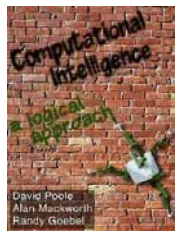
Chapter 12: Building Situated Robots

- Introduction (12.1)
- Robotics Systems (12.2)
- The Agent Function (12.3)
- Robotic Architectures (12.6)
- Implementing a Controller (12.7)
- Robots Modeling the World (12.8)



D. Poole, A. Mackworth, and R. Goebel, *Computational Intelligence: A Logical Approach*, Oxford University Press, January 1998

Introduction (12.1)

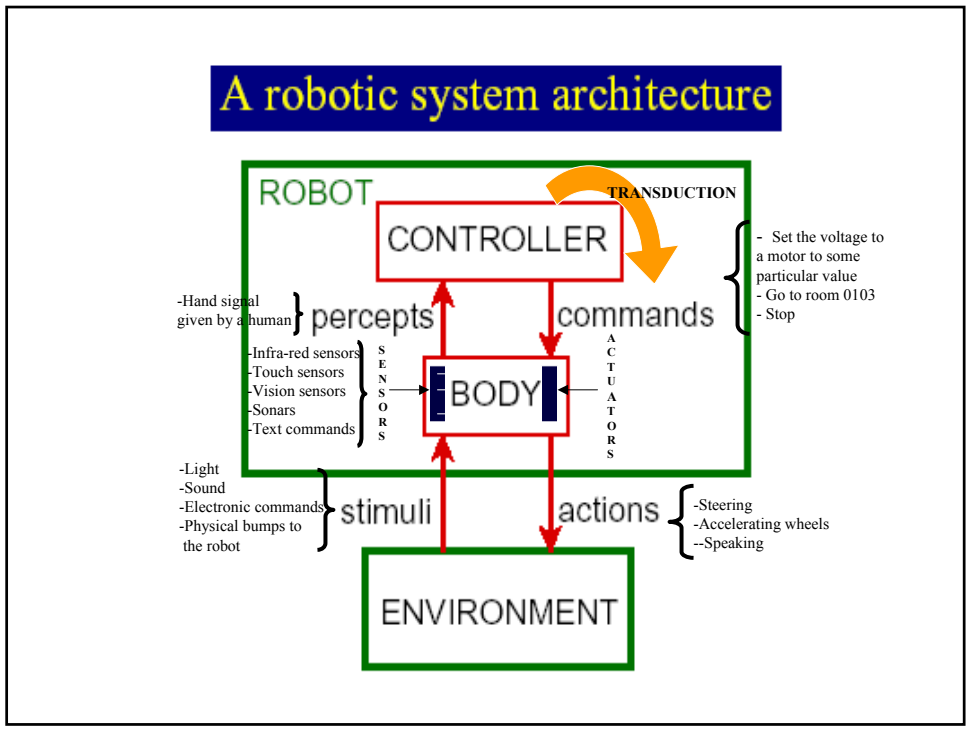
- Building Situated Robots
  - An intelligent agent perceives, reasons, and acts in time in an environment.
    - An agent is something that acts in the world.
    - A purposive agent prefers some states of the world to other states, and acts to try to achieve worlds they prefer.
    - A robot is an artificial purposive agent.

Introduction (12.1) (cont.)

- What makes an agent?
  - Agents can have sensors and effectors to interact with the environment.
  - Agents have (limited) memory and (limited) computational capabilities.
  - Agents reason and act in time.

Robotics Systems (12.2)

- A robotic system is made up of a robot and an environment.
  - A robot receives stimuli from the environment and carries out actions in the environment.
- A robot is made up of a body and a controller .
  - The controller receives percepts from the body and sends commands to the body.



The Agent Function (12.3)

- Let T be the set of time points.
- A percept trace is a function from T into P, where P is the set of all possible percepts.
- A command trace is a function from T into C, where C is the set of all commands.
- A transduction is a function from percept traces into command traces that's causal: the action trace up to time t depends only on percepts up to t.
- A controller is an implementation of a transduction.

The Agent Function (12.3) (cont.)

- States
  - A transduction specifies a function from an agent's history at time t into its action at time t.
  - An agent doesn't have access to its entire history. It only has access to what it has remembered.
  - The state of an agent at time t encodes all of the agent's history that it has access to. The state encapsulates the information about its past that it can use for current and future actions.

The Agent Function (12.3) (cont.)

- Functions implemented in a controller
  - For discrete time, a controller implements:
    - a state transition function $\sigma : S \times P \rightarrow S$, where S is the set of states and P is the set of possible percepts.

      $s_{t+1} = \sigma(s_t, p_t)$ means that s_{t+1} is the state following state s_t when p_t is observed.
    - A command function $\chi : S \times P \rightarrow C$, where S is the set of states, P is the set of possible percepts, and C is the set of possible commands.

      $c_t = \chi(s_t, p_t)$ means that the controller issues command c_t when the state is s_t and p_t is observed.

Robot Architectures (12.6)

- You don't need to implement an intelligent agent as:

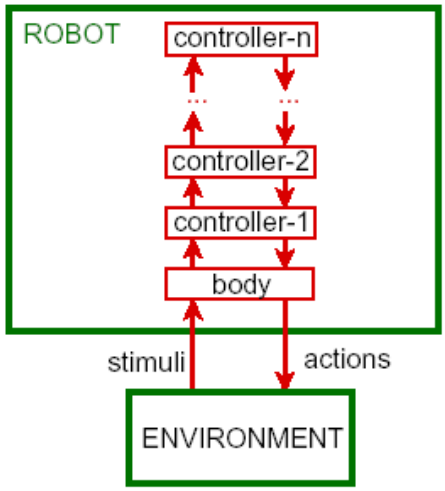
as three independent modules, each feeding into the the next.

