G54DIA: Designing Intelligent Agents

Lecture 3: Reactive Architectures I

Natasha Alechina
School of Computer Science
nza@cs.nott.ac.uk

Outline of this lecture

- role of agent architectures
- kinds of agent architectures
- simple reactive architectures
- examples
  - Braitenberg vehicles
  - Boids
- advantages and disadvantages of simple reactive architectures

Importance of architecture

- focus of this module will mostly be on *agent architectures*:
  - what sorts of architectures there are; and
  - which architectures are appropriate for different tasks and environments
- *to program an agent which is successful in a given task environment, we must choose an architecture which is appropriate for that task environment*

The architecture as a virtual machine

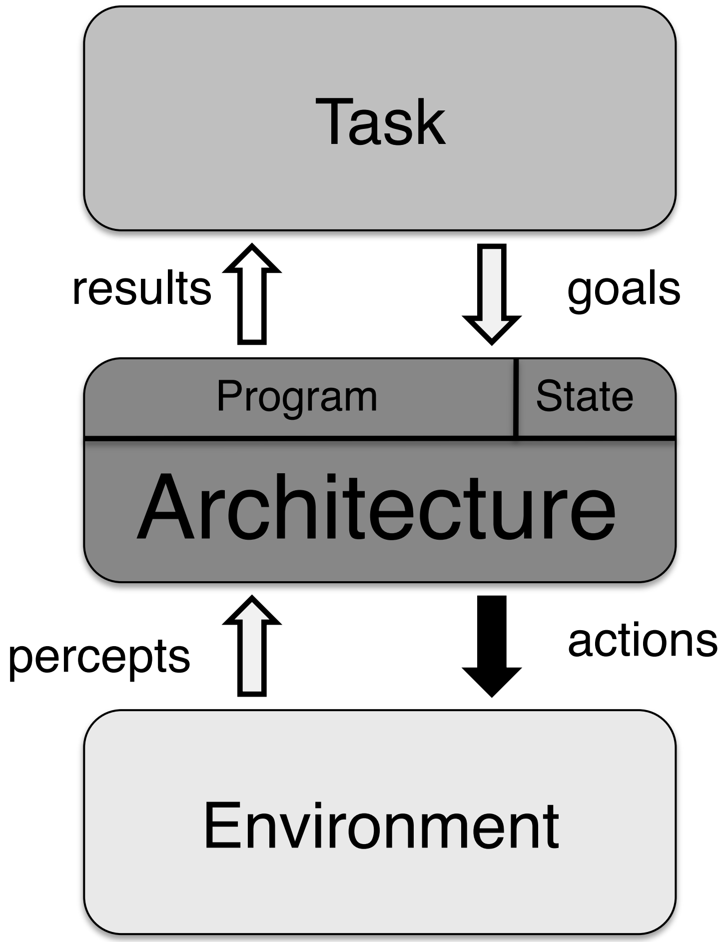
- the *architecture* defines a (real or virtual) machine which runs the agent program
- defines the *atomic operations* of the agent program and implicitly determines the *components* of the agent
- determines which operations happen *automatically*, without the agent program having to do anything
- e.g., the interaction between memory, learning and reasoning
- an architecture constrains kinds of agent programs we can write (easily)

Architectural view of an agent

- **program:** a function mapping from goals & percepts to actions (& results) expressed in terms of virtual machine operations

- **state:** the virtual machine representations on which the agent program operates

- **architecture:** a virtual machine that runs the agent program and updates the agent state



Hierarchies of virtual machines

- in many agents we have a whole hierarchy of virtual machines
  - the agent architecture is usually implemented in terms of a programming language, which in turn is implemented using the instruction set of a particular CPU (or a JVM)
  - likewise some ‘agent programs’ together with their architecture can implement a new, higher-level architecture (virtual machine)
- used without qualification, ‘agent architecture’ means the *most abstract* architecture or the *highest level* virtual machine

Properties of architectures

- an agent architecture can be seen as defining a *class* of agent programs
- just as programs have properties that make them more or less successful in a given task environment
- architectures (classes of programs) have higher-level properties that determine their suitability for a task environment
- choosing an *appropriate architecture* can make it much easier to develop an agent program for a particular task environment

Task environments & architectures

- to choose an architecture which is appropriate for a given task environment we must be able to characterise both the architecture and the task environment
- properties of task environments (last lecture)
- properties of agent architectures (this and subsequent lectures)

