Non-Symbolic AI lecture 5

We shall look at 2 alternative non-symbolic AI approaches to robotics

- Subsumption Architecture
- Evolutionary Robotics

Classical AI

When building robots, the Classical AI approach has the ro
as a scientist-spectator, seeking information from outside.

"SMPA" -- so-called by Brooks (1999)

- S sense
- M model
- P plan
- A action

Brooks' alternative

Brooks' alternative

Brooks' alternative is in terms of many individual and larg
separate **behaviours** – where any one behaviour is
generated by a pathway in the 'brain' or control system a
the way from Sensors to Motors.

No Central Model, or Central Planning system.

Subsumption architecture (1)

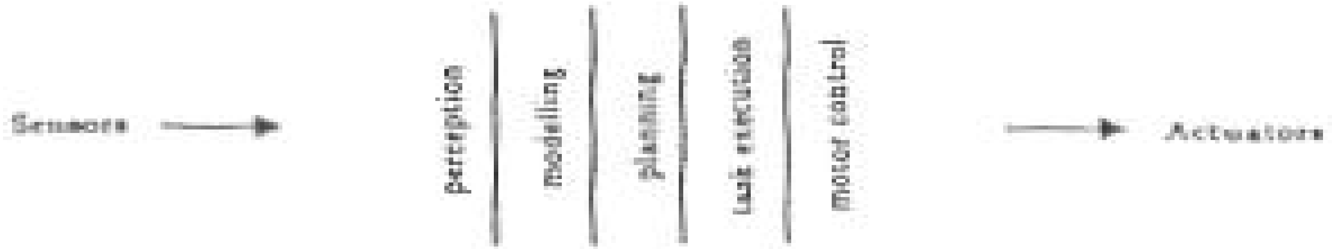


Fig. 1. Traditional decomposition of a mobile robot control system into functional modules.

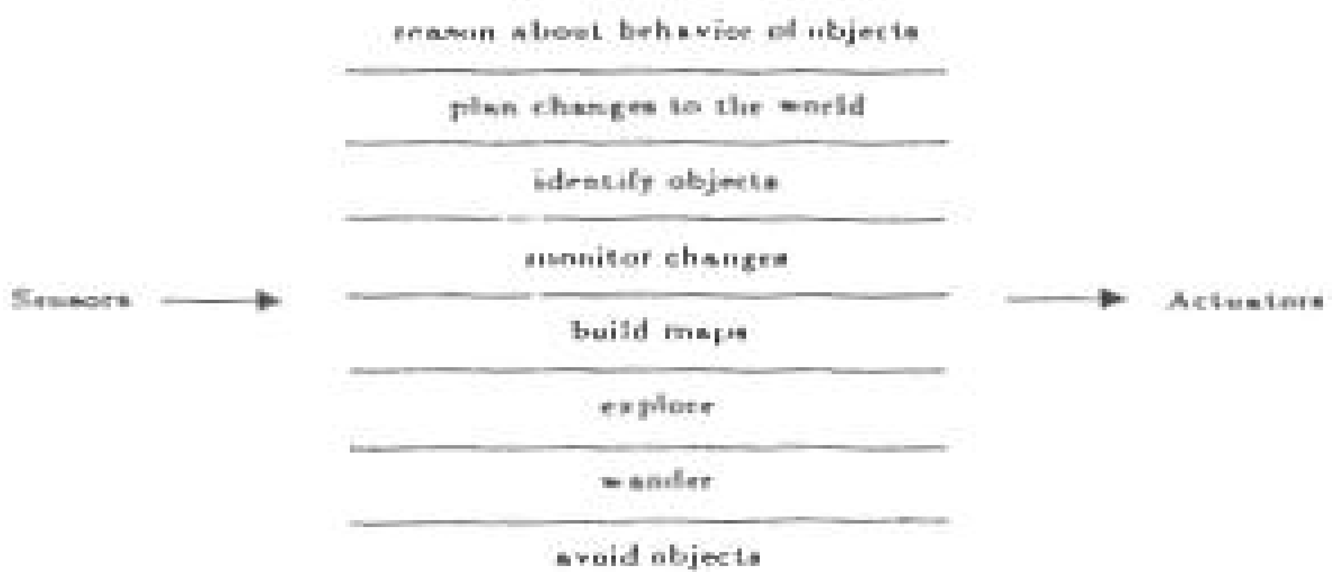


Fig. 2. Decomposition of a mobile robot control system based on task-achieving behaviors.

