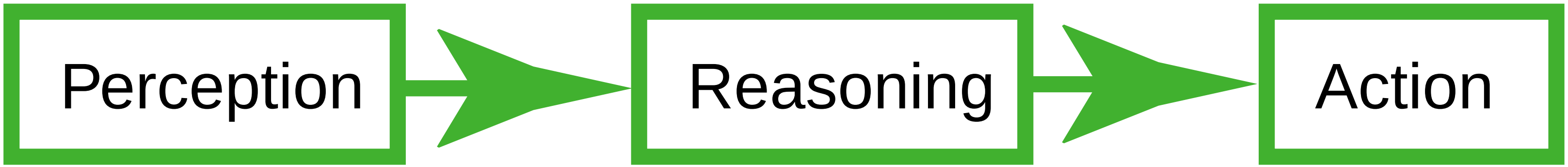
Robot Architectures

You don’t need to implement an intelligent agent as:



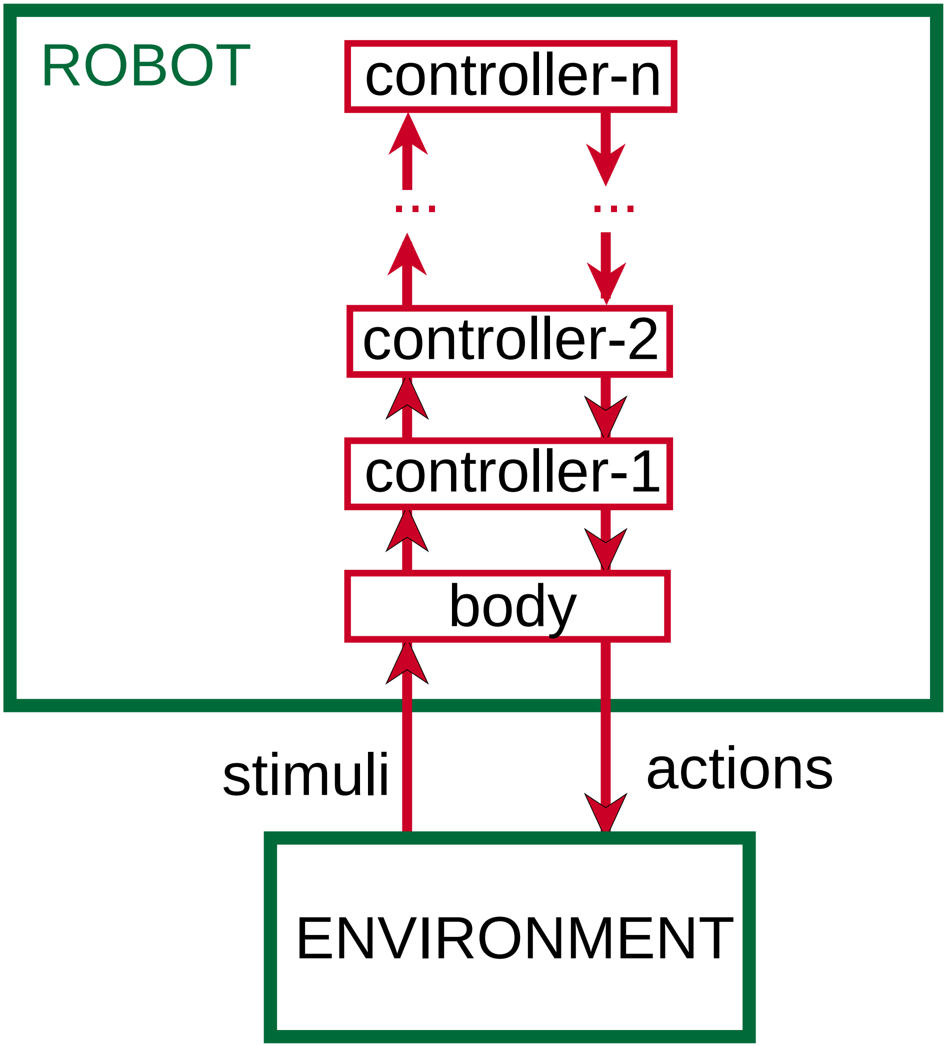
as three independent modules, each feeding into the the next.

- 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.

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.