Kinds of agent architectures

- **uniform** architectures
  - reactive architectures
  - deliberative architectures
- **hybrid** architectures
  - reactive and deliberative components
- **multi-agent system** architectures
  - many uniform or hybrid architectures, each with additional coordination component(s)

Simple reactive architectures

- *actions* are directly triggered by *percepts*
  - no representations of the environment
  - predefined, fixed response to a situation
  - fast response to changes in the environment

Action selection function

- the action selection function for a simple reactive agent looks like

$$selectAction : Event \rightarrow Action$$

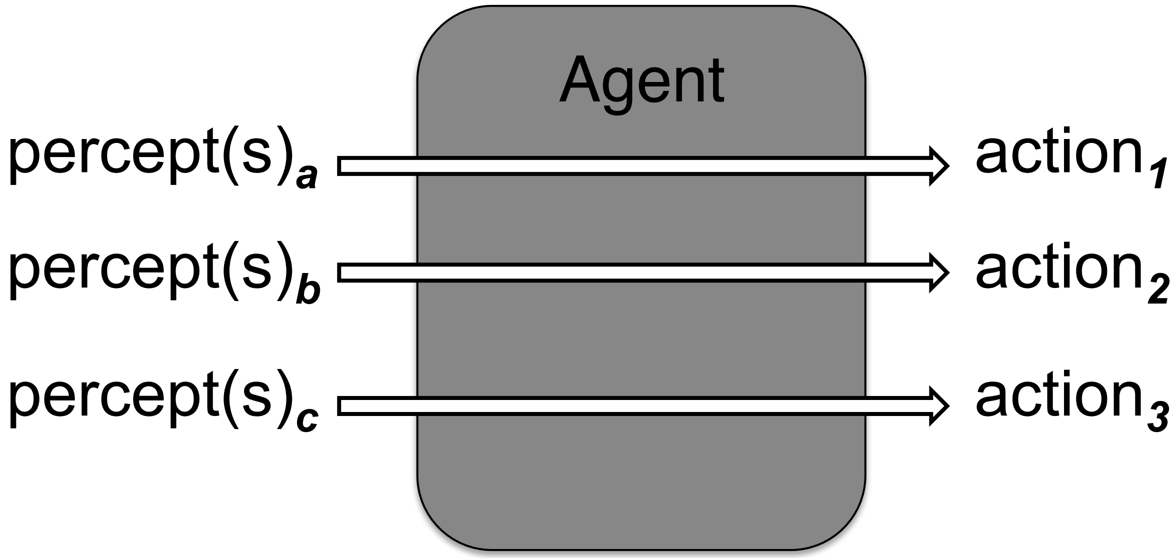
- i.e., it responds only to single events in a predetermined way

- add *state* to respond to sequences of events (next lecture)

Action selection

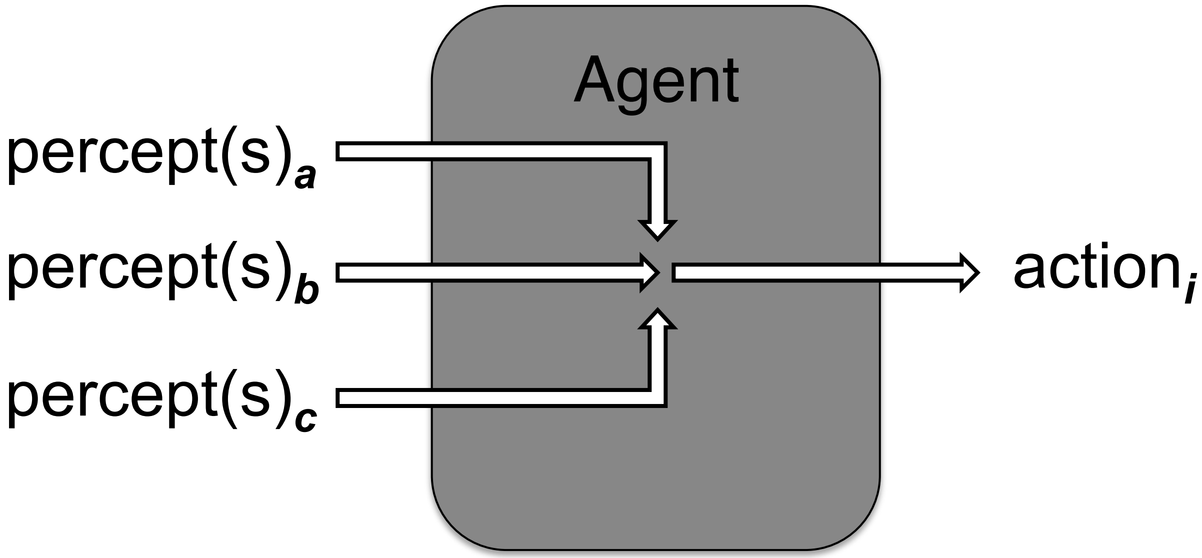
- same percept may trigger multiple actions
- actions can be combined in various ways
  - multiple actions may be executed in *parallel*
  - *combined* into a single action
  - one action may take *precedence* over the others

