Agent Architectures

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

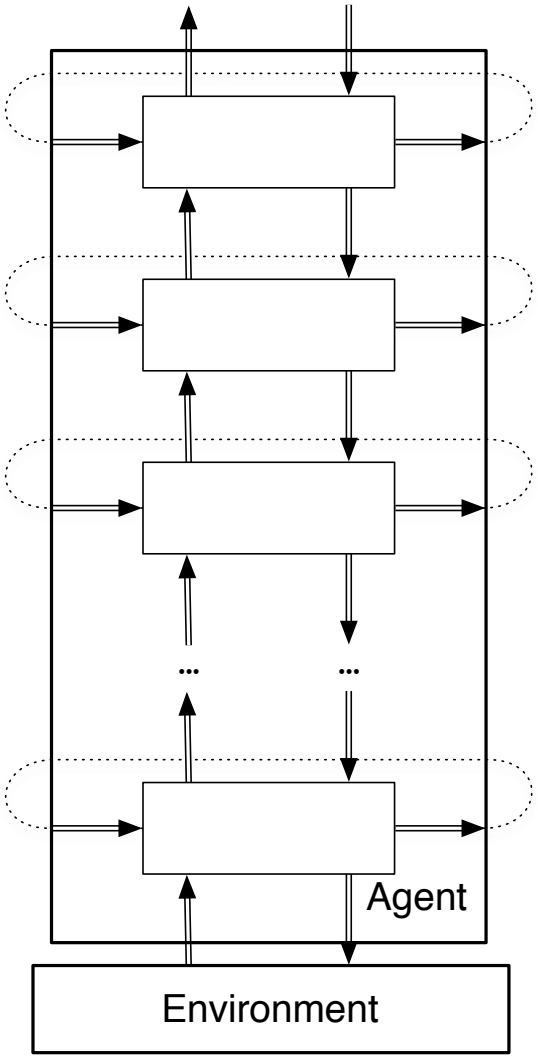
as three independent modules, each feeding into the the next.

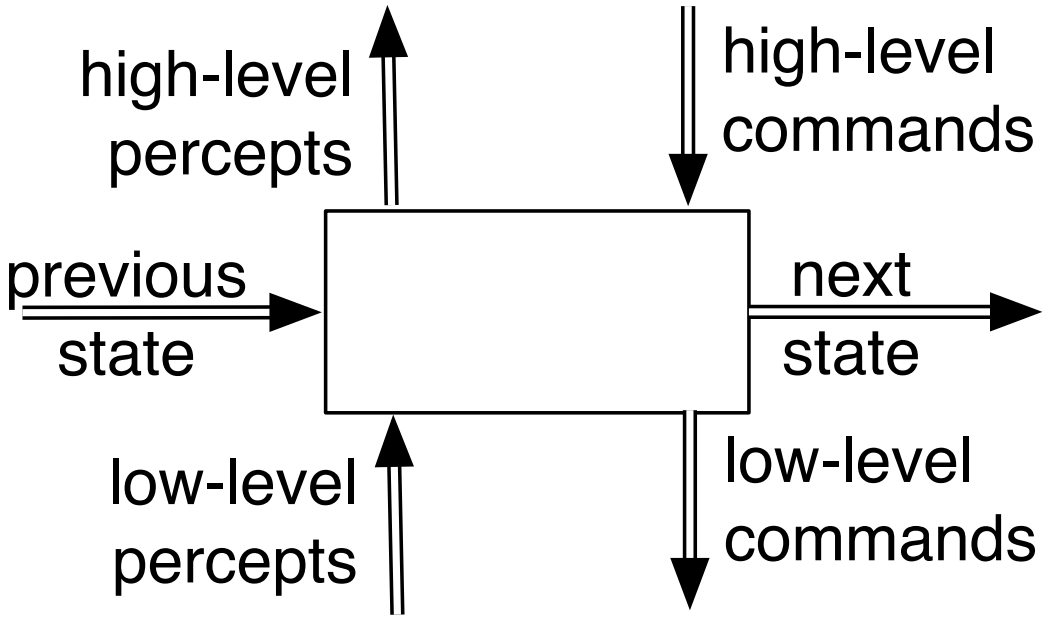
- It's too slow.
- High-level strategic reasoning takes more time than the reaction time needed to avoid obstacles.
- The output of the perception depends on what you will do with it.

