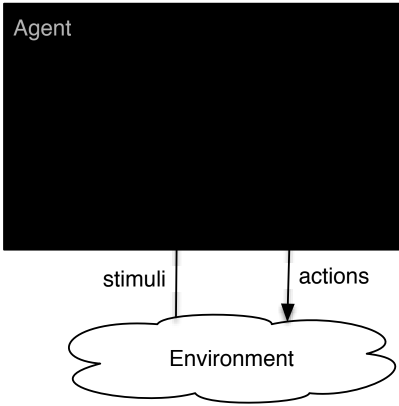
Agent architectures and hierarchical control

Overview:

- Agents and Robots
- Agent systems and architectures
- Agent controllers
- Hierarchical controllers

Agent Systems

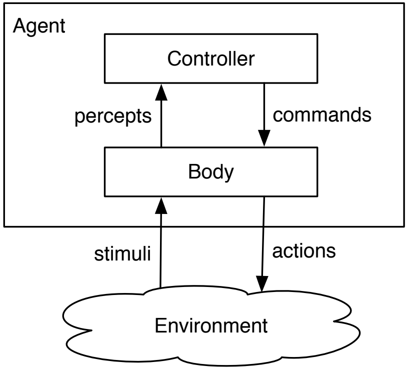


A agent system is made up of an agent and an environment.

- An agent receives stimuli from the environment
- An agent carries out actions in the environment.

Agent System Architecture

An agent is made up of a body and a controller.



- An agent interacts with the environment through its body.
- The body is made up of:
  - sensors that interpret stimuli
  - actuators that carry out actions
- The controller receives percepts from the body.
- The controller sends commands to the body.
- The body can also have reactions that are not controlled.

Implementing a controller

- A controller is the brains of the agent.
- Agents are situated in time, they receive sensory data in time, and do actions in time.
- Controllers have (limited) memory and (limited) computational capabilities.
- The controller specifies the command at every time.
- The command at any time can depend on the current and previous percepts.

Example: smart home

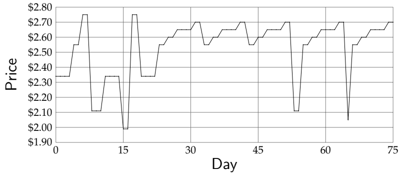
- A smart home will monitor your use of essentials, and buy them before you run out.
  Example: snack buying agent:
  - abilities: buy chips (and have them delivered)
  - goals: mimimize price, don't run out of chips
  - stimuli: price, number in stock
  - prior knowledge: range of prices, consumption rates

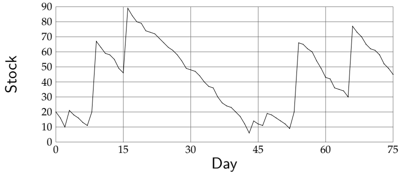
Controllers

- A percept trace is a sequence of all past, present, and future percepts received by the controller.
- A command trace is a sequence of all past, present, and future commands output by the controller.
- An agent's history at time t is sequence of past and present percepts and past commands.
- A transduction specifies a function from an agent's history at time t into its command at time t.
- A controller is an implementation of a transduction.

The Agent Functions

- A percept trace is a sequence of all past, present, and future percepts received by the controller.





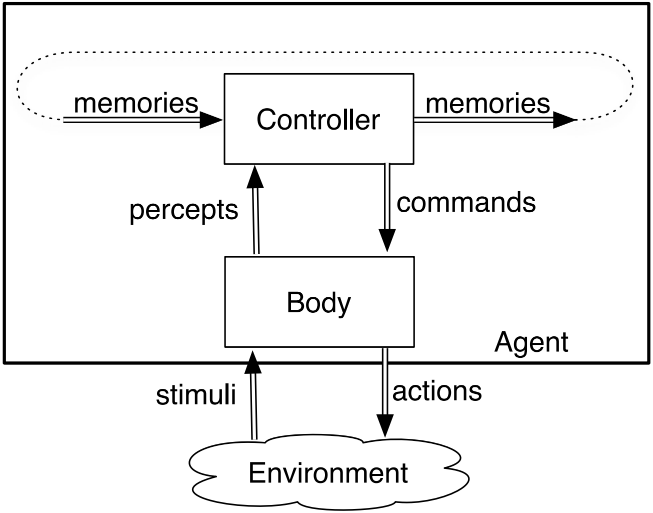
- A command trace is a sequence of all past, present, and future commands output by the controller.