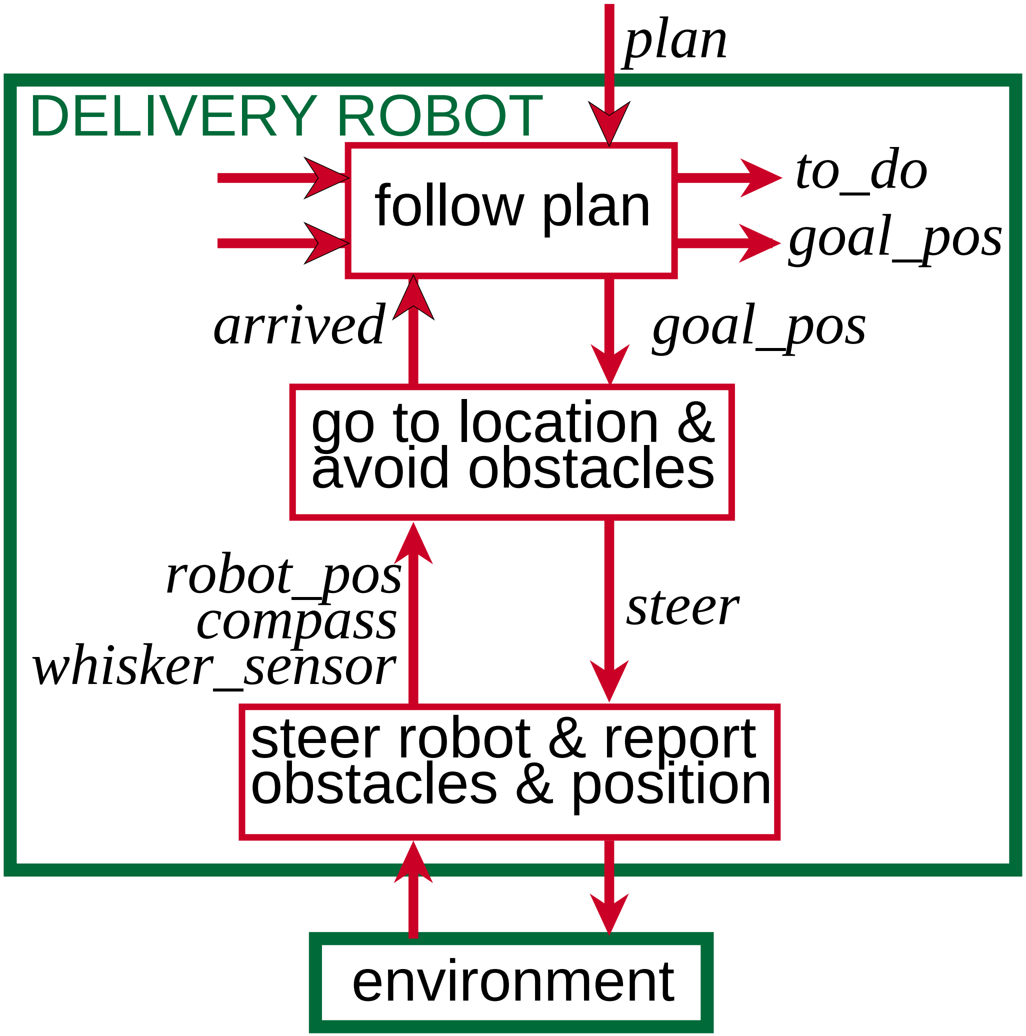
Hierarchical Robotic System Architecture



Example: delivery robot

- The robot has three actions: go straight, go right, go left. (Its velocity doesn't change).
- It can be given a plan consisting of sequence of named locations for the robot to go to in turn.
- The robot must avoid obstacles.
- It has a single whisker sensor pointing forward and to the right. The robot can detect if the whisker hits an object. The robot knows where it is.
- The obstacles and locations can be moved dynamically. Obstacles and new locations can be created dynamically.

A Decomposition of the Delivery Robot



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.

Middle Layer of the Delivery Robot

- Higher layer gives a goal position
  - Head towards the goal position:
    - If the goal is straight ahead (within an arbitrary threshold of $\pm 11°$), go straight
    - If the goal is to the right, go right
    - If the goal is to the left, go left
- Avoid obstacles:
  - If the whisker sensor is on, turn left
- Report when arrived

Code for the middle layer

$steer(D, T)$ means that the robot will steer in direction D at time T, where $D \in \{left, straight, right\}$.

The robot steers towards the goal, except when the whisker sensor is on, in which case it turns left:

$$steer(left, T) \leftarrow whisker\_sensor(on, T).$$

$$steer(D, T) \leftarrow whisker\_sensor(off, T) \wedge goal\_is(D, T)$$

$goal\_is(D, T)$ means the goal is in direction D from the robot.

$$goal\_is(left, T) \leftarrow$$
$$goal\_direction(G, T) \wedge val(compass, C, T) \wedge$$
$$(G - C + 540) \bmod 360 - 180 > 11.$$

Middle layer (continued)

This layer needs to tell the higher layer when it has arrived.

