Computational Principles of Mobile Robotics

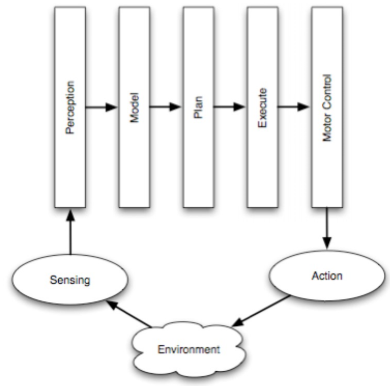
System Control

System control

- How best to structure software within a robot to integrate sensing, reasoning and action?
- Systems must provide effective software structures for device control and abstraction, parallel processing, distributed and real-time computation.
- Building an effective and efficient control architecture for autonomous systems is difficult.
  - Real time, multiple processes, external events, computational resources, complex tasks.

8.1 Horizontal decomposition

- Classic functional, horizontal, top-down methodology.
- World is processed a represented sequence of discrete set of times, actions, and events.

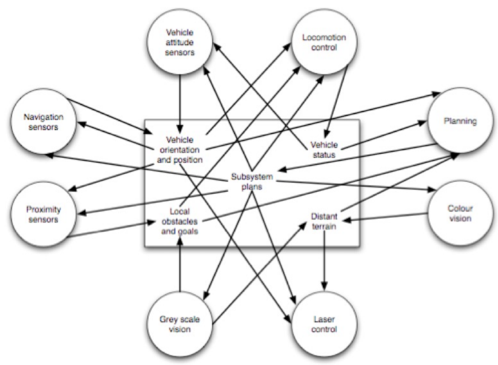


8.1.1 Hierarchical control

- Tasks decomposed by function
  - Low-level functions abstract hardware components.
  - Hierarchical structures group components together.
  - Highest level provide overall vehicle control.
- Very simple structure, often found embedded in more complex systems.

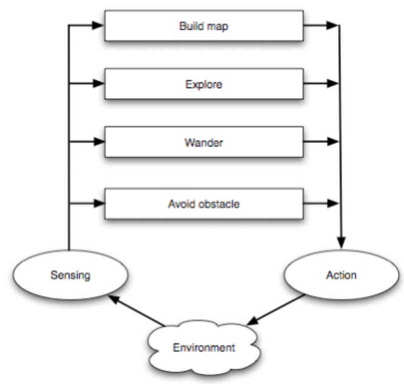
8.1.2 Blackboard systems

- Top-down systems have a hierarchical communication structure.
- Blackboard systems utilize a common ‘blackboard’ upon which parallel components communicate.



8.2 Vertical decomposition

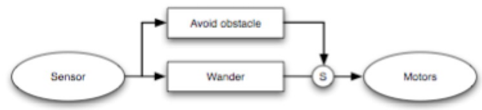
- Relate sensation to action directly.
- Can provide simpler structures of multiple goals and sensors.
- Can provide a more robust and extensive control architecture.



8.2.1 Subsumption

- Provide 'levels of competence'
  - Level 0 – avoid contact
  - Level 1 – wander
  - Level 2 – explore
  - Level 3 – build map
  - …

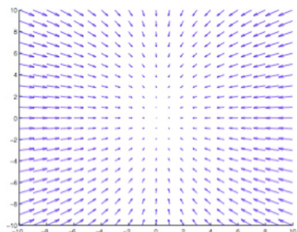
(a) Wander



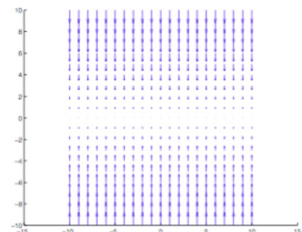
(b) Avoid obstacle

8.2.2 Motor schema

- Each behavior provides a motion command.
- Motion commands integrated using a vector sum
  - Contrasts with priority model of subsumption.



