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

- Perception → Reasoning → Action

as three independent modules, each feeding into 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

Environment

Agent

previous memories

high-level percepts

low-level percepts

previous memories

next memories

low-level commands

high-level commands
Functions implemented in a layer

- **memory function**
  \[
  \text{remember}(\text{memory, percept, command})
  \]

- **command function**
  \[
  \text{do}(\text{memory, percept, command})
  \]

- **percept function**
  \[
  \text{higher_percept}(\text{memory, percept, command})
  \]
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

DELIVERY ROBOT

follow plan

plan

to_do

goal_pos

arrived

goal_pos

go to location & avoid obstacles

robot_pos

compass

steer

steer robot & report obstacles & position

environment

©D. Poole and A. Mackworth 2010
Artificial Intelligence, Lecture 2.2, Page 6
Middle Layer

Go to target and avoid obstacles

robot position
robot orientation
whisker sensor
steer

arrived

previous target-pos
current target-pos

steer

arrived

Go to target and avoid obstacles
if \(\text{whisker\_sensor} = \text{on}\) 
then \(\text{steer} = \text{left}\)
else if \(\text{straight\_ahead}(\text{robot\_pos}, \text{robot\_dir}, \text{current\_goal\_pos})\) 
then \(\text{steer} = \text{straight}\)
else if \(\text{left\_of}(\text{robot\_position}, \text{robot\_dir}, \text{current\_goal\_pos})\) 
then \(\text{steer} = \text{left}\)
else \(\text{steer} = \text{right}\)

\[\text{arrived} = \text{distance}(\text{previous\_goal\_pos}, \text{robot\_pos}) < \text{threshold}\]
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.
Top Layer

- previous
- to_do
- previous
- target_pos
- follow plan
- to_do
- target_pos
- arrived
- plan
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).