

## Lecture 6: Three-Level Architectures

CS 344R/393R: Robotics  
Benjamin Kuipers

## Combining Different Types of Control

- We have studied local control laws.
  - Each law makes simple local assumptions
  - Tightly-coupled closed-loop control
- How do we combine control laws?
- How do we do high-level planning?

## GOFAI: Good Old-Fashioned AI

- Factoring the problem of acting intelligently
  - **Sense**: build an accurate description of the world
  - **Plan**: derive a sequence of actions that will provably achieve the goal
  - **Act**: perform that sequence of actions
- Problems with the Sense-Plan-Act cycle
  - Sensing and Acting are never perfectly accurate
  - Planning can't consider all possibilities
  - The whole cycle is too slow.

## Rodney Brooks (1986) “Subsumption Architecture”

- Changed the course of AI Robotics.
  - The Sense-Plan-Act loop is too slow, and can never have an accurate enough model.
- Use a hierarchy of fast reactive loops.
  - Each loop capable of complete behavior.
  - Higher loops modify the behavior of lower ones.
  - [James Albus, *Brains, Behavior, & Robotics*, 1981]

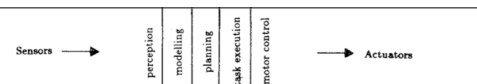
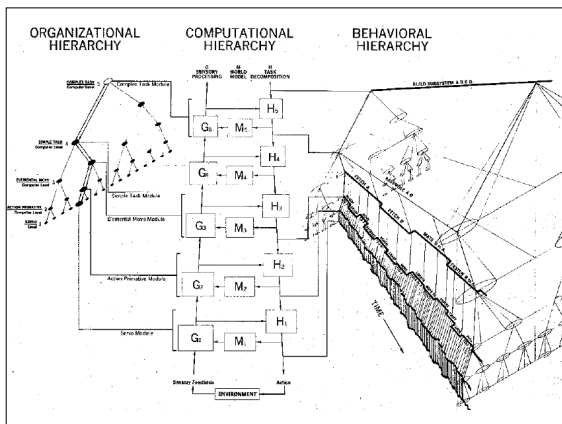


Fig. 1. Traditional decomposition of a mobile robot control system into functional modules.

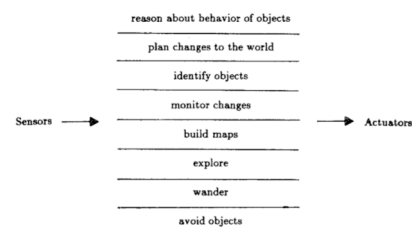
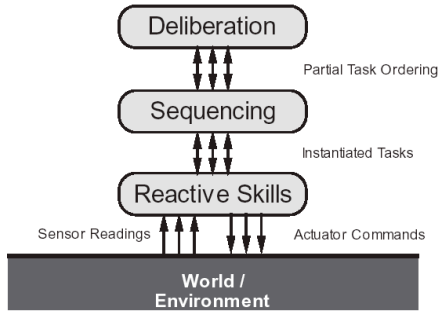


Fig. 2. Decomposition of a mobile robot control system based on task-achieving behaviors.

### Three-Level Architectures



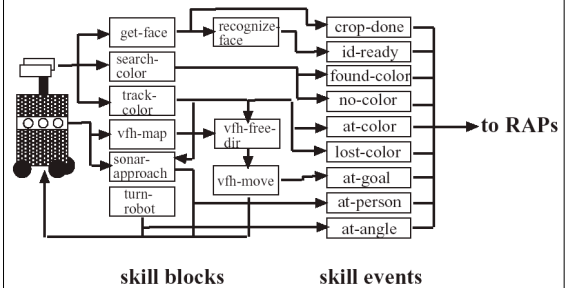
### Three-Level Architectures

- **Reactive skills:**
  - Control laws tightly coupled with environment
  - Generalized to observers and sensory transforms
- **Sequencing:**
  - Select *task network* of currently active skills
  - Accomplish specified task using skill hierarchy
- **Planning:**
  - Reason about goals, resources, and timing.
  - Symbolic AI search and planning methods.