(a) Goal-seeking schema

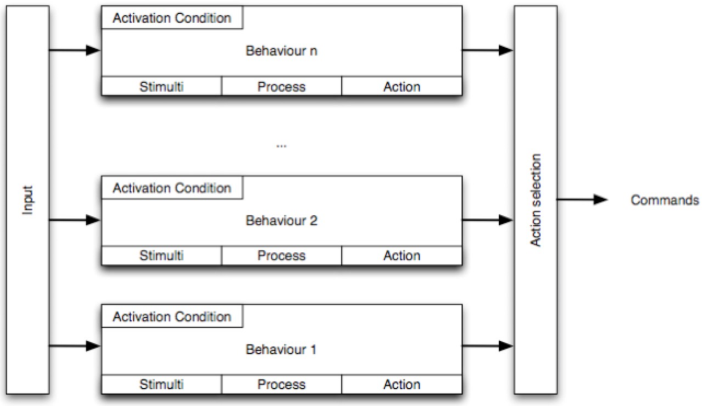


(b) Path-seeking schmea

8.2.3 Continuous control

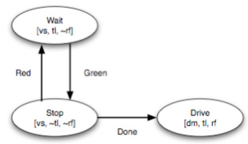
- Treat the control problem as a classic control problem.
- Inputs
  - Sensor information.
- Output
  - Commanded motion.

8.2.4 Behavior-based systems



8.3 Integrating reactive behaviors

- Can be useful to have a limited set of behaviors and to manipulate this set based on external events.
- A number of frameworks have been developed to support this.

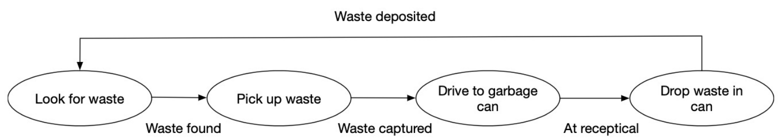


(a) Finite-state machine

```
PROCS = {
    rf  “driveOnRoad”,
    tl  “checkTrafficLight”,
    vs  “vehicleStop”,
    dm “distanceMonitor”
}
STATES = { drive, wait, stop }
EVENTS = { red, green, done }
MSGS = { distance }
WHILE drive(d) {
    SET distance = d;
    RUN rf, tl, dm;
    EVENT red GOTO wait;
    EVENT done GOTO stop;
}
WHILE wait() {
    KILL rf;
    RUN vs, tl;
    EVENT green GOTO drive;
}
WHILE stop() {
    RUN vs;
    KILL rf, tl;
}
```

8.4 Finite state machines

- Decompose task into sub-tasks.
- Link subtasks by conditions that must be true to transit from one subtask to another,



Finite State Machines

- For (simple) high level robot control finite state machines are a common solution.
  - ROS SMACH
  - ROS2 YASMIN
- Can imagine a hierarchical version in which nodes themselves are finite state machines.
- Works well for simple tasks but for complex behaviors can be insufficiently expressive.

8.5 Behavior trees

- Found in robotics and in video games (where it is typically used as the 'AI' for NPC).
- Tree structure (root, internal nodes, leaves).
- Ordering of children within the parent is known.
- On a regular basis a node is queried (called a tick) and returns one of
  - SUCCESS
  - FAILURE
  - RUNNING – task not yet complete
- Non-leaf nodes obtain their answer by ticking their children.

Behavior trees

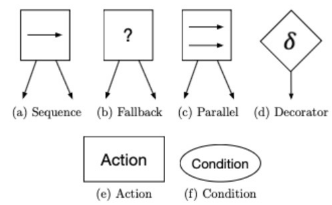
- Leaves
  - Action Node (actually cause something to happen in the environment).
    - E.g., Look for waste
  - Condition Node (query the environment)
    - E.g., Waste here
  - Return one of SUCCESS, FAILURE, RUNNING

Behavior trees

- Non-leaf Nodes
  - Decorator Node – has one child and manipulates it
  - Control Node – controls the way in which its children are ticked and their response processed
- Decorator Nodes
  - Can have custom ones
  - Sample: Negator - If the child returns SUCCESS, return FAILURE, if FAILURE then return SUCCESS