(1a)

```
                        P     M     P     T     M
                        E     O     L     A     O
                        R     D     A     S     T
                        C     E     N     K     O
SENSORS    >>>>>        E >>> L >>> N >>>    >>> R    >>>>>    ACTUA
                        P     L     I     E
                        T     I     N     X     C
                        I     N     G     E     O
                        O     G           C     N
                        N                 U     T
                                          T     R
                                          E     O
                                                L
```

Traditional decomposition of a mobile robot control syste
into functional modules

(1b)

SENSORS >>>>>>

REASON ABOUT BEHAVIOR OF OBJECTS
PLAN CHANGES TO THE WORLD
IDENTIFY SUBJECTS
MONITOR CHANGES
BUILD MAPS
EXPLORE
WANDER
AVOID OBJECTS

>>>> ACTUAT

Decomposition of a mobile robot control system based o
task-achieving behaviors

Subsumption architecture (2)

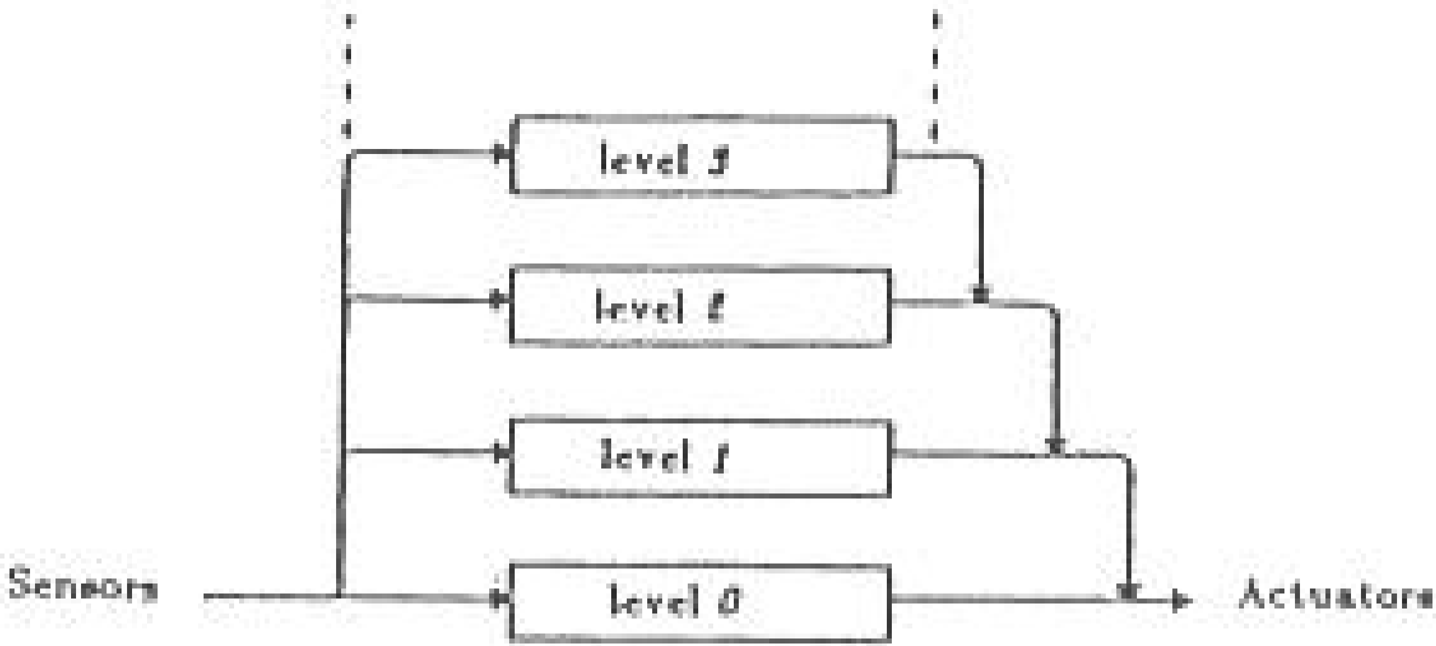
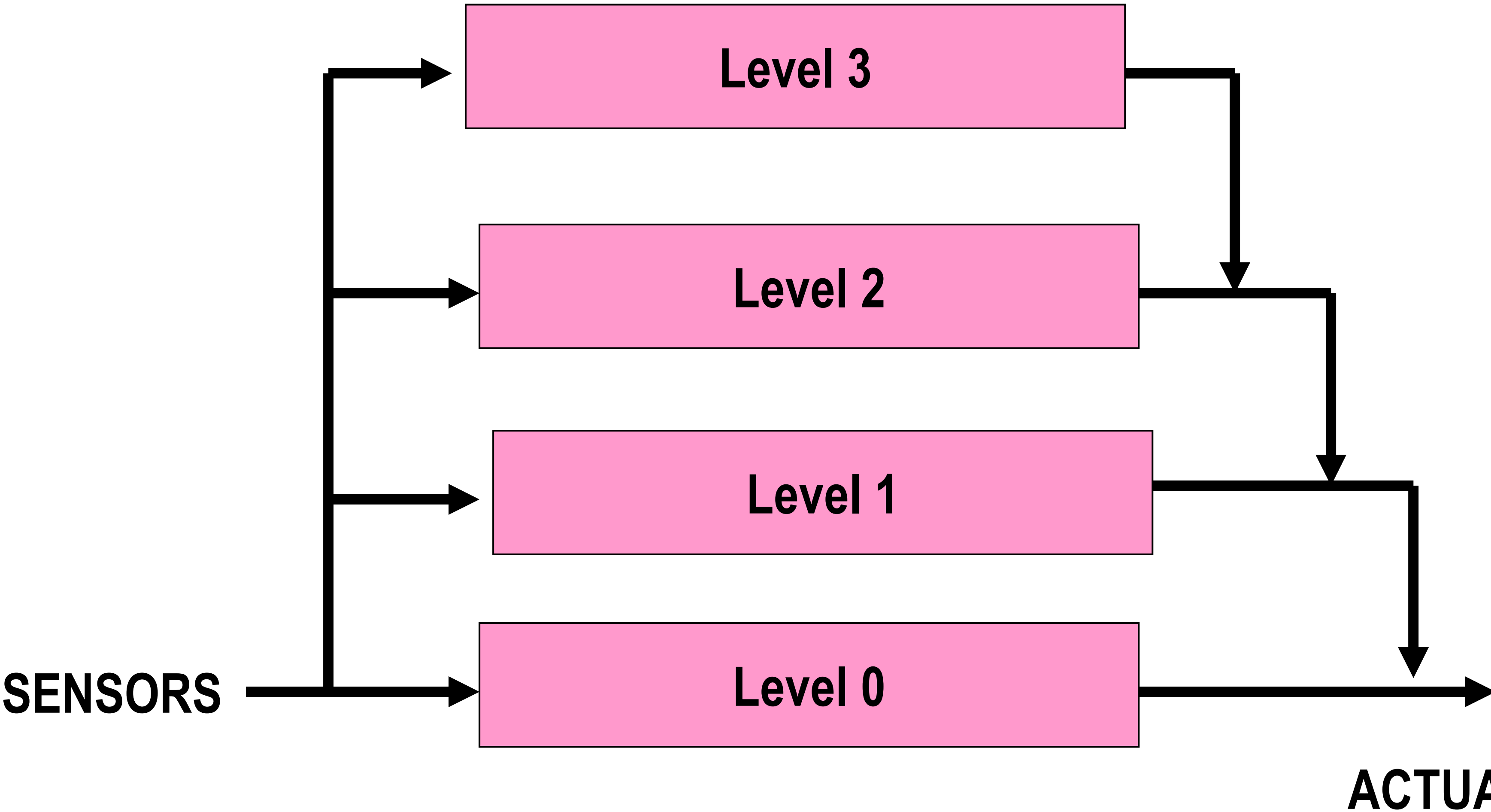