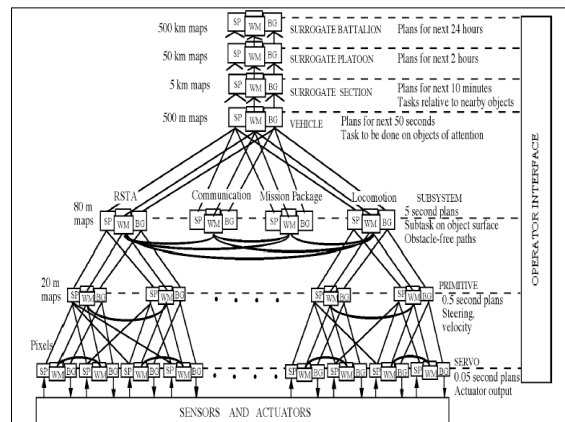
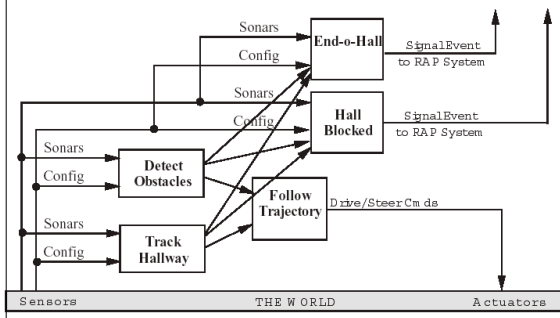
### Skills

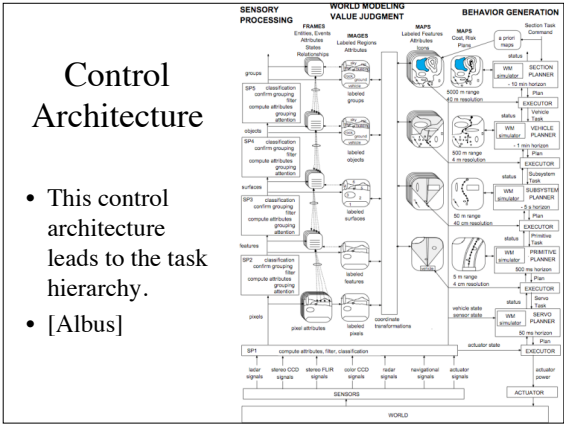
- A *skill* can be a local control law,
  - or it can estimate features from sensory input,
  - or it can compute signals for the sequencer.
- It makes a “simple-world” assumption.
- A skill has several components:
  - Input and output specifications
  - Computational transform from input to output
  - Initialization on system setup
  - Enable function on skill startup
  - Disable function on skill shutdown.

### Skill Network for Finding and Recognizing People



### Skill Network (partial) for Navigation task





### Sequencing

- Assemble an appropriate network of skills.
  - Disable skills no longer needed
  - Enable and link skills needed for current task
- Finite state machines are a possibility:
  - Select a skill to execute from a set of options
  - Decide when the current skill should terminate
- This is a good start, and we'll explore it.
  - But we will need to go beyond the FSM.

$$\dot{\mathbf{x}} = F(\mathbf{x}, \mathbf{u})$$

$$\mathbf{y} = G(\mathbf{x})$$

$$\mathbf{u} = H_i(\mathbf{y})$$

### Local Control Laws

- A local control law is a triple:  $\langle \mathbf{A}, H_i, \Omega \rangle$ 
  - Applicability predicate  $\mathbf{A}(\mathbf{y})$
  - Control policy  $\mathbf{u} = H_i(\mathbf{y})$
  - Termination predicate  $\Omega(\mathbf{y})$

### A Finite State Machine

- Define a finite state machine, where
  - the states are control laws being applied,
  - the transitions are applicability and termination conditions.
- This places requirements on the dynamical systems specified by the control laws
  - to initiate and terminate appropriately.
- The finite state machine model lets us analyze global behavior.

### Applicability Predicate $\mathbf{A}(\mathbf{y})$

- Applicability must be defined in terms of the available sensory information  $\mathbf{y}(t)$ .
  - It means that the environment satisfies the assumptions needed by the control law.
- How many control laws are applicable?
  - Ideally, exactly one.
  - Zero? What do we do?
  - More than one? How do we select?

### Termination Predicate $\Omega(\mathbf{y})$

- Termination must be defined in terms of available sensory information  $\mathbf{y}(t)$ .
- Termination is not just inverse applicability.
  - It applies in a context where the control law is running: *decision inertia* gives stability.

## Interpreted FSM Model

- Assume a collection of laws:  $\langle A_i, H_i, \Omega_i \rangle$
- Repeat:
  - Identify applicable control laws.
  - Select/compose runnable control law.
    - Selection: e.g., by priorities ...
    - Composition: e.g., potential field ...
  - Run control law.
  - Terminate control law.

## Compiled FSM Model

- Perhaps the select/compose step can be compiled into a single transition condition:  
 $active(H_i) \wedge T_{ij} \rightarrow transitionTo(H_j)$   
 $T_{ij} = \Omega_i \wedge [H_j = select(\{H_k : A_k\})]$   
 $T_{ij} = \Omega_i \wedge [H_j = compose(\{H_k : A_k\})]$
- Conditions  $T_{ij}$  are mutually exclusive.