Parallel actions



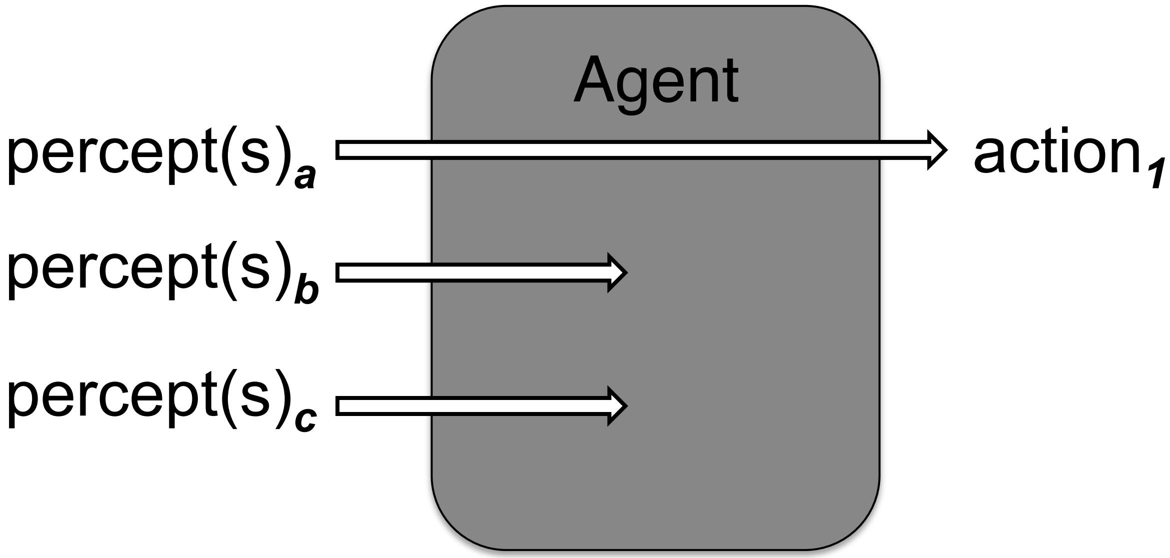
- actions which don't interfere with each other are executed in parallel (within the limitations of the architecture)

Combined actions



- distinct actions triggered by different percepts are *combined* into a single composite action

Prioritised actions



- actions interfere with each other, and the most important action takes precedence

Example: Braitenberg vehicles

- a series of thought experiments designed to show how seemingly complex behaviour can result from very simple reactive architectures
- Braitenberg created a wide range of vehicles, including those (he) imagined to exhibit:
  - cowardice
  - aggression
  - love …

Example: Braitenberg vehicles

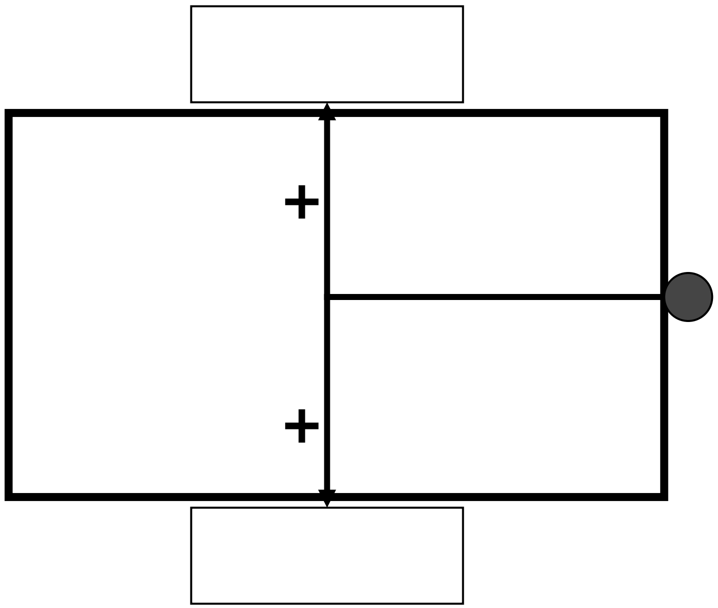
Braitenberg's vehicles use direct, excitatory and inhibitory couplings of sensors to motors:

- **sensors** respond to features in the environment, e.g., heat, light, obstacles etc.