Fig. 3. Control is layered with higher level layers subsuming the roles o lower level layers when they wish to take control. The system can b partitioned at any level, and the layers below form a complete operationa control system.

(2a)



Control is layered with higher levels subsuming control of lower layers when they wish to take control.

Subsuming

'Subsume' means to take over or replace the output from
'lower layer'.

The 2 kinds of interactions between layers are

1. Subsuming
2. Inhibiting

Generally only 'higher' layers interfere with lower, and to
relatively small extent – this assists with an incremental
design approach.

Subsumption architecture (3)

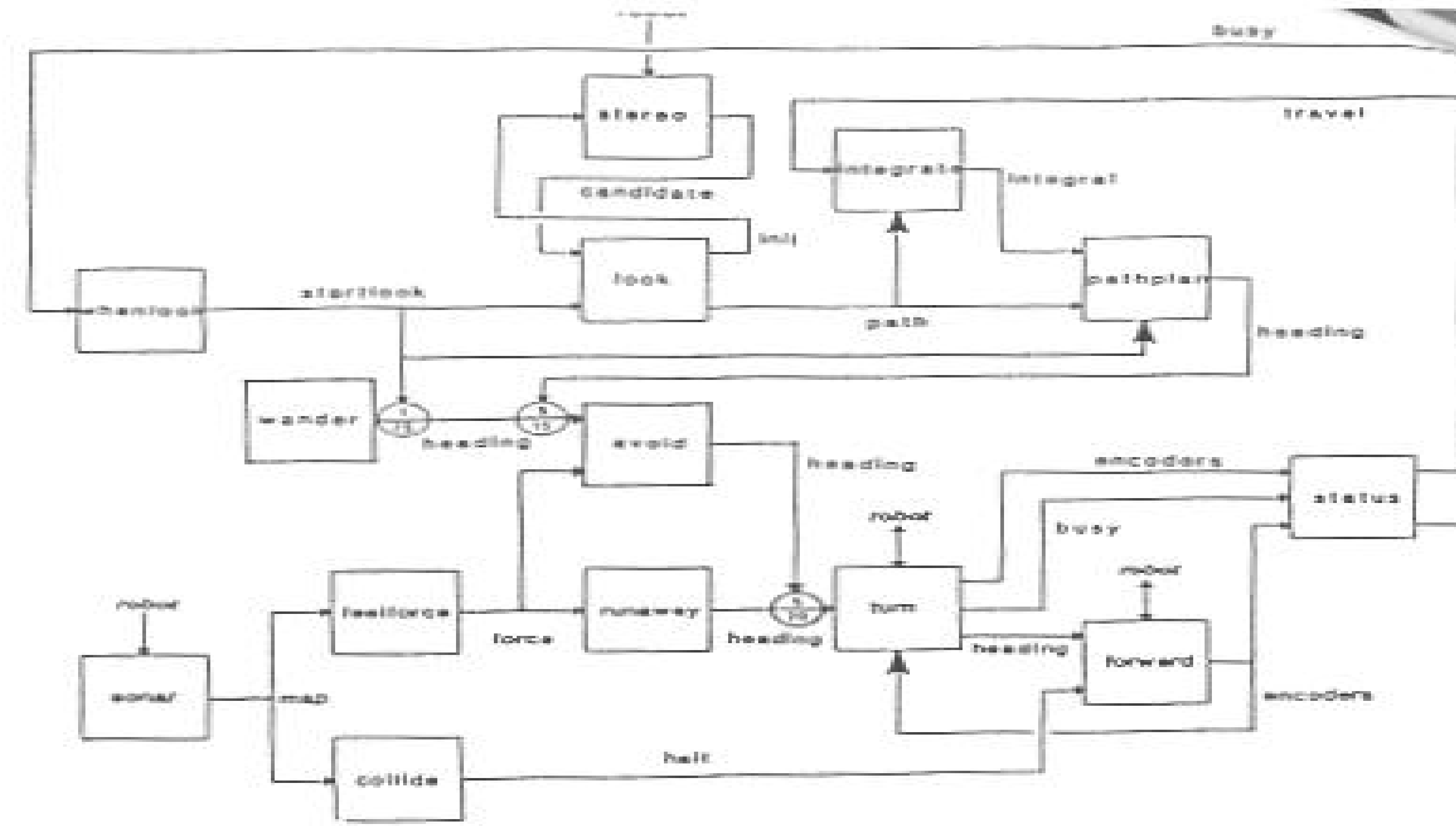


Fig. 7. Level 0 and 1 control systems augmented with the level 2 system.

Subsumption architecture (4)

That looked a bit like a Network – except rather than (artificial) Neurons the components are versions of

