Reactive Architecture

Environment

Sensing → Actuators

Reactive Control

- no memory - no look-ahead reacts to the current stimuli
- reacts directly to current sensory information
- reactive behaviors:
  (or mappings between situations and actions)

Can we achieve bigger functionality if we combine them?

Simplest scenario one situation one action:
Motivation - biology, V. Braitenberg’s Vehicles
one can design simple continuous or discrete state rules.

1. Ethology/biology study of animal behavior
   motivated by observing behaviors of real animals,
   figure out the decomposition, specification

2. Try to find high-fidelity models of neurological substrate
   (cochroach like behavior) - studying of neural pathways
   replicating it on robotic systems

3. Situated approach
   - for each situation, figure out the response

Hierarchical Hybrid Architectures

Decision making/Control

Low Level Reactive Behaviors

Mid-level coordination – task management

Goal Selection - Planning

\[ x[t + 1] = f(x[t], u[t]) \]
Intelligent Highway Systems

- Drive your car into transition area
- Place the car under automatic control
- Systems drives your car into automatic lane
- Join - split - exit - change lane
- At exit transition area the control is returned to you

More complex reactive behaviors

What are the alternatives for representing more complex behaviors?

Example: if we have n-sensors and the inputs are mutually exclusive we must encode exponential number of rules for action. Why?

For simple sensors - possible to use lookup table:
- store all the values and possible responses
- have some efficient means to search the table
- this approach is unlikely to scale to more complex systems
**Behavior compositions**

*Competitive* (one active at the time)
- Fixed prioritization
- Based on the highest level of activation
- Voting architecture (example)

*Cooperative schemes* (some fusion)
- Superposition of the vector fields or schemas

Compose behaviors to form assemblages of more elementary behaviors (examples move-to-pole, wander)

---

**Robot Architectures**

How to put the individual behaviors together?
How do you decide which one to use and when?

How to put the individual behaviors (stimulus response pairs)

=> Design of Control Architecture

---

**Subsumption Architecture**

Brooks ’85

Guidelines:
- Build the system from bottom up
- Components are task achieving behaviors
- Components are executed in parallel
- Components are organized in layers
- Lowest layers handle most basic tasks
- Higher levels exploit the lower levels
- Each component has its tight connection between
- Perception and action

Bottom up design process

---

**Subsumption Architecture**

Reason about behaviors of objects
Plan changes to the world
- Identify objects
- Monitor changes
- Build maps
- Explore
- Wander
- Avoid objects

---

sensors

actuators
Superposition of different behaviors

Go to target $w_1$

Avoid $w_2$

Go to goal $w_3$

Behavior-based Architecture

Motivation
1. To keep all the advantages of the Reactive Control
2. Allow representation of the environment
3. Allow bigger flexibility and reconfiguration depending on the task

(... this is what subsumption architecture was lacking)

Behavior Assemblages

- Power of abstraction
- Modularity
- Reuse of elementary behaviors
- "reason" over them
- Coarser level of granularity – good for adaptation and learning

Abstraction's in terms of FSM's

Composition of the behaviors

Examples: FSM for navigation
Pole finding robot (AAA competition)

Wander and avoid behavior
Example of trash collecting robot – each node is an assemblage of behaviors – more details on the transitions

Motion Planning Definition

- Given initial position and orientation and goal
- position and orientation of the robot A in
- workspace, generate a path of continuous
- positions and orientations of A avoiding contact
- with obstacles’s. Report failure when no such
- path exists.

- Geometry of the robot A and obstacles B1, … Bn
- And their position in the workspace is known

Configuration Space

Consider the motion planning problem in general, regardless of the type of robot and the type of environment – reduce the motion planning problem.

Configuration of an object – specify position of every point on the object with respect to the fixed coordinate frame.

Configuration space – set of all possible configurations – each configuration is described by a set of parameters

\[ A(q) \]  Subset of workspace occupied by a robot at configuration \( q \)

Workspace \( W \)

Mobile Robotics

Navigation (office delivery robot)

- map building and exploration
- motion planning in dynamic environments
- localization – where am I?
- manipulation
- collaboration between multiple robots
Example manipulator configuration space - cross
Product of the range of all joint angles

$S^1$ Circle revolute joint
$S^1 \times S^1 \times S^1 \ldots \times S^1$ Revolute joints
$S^1 \times S^1 \times R \times R$ Revolute and prismatic joints
$\mathbb{R}^2 \times S^1$ Mobile robot in a plane

Obstacles in Configuration space

Obstacle $B_i$ In the workspace corresponds to
A region $CB_i$ In the configuration space

$CB_i = \{q \in C | A(q) \cap B_i \neq 0\}$
All obstacles

$\bigcup_{i=1}^{m} CB_i$
Free configuration space

$C_{free} = C \setminus \bigcup_{i=1}^{m} CB_i$

Configuration space

- General path planning techniques
- Road map approaches
- Cell Decomposition approaches
- Potential Fields

- Advanced Issues
- Multiple Moving objects (robots or obstacles)
- Articulated Robots - multiple joint angles, self-collisions
- Kinematic Constraints
- Dealing with uncertainty - control and geometry are not perfect

Computational Complexity

- Computational complexity of the path planning methods - depends on the size of the configuration space, number of obstacles in the workspace and their representation
Representations of Space - Maps

- Geometric Decompositions - basic geometric primitives describing objects -
  points, polygons, lines, curves, circles, ellipsoids,
  planar surfaces, superquadrics (depends whether in 2D or 3D)
- Spatial Decompositions - discretization of the space
  - bitmaps, occupancy grids, quad-trees, binary partitioning trees, uniform decompositions,
    irregular sampling

Requirements on Motion Planning Algs.

