
- Where do the probability distributions come from?

Reinforcement Learning

- Learn the transition function and the reward function by experimenting with the environment
- Perform value iteration to compute π^*
- Other methods compute V^* or π^* directly without learning $P(X'|X,A)$ or $R(X,A,X')$
  - Q learning
  - SARSA(λ)

Scaling Methods

- Value Function Approximation
  – Compact parameterizations of value functions (e.g., as linear, polynomial, or non-linear functions)
- Policy Approximation
  – Compact representation of the policy
  – Gradient descent in "policy space"

Multiple Agents

- The Markov Decision Process is a model of only a single agent, but robocup involves multiple cooperative and competitive agents
- There is a separate reward function for each agent, but it depends on the actions of all of the other agents
  – $R_1(X, a_1, \ldots, a_N, X')$
  – $R_2(X, a_1, \ldots, a_N, X')$
  …
  – $R_N(X, a_1, \ldots, a_N, X')$

Game Theory

- Each agent ("player") has a policy for choosing actions
- The combination of policies results in a value function for each player
- Each player seeks to optimize his/her own value function
- Stable solutions: Nash Equilibrium
  - Each player's current policy is a local optimum if all of the other players' policies are kept fixed
  - Each player has no incentive to change
- Computing Nash Equilibria in general is a research problem, although there are special cases where solutions are known.

Stochastic Policies

- In games, the optimal policy may be stochastic (i.e., actions are chosen according to a probability distribution)
  - $\pi(X,A)$ = probability of choosing action A in state X
- Example: Rock, Paper, Scissors
  - Nash equilibrium: choose randomly among the three actions

How to choose actions when you don't know your opponent's policy

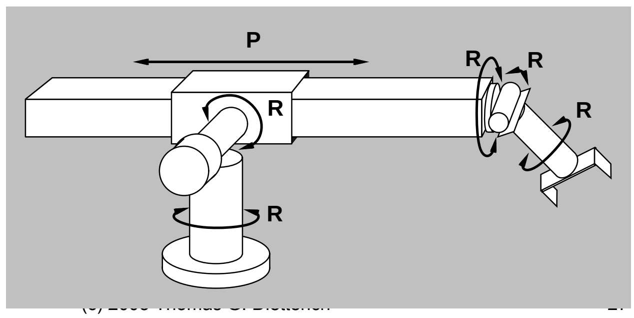
- Consider one or more policies that your opponent is likely to play
- Design a policy that works well against all of them

The Segway League?

Non-Mobile Robot Motion Planning

- Industrial robot arms
  - Degrees of Freedom (one for each independent direction in which a robot or one of its effectors can move)

How many (internal) degrees of freedom does this arm have?



Kinematics and Dynamics

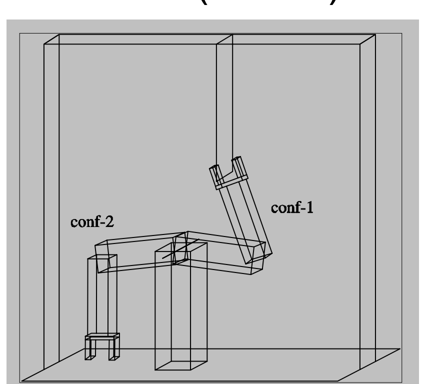
- Kinematic State
  - joint angle of each joint
- Dynamic State
  - Kinematic State + velocities and accelerations of each joint

Holonomic vs. Non-Holonomic

- Automobile (on a plane)
  - 3 degrees of freedom (x,y,θ)
  - only 2 controllable degrees of freedom
    - wheels and steering
- Holonomic: number of degrees of freedom = number of controllable degrees of freedom
  - easier to control, often more expensive
- Non-Holonomic: degrees of freedom > controllable degrees of freedom

Path Planning

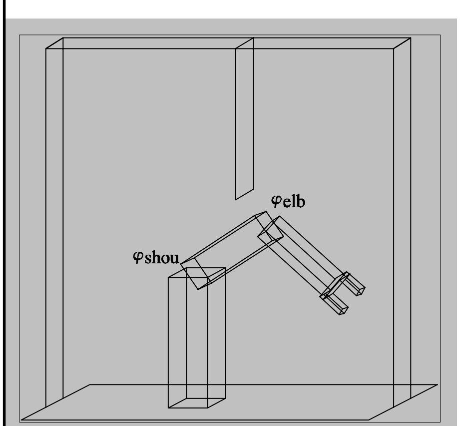
- Want to move robot arm from one location (conf-1) to another (conf-2)