Behavior trees

- Control Nodes
  - Sequence – tick children in order. For each one
    - FAILURE -> return FAILURE immediately
    - RUNNING -> return RUNNING immediately
    - SUCCESS -> continue with next child
    - If all return SUCCESS, return SUCCESS
  - FALLBACK – tick children in order. For each one
    - SUCCESS -> return SUCCESS immediately
    - RUNNING -> return RUNNING immediately
    - FAILURE -> continue with next child
    - If all return FAILURE, return FAILURE



8.6 High-level control

- Provide very high level (abstract) mission control of the system.
- Typically built in special-purpose/traditional AI planning languages (e.g., Prolog)

```
done :- current_phase(mission_abort).
done :- current_phase(mission_complete).
execute_mission :-
    initialize_mission,
    repeat,
    execute_phase,
    done.

initialize_mission :-
    ood('start networks',X),
    asserta(current_phase(1)),
    asserta(complete(0)),
    asserta(abort(0)).
execute_phase :-
    current_phase(X),
    execute_phase(X),
    next_phase(X),
    !
```

8.6.1 STRIPS

- Historically significant and relevant planning formalism.
- Procedures/actions represented by operators.
  - Precondition
  - Add list
  - Delete list

$$
\begin{aligned}
&\text{PICKUP}(x) \\
\text{precondition}: \quad & \text{EMPTYHAND} \wedge \text{Clear}(x) \wedge \text{On}(x, y) \\
\text{delete}: \quad & \text{EMPTYHAND}, \text{Clear}(x), \text{On}(x, y) \\
\text{add}: \quad & \text{INHAND}(x)
\end{aligned}
$$

8.6.2 Situation calculus

- First-order language for representing dynamically changing worlds

$$\text{On}(B, A, s) \wedge \text{On}(A, \text{Table}, s)$$

$$\forall s \forall x[\neg \text{On}(x, \text{Table}, s) \Rightarrow \text{On}(x, \text{Table}, \text{Putontable}(x, s))]$$

8.6.3 GRAMMPS

- Mission planner and execution module.
- Intended to be deployed over a collection of mobile robots.

8.6.4 Other high-level control approaches

- A number of high level (cognitive robotics) languages and representations have been developed.
- Golog as an example. It relies on the situation calculus and provides a nondeterministic programming language to plan tasks for the robot.

8.7 Alternative control formalisms

- Large range of alternative control formalisms exist
  - Neural networks (covered earlier), fuzzy logic, genetic algorithms, etc.
- These approaches build controllers using these formalism to take advantage of the non-linear properties of these approaches.
  - Also inherit their limitations, of course.

8.7.1 NN’s

- Often, NN’s are used internally (e.g., a node is implemented as a NN)
- Can have end-to-end NN controllers
  - One concern here is safety

8.7.2 Fuzzy logic

- Build a controller based on fuzzy logic
  - A many-valued logic in which truth lies between 0 (false) and 1 (true).
- Provides an expressive language for logical operations
  - Can define what to do when the goal is distant, near, close
    - And define these things in a formal structure
- Often becomes difficult to define values of primitives in a reasoned manner.

8.7.3 Genetic algorithms

- Learning algorithms based on model of genetic information exchange.
- A number of successes for robot control, especially in gait synthesis and similar tasks.

```
1.   Generate intial random population of chromosomes
2.   Evaluate fitness of each element of current population
3.   while termination criteria is not satisfied do
4.       begin
5.           Create a new population from the current population
6.               [Selection] Select two parents based on fitness
7.               [Crossover] Create new offspring using crossover
8.               [Mutation] Mutate offspring
9.           Evaluate fitness of new population
10       end
```

8.8 End-to-End Learning

- One extreme approach to vehicle control is to replace the entire control system with one end-to-end deep network, to directly map sensor input to vehicle control.
  - Advantage – system learns the entire process.
  - Disadvantages – little opportunity for human inspection, tuning and adjustment. It may also be expensive to transfer one system to another or to adapt to differences in the environment or task.