Belief States

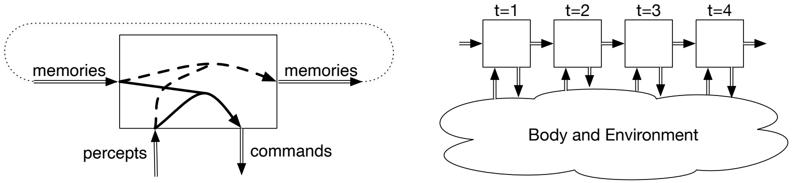
- An agent doesn't have access to its entire history. It only has access to what it has remembered.
- The memory or belief state of an agent at time t encodes all of the agent's history that it has access to.
- The belief state of an agent encapsulates the information about its past that it can use for current and future actions.
- At every time a controller has to decide on:
  - What should it do?
  - What should it remember?
    (How should it update its memory?)

  — as a function of its percepts and its memory.

Controller


memories
Controller
memories
percepts
commands
Body
Agent
stimuli
actions
Environment

Functions implemented in a controller



(a) (b)

For discrete time, a controller implements:

- belief state function *remember*(*belief_state*, *percept*), returns the next belief state.
- command function *command*(*belief_state*, *percept*) returns the command for the agent.

Snack buying controller

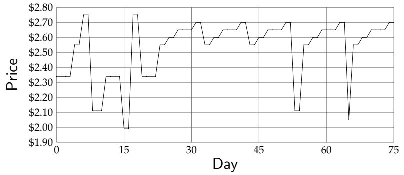
- Percepts: price, number in stock
- Action: number to buy
- Belief state: (approximate) running average
- Command function:
  - if $price < 0.9 * average$ and $instock < 60$ buy 48
  - else if $instock < 12$ buy 12
  - else buy 0
- Belief state transition function:

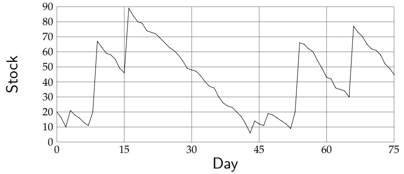
$$average := average + (price - average) * 0.05$$

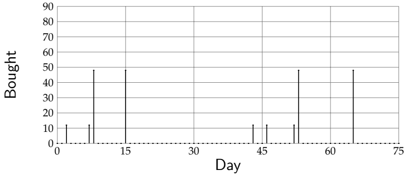
This maintains a discouning rolling avergage that (eventually) weights more recent prices more.

(see `agents.py` in AIPython distribution `http://aipython.org`)

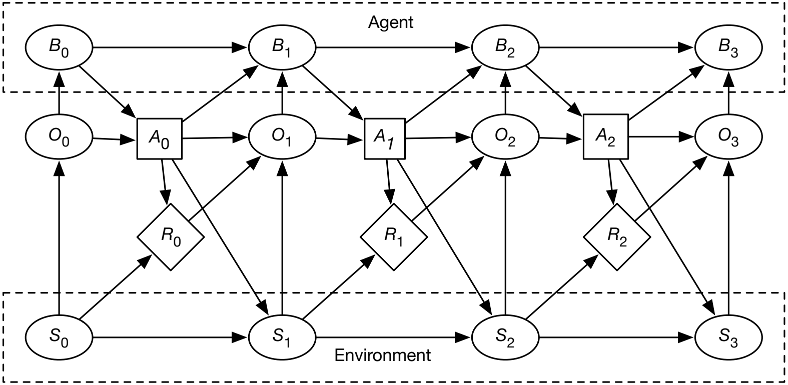
Percept and Command Traces (POMDP)







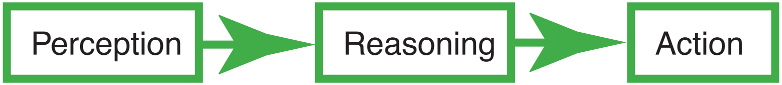
Agents acting in time



B_i agent's belief state at time i.A_i agent's action.O_i is what the agent observes.R_i is the reward.S_i is the world state.