- It's too slow.
- High-level strategic reasoning takes more time than the reaction time needed to avoid obstacles.
- The output of the perception depends on what you will do with it.

Robot Architectures (12.6) (cont.)

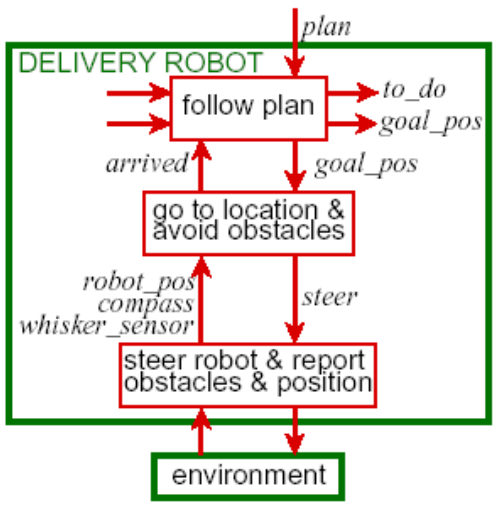
- Hierarchical Control
  - A better architecture is a hierarchy of controllers.
  - Each controller sees the controllers below it as a virtual body from which it gets percepts and sends commands.
  - The lower-level controllers can
    - run much faster, and react to the world more quickly
    - deliver a simpler view of the world to the higher-level controllers.

Hierarchical Robotic System Architecture



Implementing a Controller (12.7)

A Decomposition of the Delivery Robot



Implementing a Controller (12.7) (cont.)

- Axiomatizing a Controller
  - A fluent is a predicate whose value depends on the time.
  - We specify state changes using *assign(Fl; Val; T)* which means fluent *Fl* is assigned value *Val* at time *T*.
  - *was* is used to determine a fluent's previous value. *was(Fl; Val; T_1; T)* is true if fluent *Fl* was assigned a value at time T_1, and this was the latest time it was assigned a value before time *T*.
  - *val(Fl; Val; T)* is true if fluent *Fl* was assigned value *Val* at time *T* or *Val* was its value before time *T*.

Implementing a Controller (12.7) (cont.)

- Middle Layer of the Delivery Robot

Steer(D, T) means that the robot will steer in direction D at time T, where D ∈ {left, straight, right}.
The robot steers towards the goal, except when the whisker sensor is on, in which case it turns left:

Steer(left, T) ← whisker_sensor(on, T):
Steer(D, T) ← goal_is(D, T) ∧ ~whisker_sensor(on, T):

goal_is(D, T) means the goal is in direction D from the robot.

goal_is(left, T) ←
goal_direction(G; T) ∧ val(compass, C, T) ∧
(G - C + 540) mod 360 - 180 > 11.

Implementing a Controller (12.7) (cont.)

- Middle Layer of the Delivery Robot (cont.)

This level needs to tell the higher level when it has arrived. *arrived(T)* is true if the robot has arrived at, or is close enough to, the previous goal position:

$$
\begin{aligned}
&arrived(T) \leftarrow \\
&\quad was(goal\_pos, Goal\_Coords, T_0, T) \wedge \\
&\quad robot\_pos(Robot\_Coords, T) \wedge \\
&\quad close\_enough(Goal\_Coords, Robot\_Coords). \\
&close\_enough((X_0, Y_0), (X_1, Y_1)) \leftarrow \\
&\quad \sqrt{(X_1 - X_0)^2 + (Y_1 - Y_0)^2} < 3.0.
\end{aligned}
$$

Here 3:0 is an arbitrarily chosen threshold.

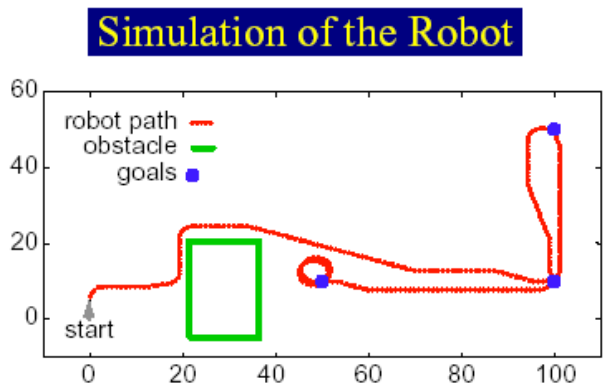
Implementing a Controller (12.7) (cont.)

- Top Layer of the Delivery Robot

The top layer has two state variables represented as fluents.
The value of the fluent *to_do* is the list of all pending actions.
The fluent *goal_pos* maintains the goal position.

$$
\begin{aligned}
&assign(goal\_pos, Coords, T) \leftarrow \\
&\quad arrived(T) \wedge \\
&\quad was(to\_do, [goto(Loc)|R], T_0, T) \wedge \\
&\quad at(Loc, Coords). \\
&assign(to\_do, R, T) \leftarrow \\
&\quad arrived(T) \wedge \\
&\quad was(to\_do, [C|R], T_0, T).
\end{aligned}
$$

Implementing a Controller (12.7) (cont.)



assign(*to_do*, [*goto*(*o*109), *goto*(*storage*), *goto*(*o*109), *goto*(*o*103)], 0).

arrived(1).

Robots Modeling the World (12.8)

- What should be in an agent's state?
  - An agent decides what to do based on its state and what it observes.
  - A purely reactive agent doesn't have a state.
    A dead reckoning agent doesn't perceive the world.
    - neither work very well in complicated domains.
  - It is often useful for the agent to model the world (itself and the environment).
  - In active perception a prediction of the world is combined with perception to give an updated world model.