$arrived(T)$ is true if the robot has arrived at, or is close enough to, the (previous) goal position:

$$
\begin{aligned}
&arrived(T) \leftarrow \\
&\quad was(goal\_pos, Goal\_Coords, T_0, T) \wedge \\
&\quad robot\_pos(Robot\_Coords, T) \wedge \\
&\quad close\_enough(Goal\_Coords, Robot\_Coords).
\end{aligned}
$$

$$
\begin{aligned}
&close\_enough((X_0, Y_0), (X_1, Y_1)) \leftarrow \\
&\quad \sqrt{(X_1 - X_0)^2 + (Y_1 - Y_0)^2} < 3.0.
\end{aligned}
$$

Here 3.0 is an arbitrarily chosen threshold.

Top Layer of the Delivery Robot

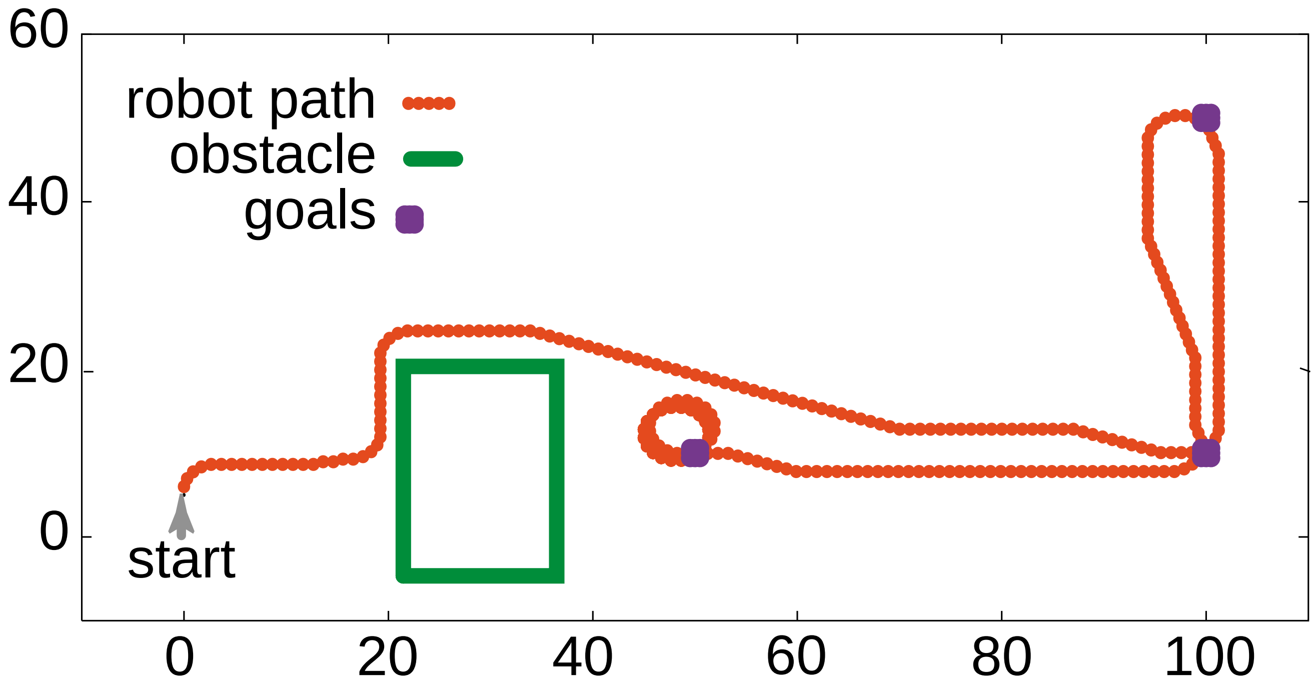
- The top layer is given a plan which is a sequence of named locations.
- The top layer tells the middle layer the goal position of the current location.
- It has to remember the current goal position and the locations still to visit.
- When the middle layer reports the robot has arrived, the top layer takes the next location from the list of positions to visit, and there is a new goal position.

Code for the top layer

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

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

Simulation of the Robot



$$assign(to\_do, [goto(o109), goto(storage), goto(o109),$$
$$goto(o103)], 0).$$

$$arrived(1).$$

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's belief state to be a model of the world (itself and the environment).