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





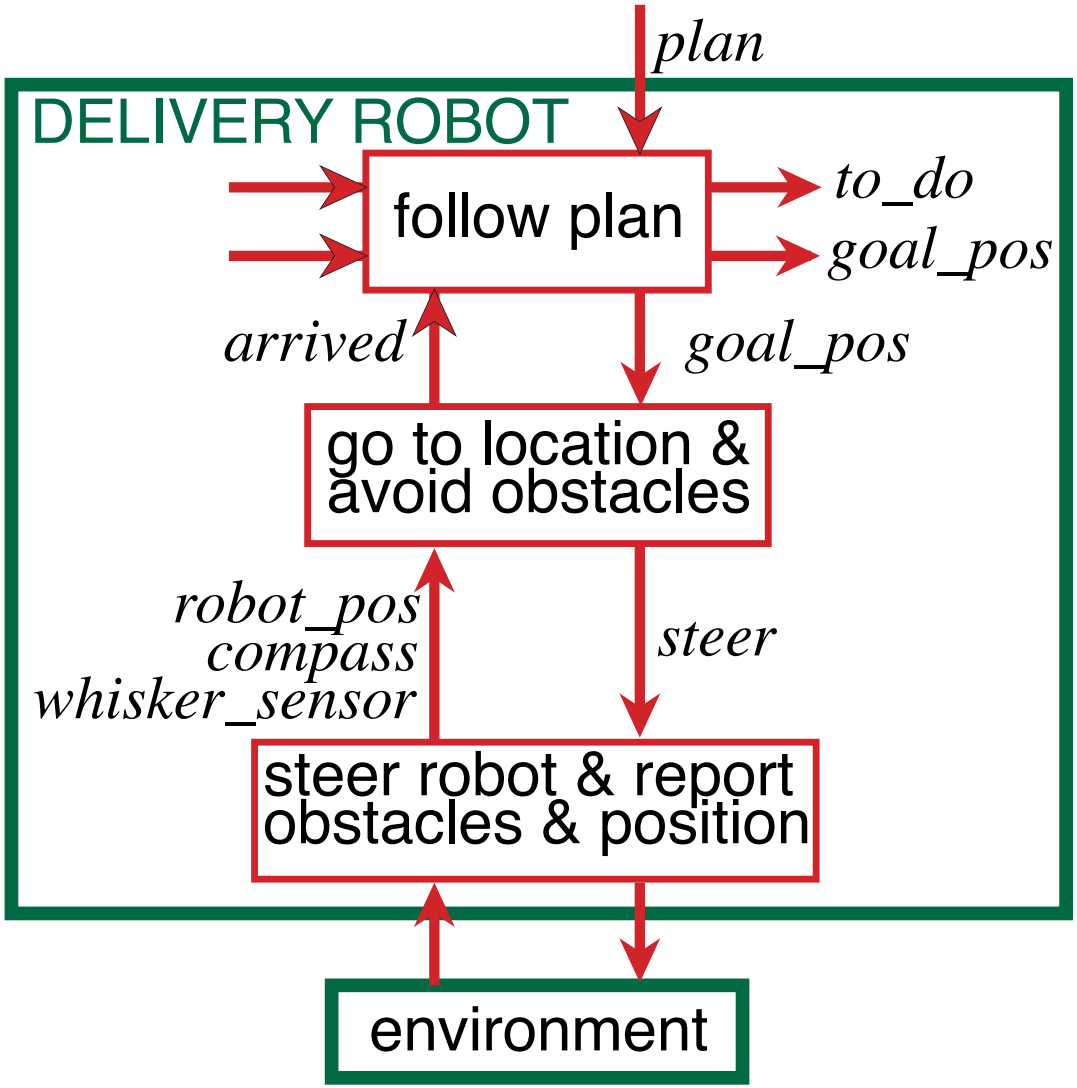
Inputs/Outputs of layers

- There are three types of inputs to each layer at each time:
  - the features that come from the belief state, which are referred to as the remembered or previous values of these features;
  - the features representing the percepts from the layer below in the hierarchy; and
  - the features representing the commands from the layer above in the hierarchy.
- There are three types of outputs from each layer at each time:
  - the higher-level percepts for the layer above,
  - the lower-level commands for the layer below, and
  - the next values for the belief-state features.
- An implementation of a layer specifies how the outputs of a layer are a function of its inputs.

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



Top Layer

- The top layer takes in a plan to execute. The plan is a list of named locations to visit in order.
- The locations are selected in order. Each selected location becomes the current target.
- This layer determines the x-y coordinates of the target. These coordinates are the target position for the lower level.
- The upper level knows about the names of locations, but the lower levels only know about coordinates.
- The top layer maintains a belief state consisting of a list of names of locations that the robot still needs to visit and the coordinates of the current target.
- It issues commands to the middle layer in terms of the coordinates of the current target.

Middle Layer

- It tries to keep traveling toward the current target position, avoiding obstacles.
- The target position is obtained from the top layer.
- When the middle layer has arrived at the target position, it signals to the top layer that it has achieved the target.
- When notified, the top layer then changes the target position to the coordinates of the next location on the plan.
- The middle layer uses a simple strategy of trying to head toward the target unless it is blocked, in which case it turns left.
- The middle layer is built on a lower layer that provides a simple view of the robot: it takes in steering commands and reports the robot's position, orientation, and whether the sensor is on or off.