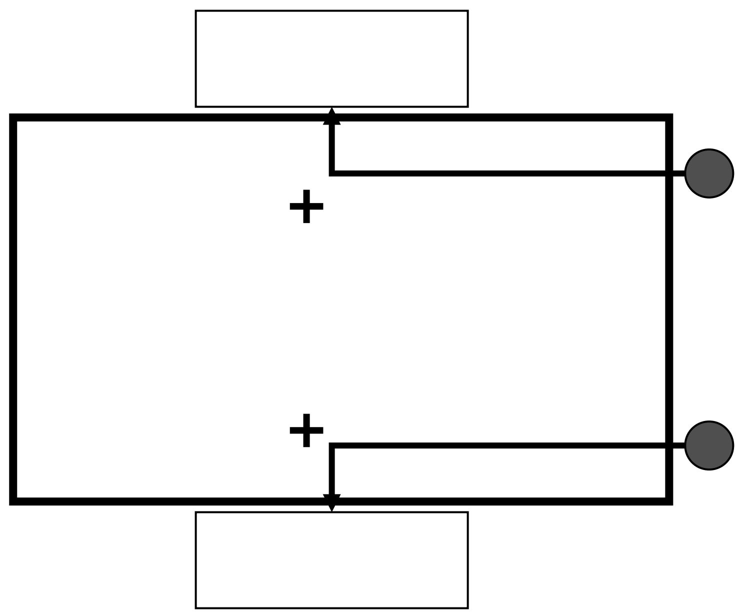
- **motors** move the vehicle in response to signals from the sensors

- **connections** carry signals from the sensors to the motors and either cause them to turn or inhibit them from turning