AFSMs

Augmented

Finite

State

Machines

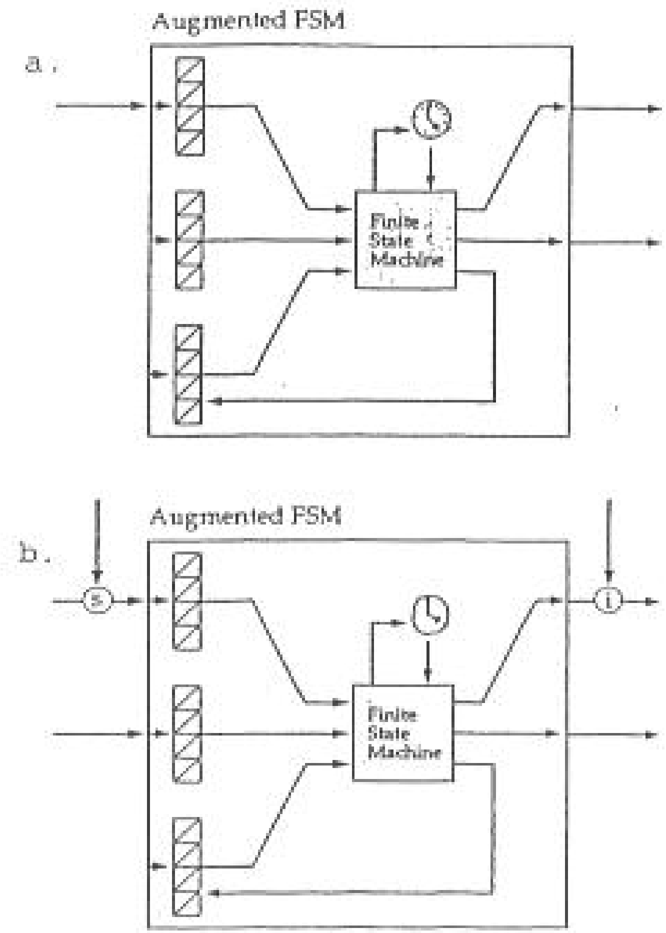


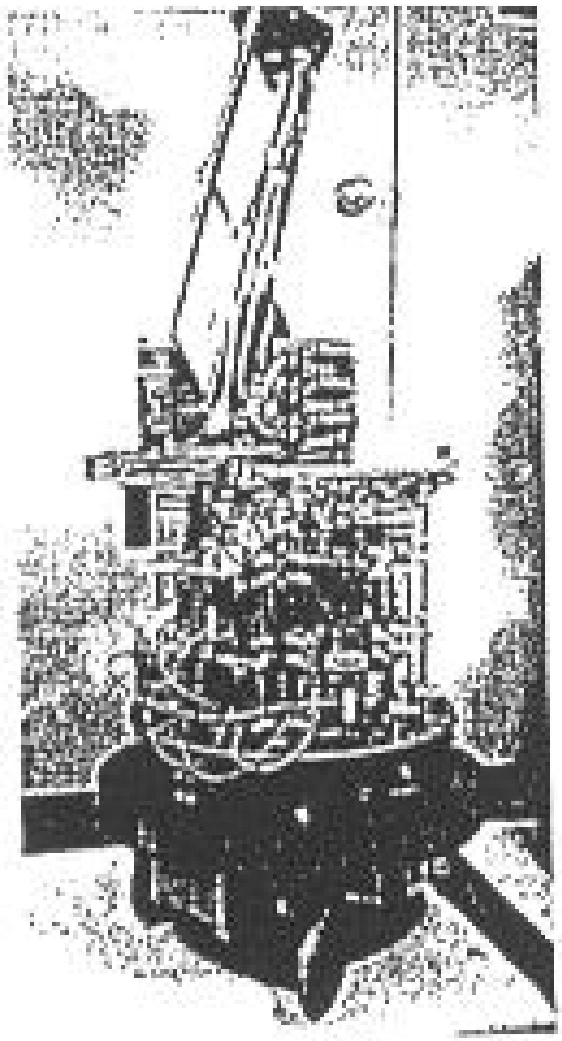
Figure 2.5: An augmented finite state machine (a) consists of registers, alarm cl
natorial network and a regular finite state machine. Input messages are delivered to
messages can be generated on output wires. AFSMs are wired together in networks

AFSMs

An AFSM consists of registers, alarm clocks (**time!**), a combinat network and a regular finite state machine. Input messages are delivered to registers, and messages can be generated on outpu wires.

As new wires are added to a network (lower figure before), they connect to existing registers, inhibit outputs, or suppress inputs.