Code for the Middle Layer of the Delivery Robot

if $whisker\_sensor = on$

then $steer = left$

else if $straight\_ahead(robot\_pos, robot\_dir, current\_goal\_pos)$

then $steer = straight$

else if $left\_of(robot\_position, robot\_dir, current\_goal\_pos)$

then $steer = left$

else $steer = right$

$$arrived = distance(previous\_goal\_pos, robot\_pos) < threshold$$

Code for the Top Layer of the Delivery Robot

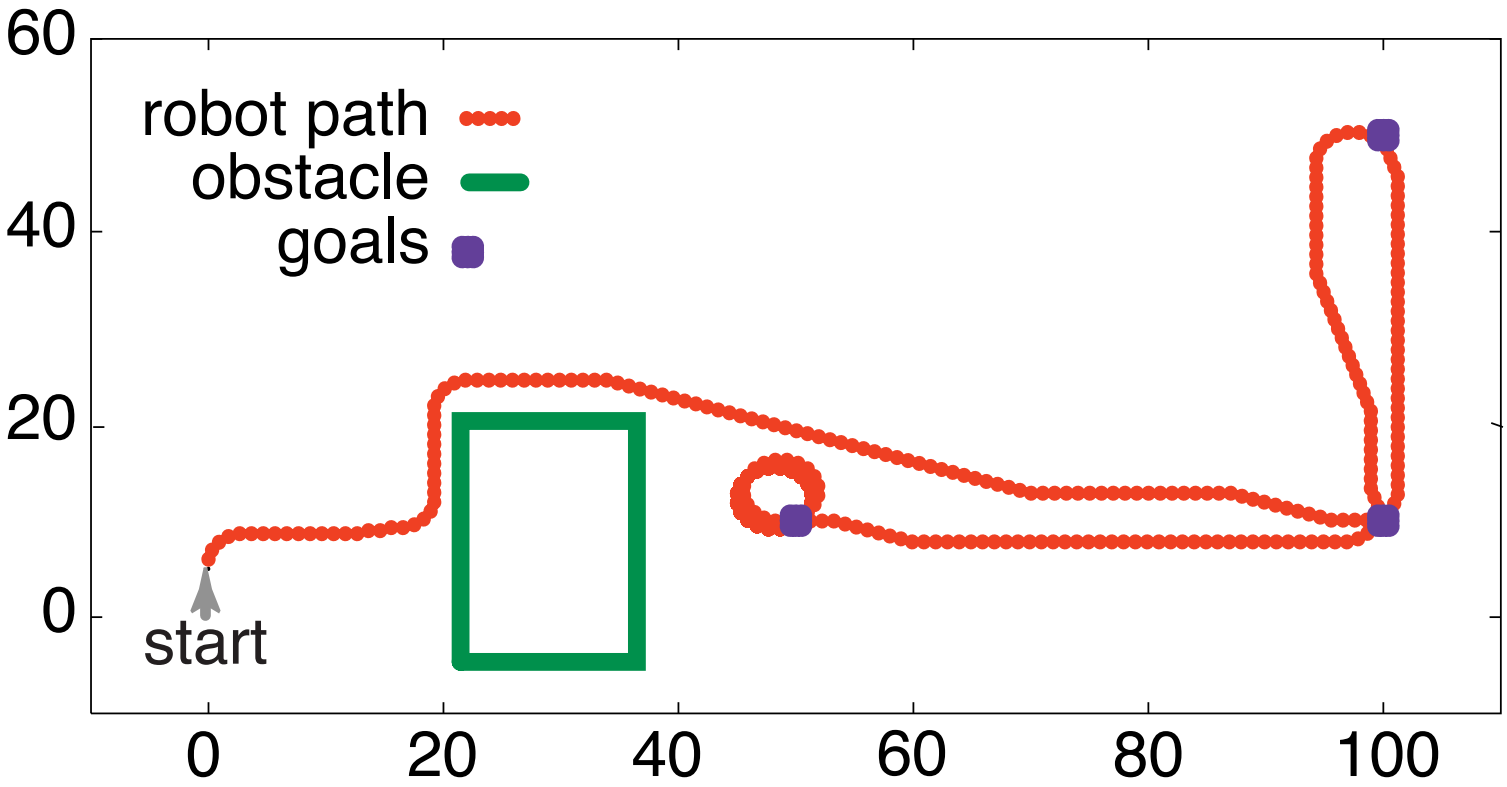
The top layer has two belief state variables:

- *to_do* is the list of all pending locations
- *goal_pos* is the current goal position

if *arrived*
then $goal\_pos = coordinates(head(to\_do'))$.
if *arrived*
then $to\_do = tail(to\_do')$.

Here to_do' is the previous value for the *to_do* feature.

Simulation of the Robot



$to\_do = [goto(o109), goto(storage), goto(o109),$
$goto(o103)]$
$arrived = true$

What should be in an agent's belief state?

- An agent decides what to do based on its belief state and what it observes.
- A purely reactive agent doesn't have a belief 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).

Embedded and Simulated Agents

- An embedded agent is one that is run in the real world, where the actions are carried out in a real domain and where the sensing comes from a domain.
- A simulated agent is one that is run with a simulated body and environment; that is, where a program takes in the commands and returns appropriate percepts. This is often used to debug a controller before it is deployed.
- A agent system model is where there are models of the controller (which may or may not be the actual code), the body, and the environment that can answer questions about how the agent will behave. Such a model can be used to prove properties of agents before they are built, or it can be used to answer hypothetical questions about an agent that may be difficult or dangerous to answer with the real agent.