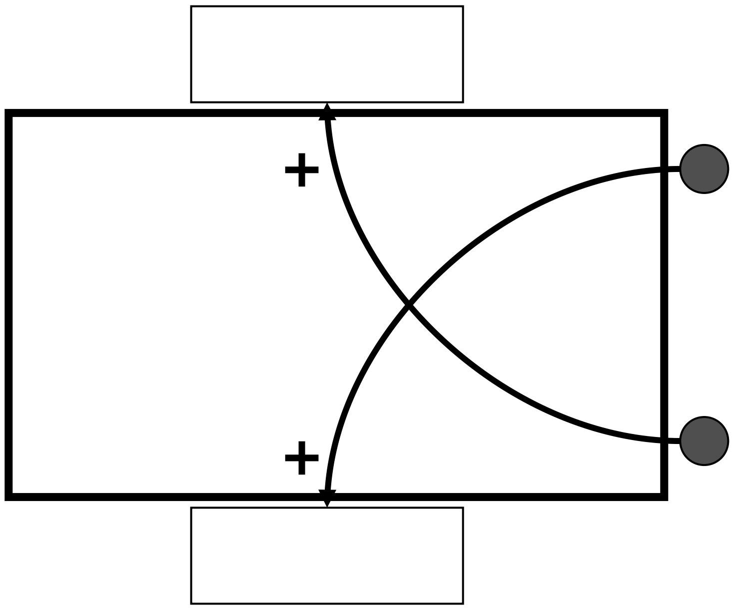
Braitenberg vehicle 1



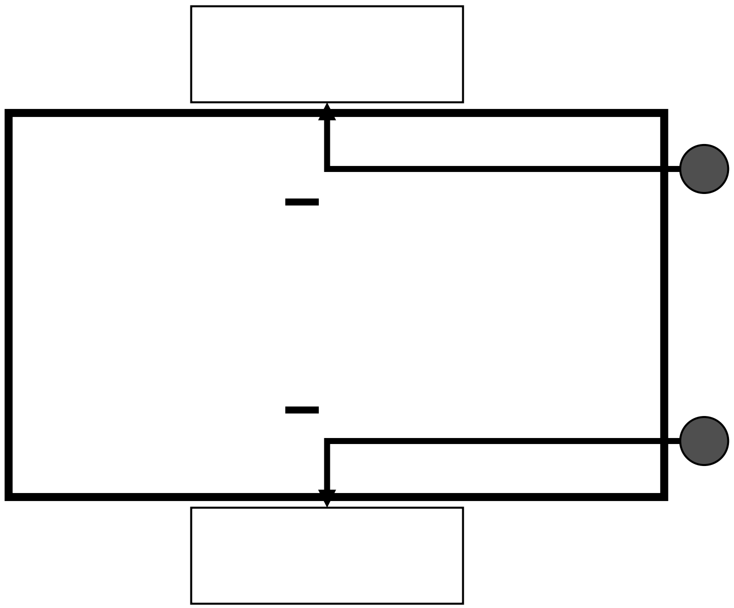
Braitenberg vehicle 2a



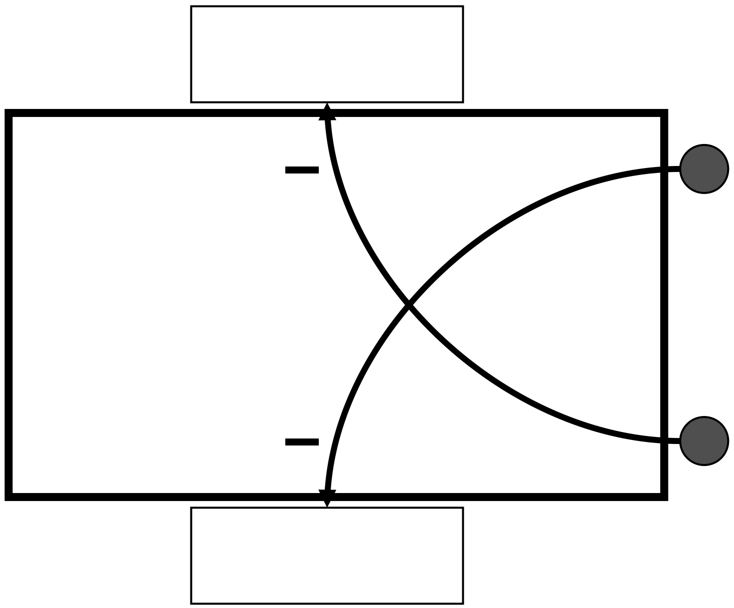
Braitenberg vehicle 2b



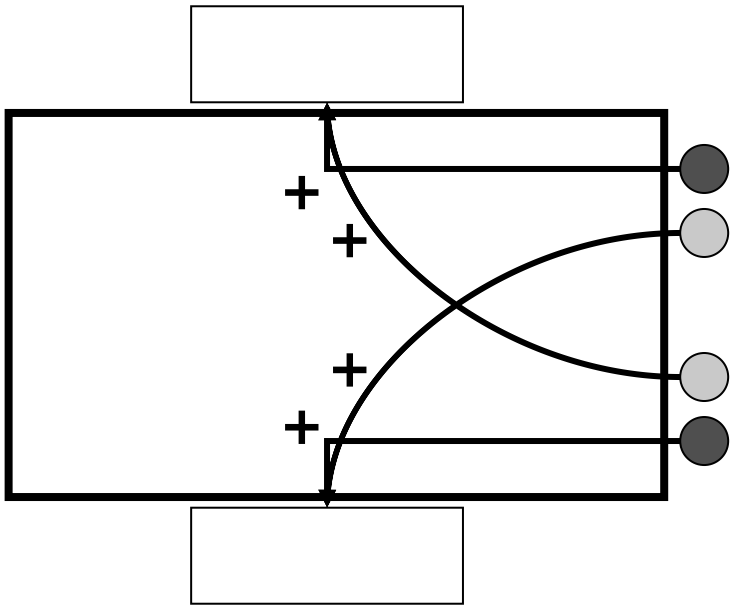
Braitenberg vehicle 3a



Braitenberg vehicle 3b



Braitenberg vehicle 3c



Braitenberg vehicles summary

- Braitenberg's vehicles illustrate how simple reactive architectures can produce complex emergent behaviour

  - however complexity *may* be a reflection of a complex environment

  - we can *ascribe* goals to Braitenberg vehicles, e.g., goal of avoiding collisions, but there is no internal representation of goals