Agent Architectures

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



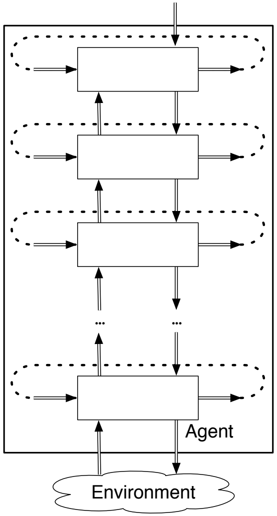
as three independent modules, each feeding into the the next.

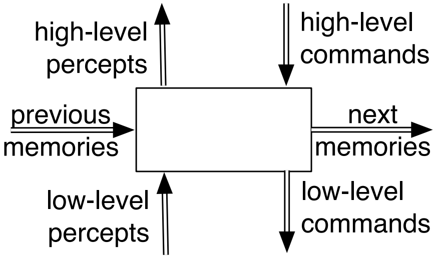
- It's too slow.
- High-level strategic reasoning takes more time than the reaction time needed (e.g. to avoid obstacles).
- The output of the perception depends on what you will do with it.

Hierarchical Control

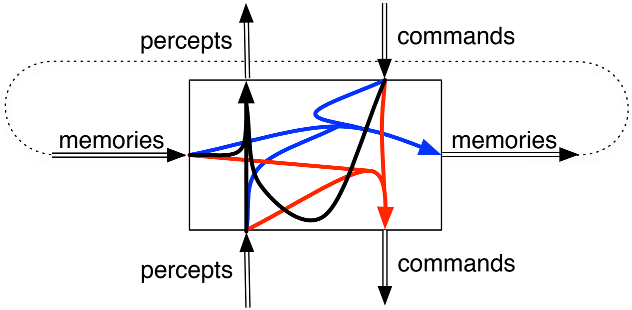
- A better architecture is a hierarchy of controllers.
- Each controller sees the controllers below it as a virtual body from which it gets percepts and sends commands.
- The lower-level controllers can
  - run much faster, and react to the world more quickly
  - deliver a simpler view of the world to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

Hierarchical Robotic System Architecture





Functions implemented in a layer

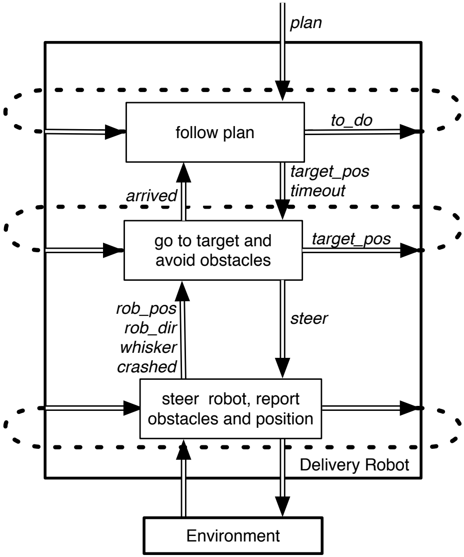


- memory function $remember(memory, percept, command)$
- command function
  $do(memory, percept, command)$
- percept function $higher\_percept(memory, percept, command)$