Completeness - Is the algorithm guaranteed to find a path if one exists
Resolution completeness - complete up to its limits of resolution
Optimality - Is the cost of obtained path versus the optimal path
Space and time complexity - time and space needed to find the path

Motion Planning

How to compute $C_{free}$ how to find a path in it.
Idea - reduce the problem to the problem of finding a path in a configuration space - reduces the problem to a point-like robot.

Examples:
2D polygonal environment - circular robot what is
2D polygonal envir., polygonal robot, purely translation case (ordering of normal's algorithm )

Road Map Methods

Given $C_{free}$ compute some representation of it which simplifies the path planning tasks.
Reduces the path planning to following the roadmap + getting onto a roadmap
Roadmaps construction/path planning algorithms
  - Deterministic - complete, guarantee that when there is a path it will find it
    Visibility Graphs - construction algorithm + complexity analysis,
Reduced visibility graphs - polygonal representations
  - Generalized Voronoi Diagrams - definition, construction algorithm, complexity analysis, path planning
Visibility Graphs $G = (V,E)$

Visibility graph - undirected graph
Polygonal obstacles
Nodes - vertices of polygons
Two nodes are connected with an edge if the edge is an edge of polygonal obstacle or if the straight line segment lies in connecting two vertices lies in $C_{\text{free}}$
Suitable for two dimensional configuration space
Reduced visibility graph - not all edges are needed
- keep the convex vertices of $CB$
- keep the non-$CB$ edges that are tangent segments
$CB$ Obstacle region in the configuration space

Generalized Voronoi Diagrams
Retract the 2-D free configuration space to 1-D subset of itself. Definition (in notes).
Voronoi diagram of set of points - set of straight line segments
- Compute the lines bisecting each pair and their intersections
- keep line segments with more than one nearest neighbor
In polygonal environments - Voronoi Diagram is a collection of Straight lines and parabolic curve segments
Voronoi Diagram of polygonal env. (simple method)
1. Compute all parabolic and line segments for point-point, point-line line-line pairs
2. Compute intersection points (dividing arcs into segments)
3. Keep segments which are closest only to vertices/edges that defined them.

Probabilistic Roadmaps
Constructing C-space is expensive – create roadmap by sampling
Pick n-configurations at random
Keep those that are in free configuration space (collision checking)
For each pair of free configurations
- try to connect them (e.g. via straight line or using simple motion planner)
- each connection which lies in free configuration space is an edge of a roadmap
Path planning - connect start and end to roadmap; search the roadmap
Good for high-dimensional spaces, building the roadmap is the difficult part, probabilistically complete (if run long enough)
Issues - sampling (uniform? depends on the environment - how to expand the roadmap, distance measure etc)

Potential Field Methods
Idea robot is a particle
Environment is represented as a potential field (locally)
Advantage - capability to generate on-line collision avoidance
Compute force acting on a robot - incremental path planning
$$F(q) = -\nabla U(q)$$
Example: Robot can translate freely, we can control independently $\dot{x}, \dot{y}$
Environment represented by a potential function
$$U(x, y)$$
Force is proportional to the gradient of the potential function
$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = -\nabla U(x, y)$$
Attractive potential field
- Linear function of distance
  $$U_a(q) = \xi \|q - q_{goal}\| \quad F_a(q) = -\nabla U_a(q) = -\xi \frac{(q - q_{goal})}{\|q - q_{goal}\|}$$

- Quadratic function of distance
  $$U_a(q) = \xi \frac{1}{2} \|q - q_{goal}\|^2 \quad F_a(q) = -\nabla U_a(q) = -\xi (q - q_{goal})$$

Combination of two - far away use linear, closer by use parabolic well

Repulsive potential field
$$U_r(q) = \frac{1}{2} \nu \left( \frac{1}{\rho(q,q_{obst})} - 1 \right) \quad \text{if} \quad \rho(q,q_{obst}) \leq \rho_0$$
$$U_r(q) = 0 \quad \text{else}$$

Minimal distance between the robot and the obstacle

Resulting force
$$F(q) = -\nabla (U_a(q) + U_r(q))$$

Iterative gradient descent planning
$$q_{i+1} = q_i + \delta_i \frac{F(q_i)}{\|F(q_i)\|}$$

Issues - multiple obstacles - nonconvex obstacles - how to compute Distance - Can be computed for polygonal and polyhedral obstacles

Issues - local minima

Heuristics for escaping the local minima
Can be used in local and global context
Numerical techniques
Random walk methods

Grid search algorithms (AI)
- Pick a discretization of the free configuration space, set up a grid
- Define neighbourhood structure
  1-neighbourhood, 2-neighbourhood
- The resolution should be sufficient

Good - always finds a solution if one exists
Bad - time and space explode as the dimension and neighbourhood structure rises

Set up cost function - which determines which node to expand next

Alternatives
- Same as potential function - discrete (example))
- Wavefront propagation algorithm
- Compute the skeleton example

Sensor Based Path Planning

Bug algorithm
- Move directly towards goal
- Circumnavigate obstacle + variations of this
- Need to know the goal

Later
Navigating in the presence of uncertainty
- Landmark based navigation, probabilistic maps, exploration and navigation

Landmark-based methods
• Map building and exploration

• Automated exploration
• Map alignment