Herbert



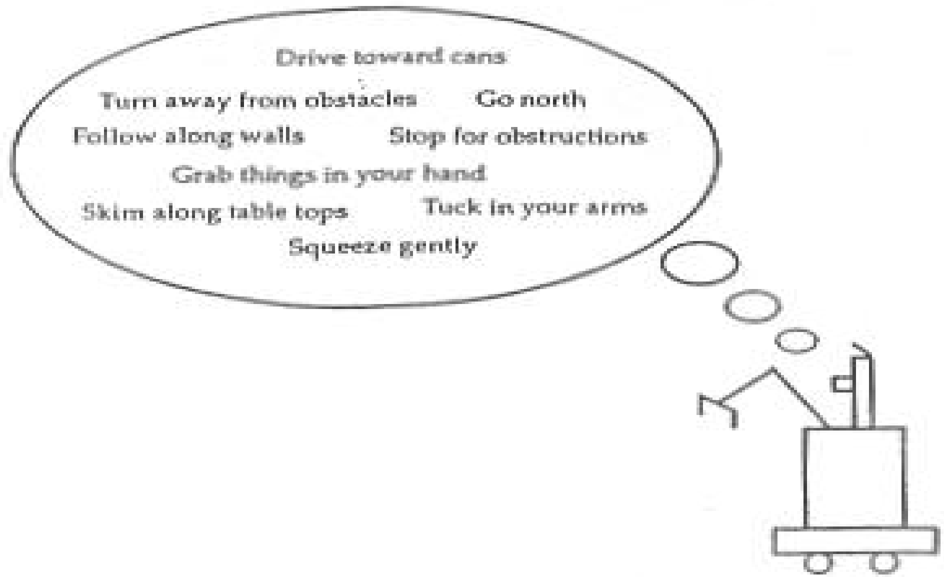


Figure 12.2: Herbert was controlled by a "colony" of independent agents all wanting
Sensing to action with no communication with eachother except via the world.

16 infrared sensors, compass, laser ligh striper for finding soda-cans. 24 8-bit microprocessors distributed around the b

Herbert's actions

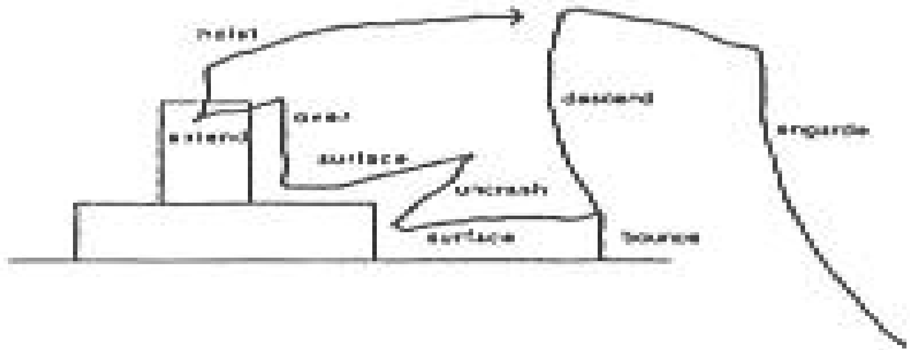
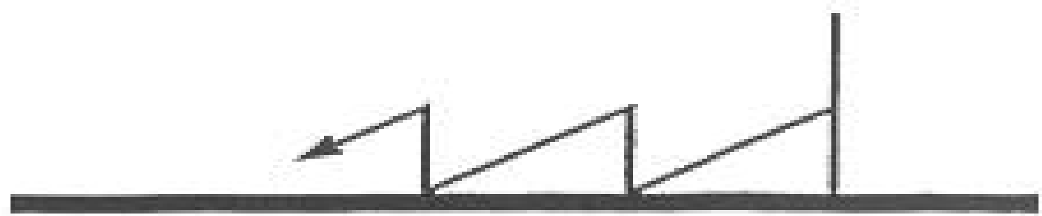


Figure 12.13: Like figure 12.12 this shows the path of the finger tips while searching for a soda can. This time there is an obstacle on the table surface, and a very different "plan" emerges from the interaction of the robot and its environment.