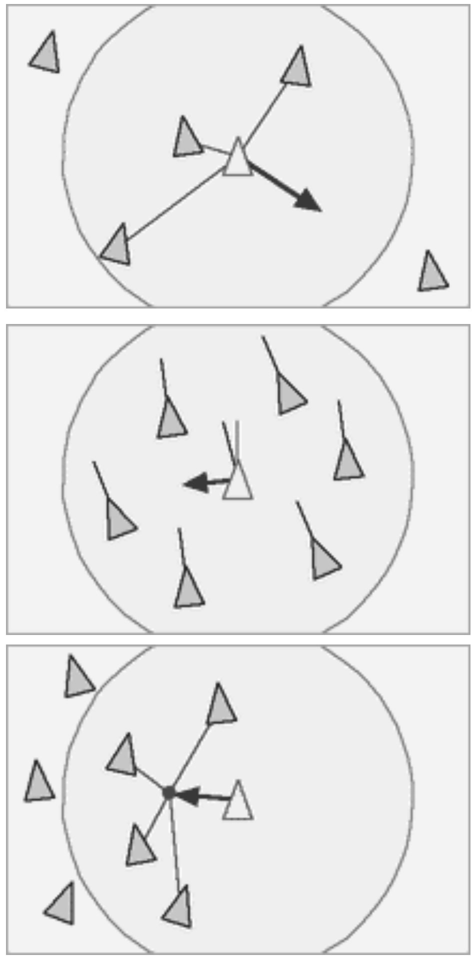
  - "*adopting the intentional stance*"

Example: Boids

- a *boid* is a simple agent that navigates according to its local perception of its environment, the simulated physics of the environment and a set of simple *behavioural rules*:
  - **collision avoidance:** avoid collisions with nearby boids (& static obstacles)
  - **velocity matching:** attempt to match velocity with nearby boids
  - **flock centring:** attempt to stay close to nearby boids
- each boid also has a 'migratory urge', a global direction or position towards which the boids will fly

Behavioural rules

- *collision avoidance* uses only the current position of other boids– achieves minimum separation between boids
- *velocity matching* uses only the current velocity of other boids– maintains minimum separation between boids
- *flock centring* has little effect on boids in the middle of the flock– greatest effect on boids at the edge of the flock



The boid's environment

- physics of the environment implements a simple model of a creature with a finite amount of available energy
- maximum acceleration of a boid is bounded
- simple model of viscous speed damping is used to limit a boid's maximum speed

Boid motion

- each behaviour (collision avoidance, velocity matching and flock centring) produces an acceleration in the form of a 3D vector
- in determining the acceleration for each behaviour, the contribution of each boid to the behaviour of a given boid is inversely proportional to the square of the distance
- maximum acceleration produced by any single behaviour is limited to the boid's maximum acceleration
- basic behaviours are *combined* to give the final motion for each boid

Vector combination

- behaviours are *prioritised*, with collision avoidance being more important than velocity matching which in turn is more important than flock centring
- vectors are combined by adding them up until the boid's maximum acceleration threshold is reached
- if the threshold would be exceeded, remaining vector(s) are scaled to stay within the acceleration threshold
- gives priority to the most important behaviours, e.g., will suppress flock centring and velocity matching if a collision is imminent
- mixture of *combined* and *prioritised* action selection

Boids summary

Boids illustrate how simple reactive architectures can produce complex emergent behaviour:

- “The aggregate motion we intuitively recognise as ‘flocking’ depends on a limited, localised view of the world.”
- “The isolated behaviour of a flock tends to reach a steady state and becomes rather sterile. … Environmental obstacles and the boid’s attempt to navigate around them increase the apparent complexity of the behaviour of the flock.”

– (Reynolds 1987)

Advantages of simple reactive architectures

- simple architectures can produce complex behaviour
- no representations of the environment or complex problem solving
- can use dedicated, parallel hardware
- fast (often real-time) response to changes in the environment

Disadvantages of simple reactive architectures

- fixed response to a given situation
- all responses must be defined in advance
- can’t cope with novel situations for which they don’t have a predefined behaviour
- can’t solve some problems at all

The next lecture

Reactive Architectures II

Suggested reading:

- Braitenberg (1984), *Vehicles: Experiments in Synthetic Psychology*, MIT Press.

Behavioural rules

- **collision avoidance** uses only the current position of other boids
  - achieves minimum separation between boids
- **velocity matching** uses only the current velocity of other boids
  - maintains minimum separation between boids
- **flock centring** has little effect on boids in the middle of the flock
  - greatest effect on boids at the edge of the flock