## Analyzing the Transitions

- **Safety:**
  - Do we avoid the bad states?
  - Do we avoid getting stuck (physically)?
- **Liveness:**
  - Do we reach the good states?
  - Do we avoid getting stuck (computationally)?
- How many control laws are applicable?

## Issues with FSM Model

- Applicability vs selection of control law
- Selection vs combination of control laws
- Sequential vs parallel combination
- Vector addition of control laws
- Success vs. failure termination
- Interruption vs. continuity of execution

## Alternative Approaches To Sequencers

- Jim Firby, RAPS
- Roger Brockett, MDL
  - Hristu-Varsakelis & Andersson, MDLe.
- ... there are others ...
- The right answer is not completely clear.
  - How about Lua, or Tekkotsu?

## Reactive Action Plans: RAPS

- Jim Firby's PhD thesis (Yale, 1989)
  - *Adaptive Execution in Complex Dynamic Worlds*
- Widely used by NASA and others.
- A major three-level architecture
  - Skills: continuous control
  - Sequencing: reactive behaviors
  - Planning: deliberation and inference

## Processes Cause Behavior

- Behavior comes from interacting processes acting on input from the environment.
- Behavior is controlled by enabling and disabling the processes.
- Processes detect conditions (good and bad) and send asynchronous signals.

## RAP: A Set of Methods for Accomplishing a Task

- Task name and arguments
- Success (termination) condition
- Multiple methods (OR)
  - Context (applicability) conditions
  - Network of subtasks (AND)
- A RAP is an AND/OR tree,
  - plus sequencing and signals

## Example RAP

```
(define-rap (arm-pickup ?arm ?thing)
  (succeed (ARM-HOLDING ?arm ?thing))
  (method
    (context (not (TOOL-NEEDED ?thing ?tool true)))
    (task-net
      (t1 (arm-move-to ?arm ?thing) (for t2))
      (t2 (arm-grasp-thing ?arm ?thing))))
  (method
    (context (TOOL-NEEDED ?thing ?tool true))
    (task-net
      (t1 (arm-pickup ?arm ?tool) (for t2))
      (t2 (arm-move-to ?arm ?thing) (for t3))
      (t3 (arm-grasp-thing ?arm ?thing))))))
```

## Problems with Simple RAPs

- Assumption: Tasks are *atomic*.
  - One subtask is processed at a time.
  - Each task either succeeds or fails.
  - Success or failure is known.
  - Success or failure is well defined.
- Extend to concurrency and arbitrary signals.

## Waiting for a Signal

```
(task-net
  (t1 (approach-target ?target)
    (wait-for (at-target) :proceed)
    (wait-for (stuck) :terminate)))
```

- Initiate subtask behavior.
- Wait for problem-specific signals to arrive.
- Send different signals for success and failure.

## Concurrent Tasks

- Initiate concurrent threads of execution.
- Annotations to coordinate threads:
  - **until-end**: stop this task when another ends.
  - **until-start**: stop this task when another starts.
  - **wait-for**: pause until signal is received.
- Synchronize task processing with progress in the real world.

## Concurrent Task Net

```
(task-net
  (t1 (approach-target ?target)
      (wait-for (at-target) :proceed)
      (wait-for (stuck) :terminate))
  (t2 (track-target ?target)
      (wait-for (lost-target) :terminate)
      (wait-for (camera-problem) :terminate)
      (until-end t1)))
```

**until-end** means that when t1 terminates,  
process t2 is also terminated.

That is, track the target until we reach it, or fail.

## Enabling and Cleanup Tasks

- Success/failure is too limited.
  - Failure means terminate all subtasks.
  - Some subtasks (cleanup) should take place whether the main task has succeeded or failed.
- Must be able to express an arbitrary signal-driven transition graph among subtasks.

## Complex Task Net with Cleanup

```
(task-net
  (t0 (camera-on) (wait-for :success t1) (for t2))
  (t1 (approach-target ?target)
      (wait-for (at-target) t3)
      (wait-for (stuck) t3)
      (until-start t3))
  (t2 (track-target ?target)
      (wait-for (lost-target) t3)
      (wait-for (camera-problem) :terminate)
      (until-start t3))
  (t3 (camera-off)))
```

**until-start** means if either t1 or t2 starts t3,  
the other is terminated.

## The Sequencing Level

- Goal: Factor a complex task into a discrete set of interacting continuous skills.
- The RAP task net
  - Finite state transition graph.
  - AND/OR graph of subtasks, with conditions.
  - Concurrent threads of task execution.
  - Synchronization of threads via signals.
- Incorporate these concepts in your code.
  - Further improvements may well be possible.