Agent architectures and hierarchical control

Overview:
- Agents and Robots
- Agent systems and architectures
- Agent controllers
- Hierarchical controllers
Example: smart house

A smart house will monitor your use of essentials, and buy them before you run out.
Example: snack buying agent that ensures you have a supply of chips:

- **abilities**: buy chips (and have them delivered)
- **goals**:
- **stimuli**:
- **prior knowledge**:
A agent system is made up of a agent and an environment.

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

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

![Percept Trace Diagram]

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

![Command Trace Diagram]
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.
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.
- A transduction is a function from percept traces into command traces.
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.
- A **transduction** is a function from percept traces into command traces.
- A transduction is **causal** if the command trace up to time $t$ depends only on percepts up to $t$. 
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.
- **A transduction** is a function from percept traces into command traces.
- A transduction is **causal** if the command trace up to time $t$ depends only on percepts up to $t$.
- **A controller** is an implementation of a causal 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.
- A **transduction** is a function from percept traces into command traces.
- A transduction is **causal** if the command trace up to time $t$ depends only on percepts up to $t$.
- A **controller** is an implementation of a causal transduction.
- An agent’s **history** at time $t$ is sequence of past and present percepts and past commands.
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.
- A **transduction** is a function from percept traces into command traces.
- A transduction is **causal** if the command trace up to time $t$ depends only on percepts up to $t$.
- A **controller** is an implementation of a causal transduction.
- An agent’s **history** at time $t$ is sequence of past and present percepts and past commands.
- A causal 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.
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.
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.
For discrete time, a controller implements:

- **belief state function** `remember(belief_state, percept)`, returns the next belief state.
- **command function** `command(memory, percept)` returns the command for the agent.
Example: smart house

- A smart house 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:** minimize price, don’t run out of chips
- **stimuli:** price, number in stock
- **prior knowledge:** ??

- Percept trace:
- Control trace:
- Transduction:
- Belief state:
- Belief state transition function:
- Control Function:
Implemented Example

- Percepts: price, number in stock
- Action: number to buy
- Belief state: (approximate) running average
- controller:
  - if \( \text{price} < 0.9 \times \text{average} \) and \( \text{instock} < 60 \) buy 48
  - else if \( \text{instock} < 12 \) buy 12
  - else buy 0

Belief state transition function:
\[
\text{average} := \text{average} + (\text{price} - \text{average}) \times 0.05
\]
This maintains a discounting rolling average that (eventually) weights more recent prices more.

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

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

  $$average := average + (price - average) \times 0.05$$

This maintains a discounting rolling average that (eventually) weights more recent prices more.

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

- Percepts: price, number in stock
- Action: number to buy
- Belief state: (approximate) running average
- controller:
  - if \( \text{price} < 0.9 \times \text{average} \) and \( \text{instock} < 60 \) buy 48
  - else if \( \text{instock} < 12 \) buy 12
  - else buy 0
- Belief state transition function:

  \[
  \text{average} := \text{average} + (\text{price} - \text{average}) \times 0.05
  \]

  This maintains a discounting rolling average that (eventually) weights more recent prices more.

(see agents.py in AI-Python distribution http://aipython.org)