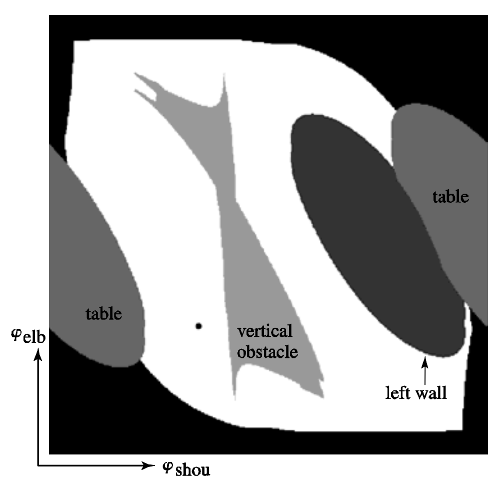


Two Different Coordinate Systems

- Locations can be specified in two different coordinate systems
  - Workspace Coordinates
    - position of end-effector (x,y,z) and possibly its orientation (roll,pitch,yaw)
  - Joint Coordinates
    - angle of each joint

Configuration Space (C-Space)





Forward and Inverse Kinematics

- Forward Kinematics
  - Given joint angles compute workspace coordinates
    - easy
- Inverse Kinematics
  - Given workspace coordinates compute joint angles
    - hard: may exist multiple solutions (often infinitely many)
- Path planning involves
  - finding a path
    - easy to do in joint angle space
  - avoiding obstacles
    - easy to do in workspace

Computing Obstacle Representations in C-Space

- Must convert each obstacle from a region of workspace to a region in configuration space
- Often done by sampling
  - generate grid of points in C-space
  - test if corresponding point is occupied by obstacle
- Interesting computational geometry challenge

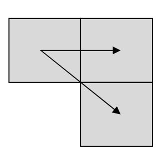
Path Planning in Configuration Space

- Cell Decomposition Methods
- Potential Field Methods
- Voronoi Graph Methods
- Probabilistic Roadmap Methods

- key problem: C-Space is continuous!

Cell Decomposition

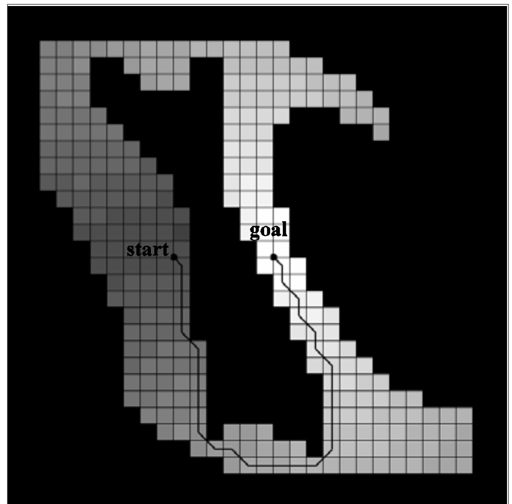
- Define a grid of cells for free space
  – A path consists of a sequence of cells
  – Legal moves: go from center of one cell to center of 8 neighboring cells:



- Converts path planning to discrete search problem (use A* or Value Iteration)

Cell Decomposition

cell color indicates optimal value function (distance to goal along optimal policy)



Problems with Cell Decomposition

- How do we handle cells that overlap obstacles?
  - ignore: algorithm is incomplete (possible plan will not be found)
  - include: algorithm is unsound (plan may not work)
- Number of cells grows exponentially with number of joints (dimensionality of C-Space)
- Paths may touch (or pass too close to) obstacles