SKIM level of control

Figure 12.10: The strategy for Herbert's arm to find something that is in front of it is fo skim along a surface in a sawtooth pattern. It reaches forward and down, bouncing up wh the touch sensors on the finger tips detect a surface.

Subsumption summary

- New philosophy of hand design of robot control systems
- Incremental engineering – debug simpler versions first
- Robots must work in **real time** in the **real world**
- Spaghetti-like systems unclear for analysis
- Not clear if behaviours can be re-used
- Scaling – can it go more than 12 behaviours?

Evolutionary Robotics

Evolutionary Robotics (ER) can be done

- ✓ for Engineering purposes - to build useful robots
- ✓ for Scientific purposes - to test scientific theories

It can be done

- ✓ for Real or
- ✓ in Simulation

Here we shall start with the most difficult, robots with Dynamic Recurrent Neural Nets, tested for Real.

Then we shall look at simplifications and simulations.

The Evolutionary Approach

Humans are highly complex, descended over 4 bn yrs from the 'origin of life'.

Let's start with the simple first - 'today the earwig'
(not that earwigs are that simple ...)

Brooks' subsumption architecture approach to robotics is 'design hand', but still inspired by an incremental, evolutionary approach

✓ Get something simple working (debugged) first
✓ Then try and add extra 'behaviours'