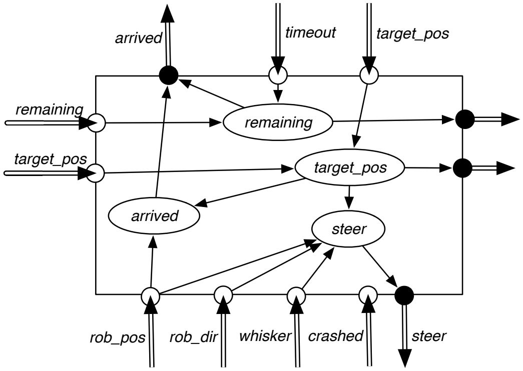
Example: delivery robot

- 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



Middle Layer



Middle Layer of the Delivery Robot

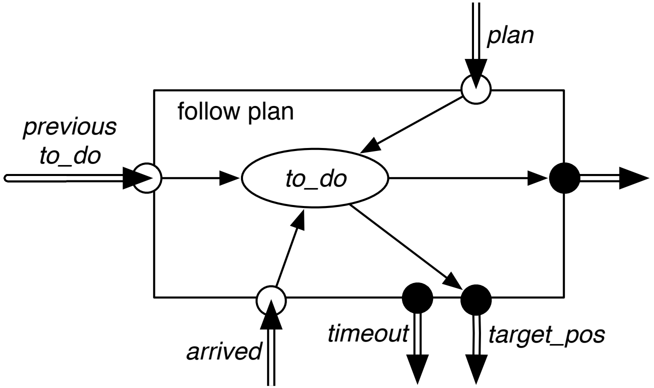
```
given $timeout$ and $target\_pos$:
    $remaining := timeout$
    while not $arrived()$ and $remaining \neq 0$
        if $whisker\_sensor = on$
            then $steer := left$
        else if $straight\_ahead(rob\_pos, robot\_dir, target\_pos)$
            then $steer := straight$
        else if $left\_of(rob\_pos, robot\_dir, target\_pos)$
            then $steer := left$
        else $steer := right$
        $do(steer)$
        $remaining := remaining - 1$
    tell upper layer $arrived()$

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

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.

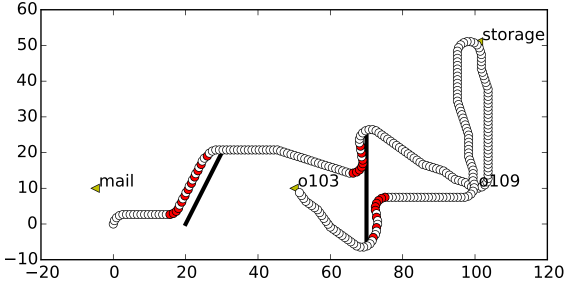
Top Layer


plan
follow plan
previous to_do
to_do
arrived
timeout
target_pos

Code for the top layer

```
given $plan$:
    $to\_do := plan$
    $timeout := 200$
    while not $empty(to\_do)$
            $target\_pos := coordinates(first(to\_do))$
            $do(timeout, target\_pos)$
            $to\_do := rest(to\_do)$
```

Simulation of the Robot



Robot starts at $(0, 0)$ facing up.

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

Red = whisker sensor on

(Run commands at the bottom of `agentTop.py` in AIPython.org)

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).

Moral Machines

" . . . self-driving cars . . . The technology is essentially here. We have machines that can make a bunch of quick decisions that could drastically reduce traffic fatalities, drastically improve the efficiency of our transportation grid, and help solve things like carbon emissions that are causing the warming of the planet. But . . . what are the values that we're going to embed in the cars? There are gonna be a bunch of choices that you have to make, the classic problem being: If the car is driving, you can swerve to avoid hitting a pedestrian, but then you might hit a wall and kill yourself. It's a moral decision, and who's setting up those rules?"

– Barack Obama, 2016

Moral Machines

- Trolley problems are classical experiments in moral choice. People have to decide on which hypothetical scenario for a runaway trolley (streetcar / tram) is preferred.
- In a modern variant, the moral machines experiment asked millions of people from 233 countries about what autonomous vehicles (self-driving cars) should do in various circumstances.
  `https://www.moralmachine.net`
  `https://dx.doi.org/10.1038/s41586-018-0637-6`
- Suppose there is a self-driving car with sudden brake failure, and it has to choose:
  - It can go straight ahead which will result in the death of a man and a baby who are flouting the law by crossing on a red signal.
  - It can swerve which will result in the death of a pregnant woman who was abiding by the law.

  What should it do?

Moral Machines

Alternative scenarios included

- number of deaths
- people versus animals,
- men versus women,
- young versus old
- lawful versus unlawful,
- fit versus unfit.
- social status (eg, doctor versus homeless drug addict)

Some preferences seemed universal (across cultures):

- preference for humans over animals
- fewer lives over more lives
- young over old

Some were culturally specific.

Moral Machines

We can embrace the challenges of machine ethics as a unique opportunity to decide, as a community, what we believe to be right or wrong; and to make sure that machines, unlike humans, unerringly follow these moral preferences.

– Awad at al., [2018]

Issues:

- Some principles are universal, but some are culturally specific.
- The outcomes were all well defined; there was no uncertainty.
- Some outcomes will tend to be controversial to implement; E.g., people thought it was more important to save pedestrians than to save people in the vehicle; so drivers should not be the ones giving the preferences for their car!