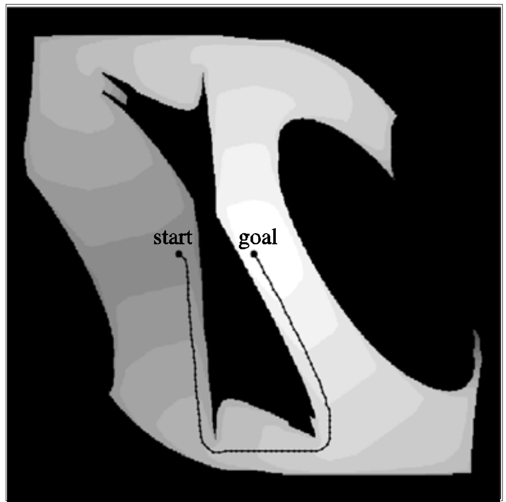
Solutions to Cell Problems

- Cells too big/too small
  - Use variable resolution cell size. Degree cell size near obstacles
- Cell scaling
  - Voronoi and Roadmap methods
- Touching obstacles
  - Potential Field Methods

Potential Field Method

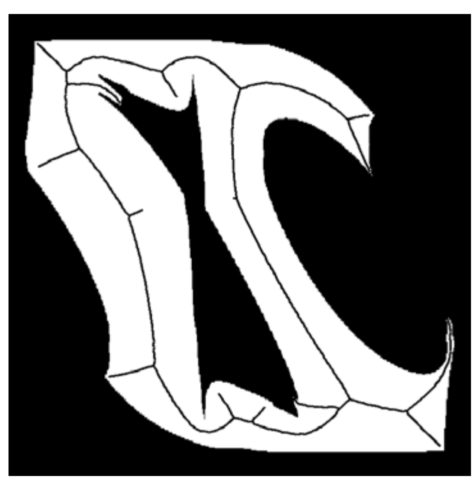
- Define a "cost" for getting close to obstacles ("the potential")
- Find optimal path that minimizes the combined path length + cost

Potential Field Result



Voronoi Methods (“skeletonization”)

- Define set of points equidistant from two or more obstacles
- This has lower dimensionality (often 1-D). Finitely-many intersections.
- Path: from start to Voronoi skeleton, along skeleton, from skeleton to end



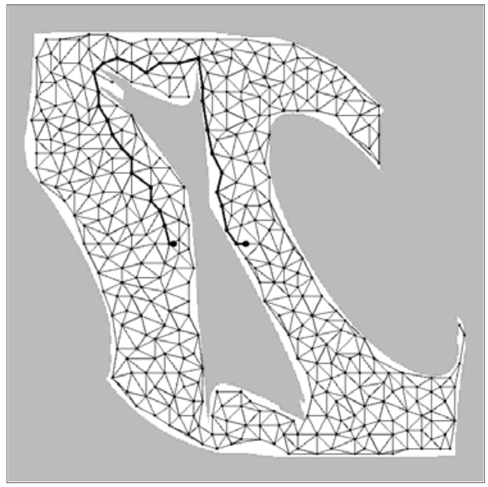
Problems with Voronoi Method

- Resulting paths maximize “clearance” from obstacles
- Does not work well in large open spaces
  - Path goes through middle of space
- Computing the diagram can be difficult in C-Space.

Probabilistic Roadmap

- Draw a sample of points in C-Space.
- Keep those points that are in free space.
- Compute Delauney Triangulation of the sample points
- This gives a graph of points in free space
- Search in this graph

Probabilistic Roadmap



Scaling Problems

- All of these methods do not scale to very high dimensional spaces
  - Probabilistic roadmap and Voronoi method scale best
  - Probabilistic roadmap is cheapest to compute
    - Sampling can be dynamically refined based on initial paths

Executing Robot Plans

- Path only specifies the kinematic state of the robot arm
- Actually moving the arm must deal with dynamics: acceleration, mass, friction, etc.
- Control theory has well-developed methods for smoothly following a trajectory
  - e.g., PID controllers (proportional integral derivative controllers)

Robotics Summary

- Robots live in partially-observable, stochastic environments that may contain other cooperative and competitive agents
- Robot Tasks
  - localization
  - mapping
  - action selection
  - planning (single agent; multiple cooperative agents; multiple competitive agents; teams)
  - action execution

Robot Planning

- For single-agent stochastic environments
  - MDP model
    - Value Iteration
    - Reinforcement Learning
- For multiple-agent stochastic environments
  - Game theory model
    - Still a research topic
- For single-agent deterministic (non-mobile) environment
  - Configuration space planning