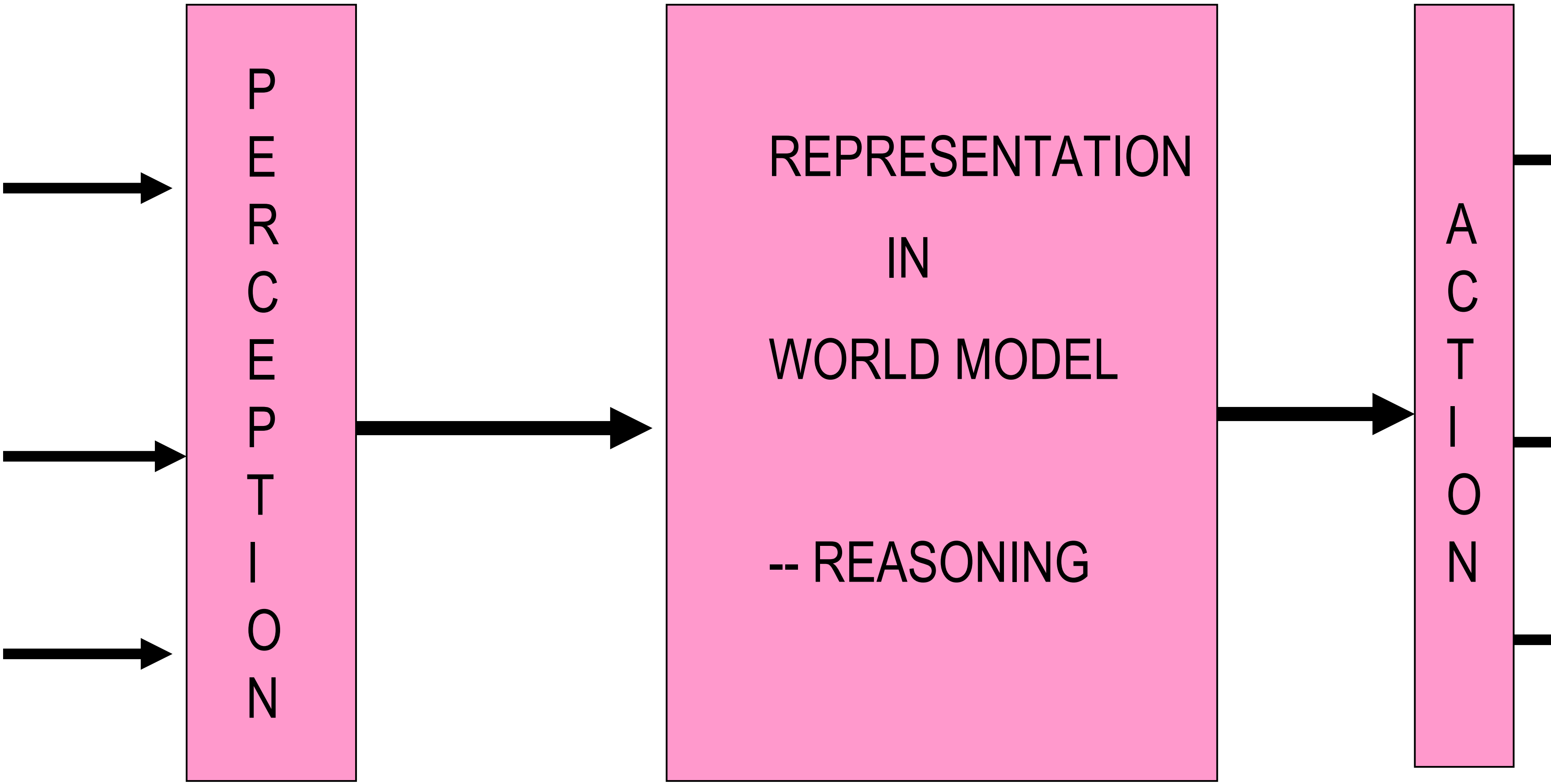
What Class of Nervous System

When evolving robot 'nervous systems' with some form of GA, th
the genotype ('artificial DNA') will have to encode:

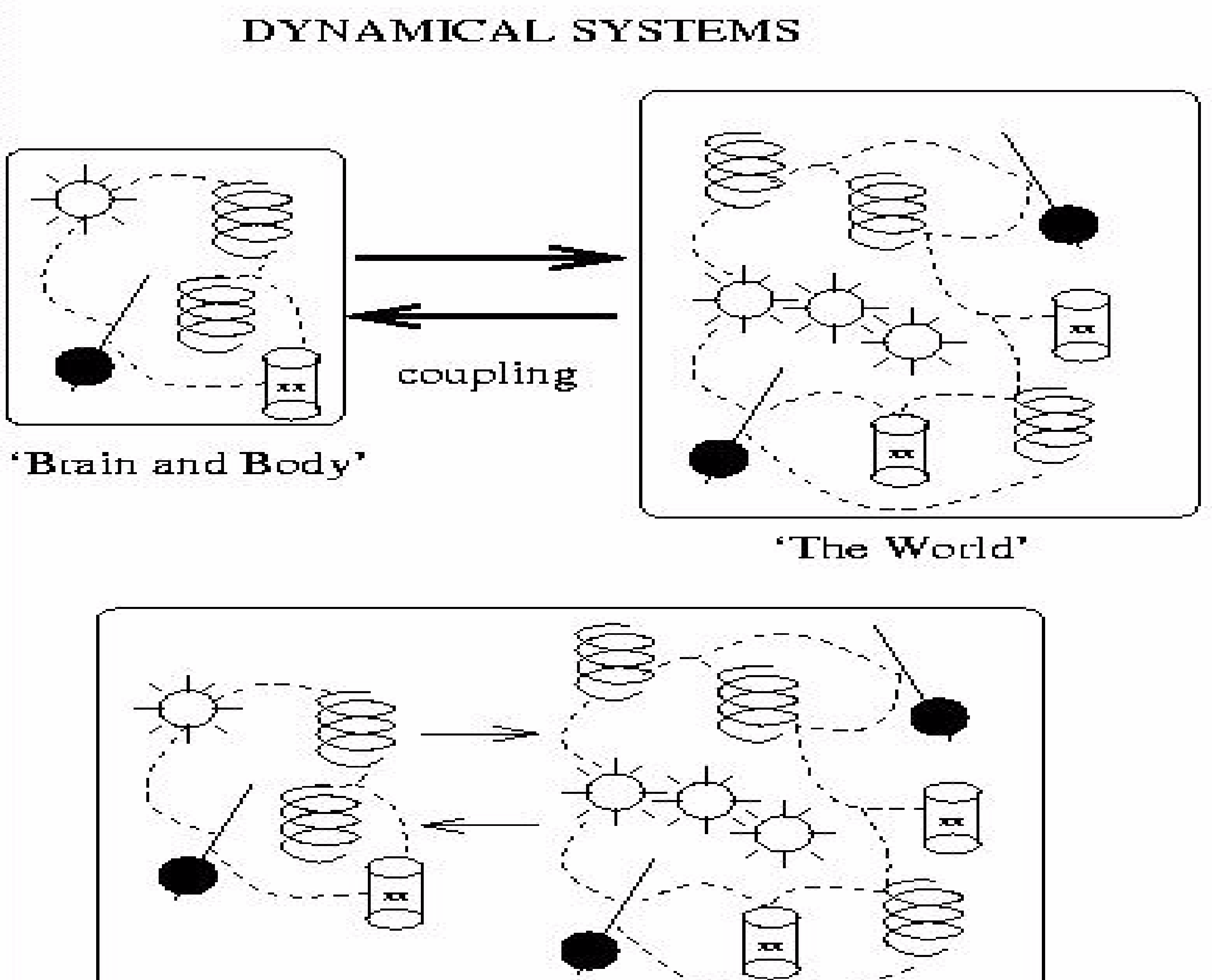
- ✓The architecture of the robot control system
- ✓Also maybe some aspects of its body/motors/sensors

But what kind of robot control system, what class of possible systems should evolution be 'searching through' ?

… could be a classical approach ?



… or a Dynamical Systems Approach



DS approach to Cognition

cf R Beer 'A Dynamical Systems Perspective on Autonomous Agents' Tech Report CES-92-11. Case Western Reserve Univ. Also papers by Tim van Gelder.

In contrast to Classical AI, computational approach, the DS approach is one of 'getting the dynamics of the robot nervous sy right', so that (coupled to the robot body and environment) the behaviour is adaptive.

Brook's subsumption architecture, with AFSMs (Augmented Finite State Machines) is one way of doing this.

Dynamic Recurrent Neural Networks

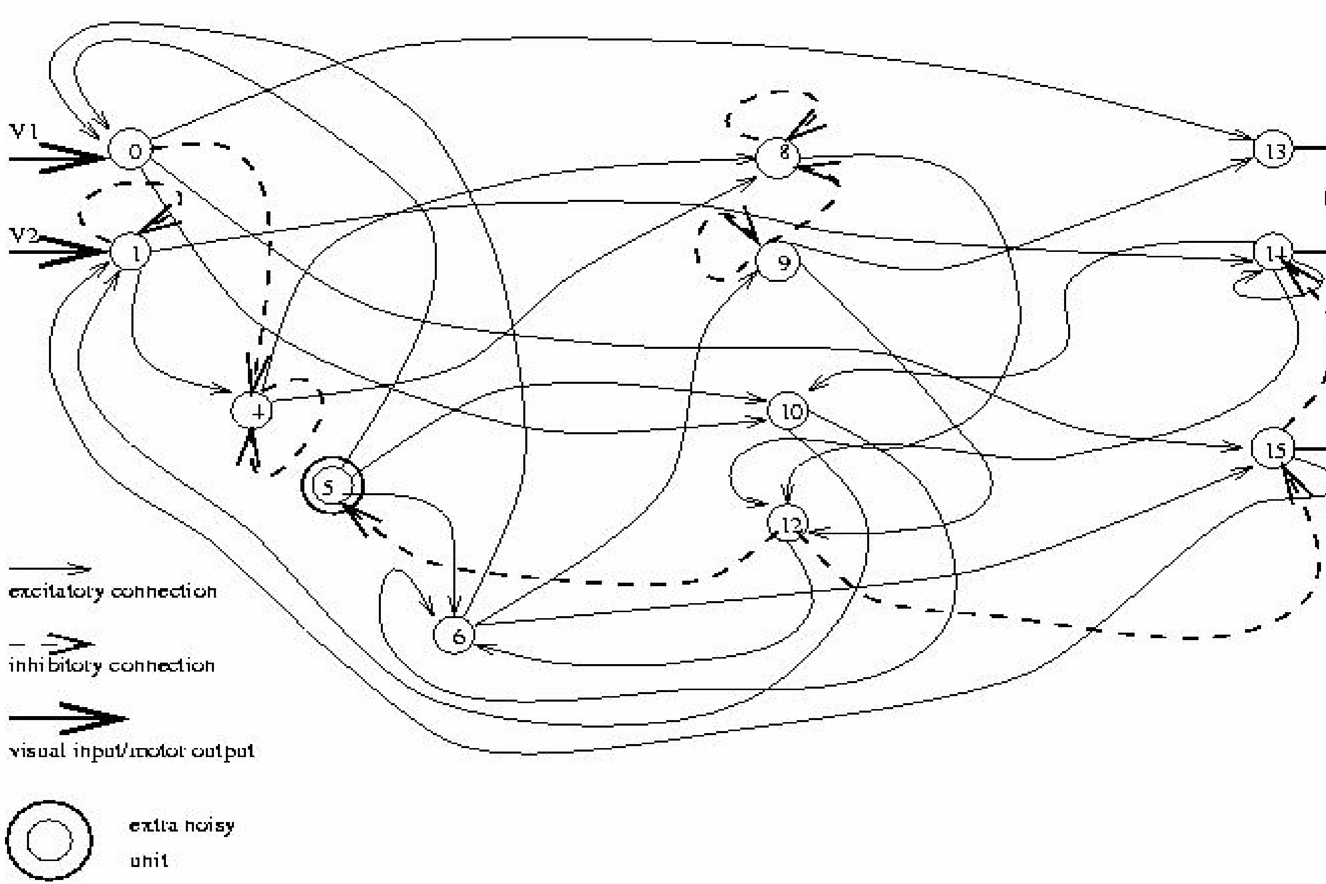
Dynamic Recurrent Neural Networks

DRNNs (or CTRNs = Continuous Time Recurrent Networks) are another (really quite similar way).

You will learn about other flavours of Artificial Neural Networks (ANNs) in Adaptive Systems course.
-- eg ANNs that 'learn' and can be 'trained'.

These DRNNs are basically different -- indeed basically just a convenient way of specifying a class of dynamical systems
-- so that different genotypes will specify different DSs, giving rob different behaviours.

One possible DRNN, wired up

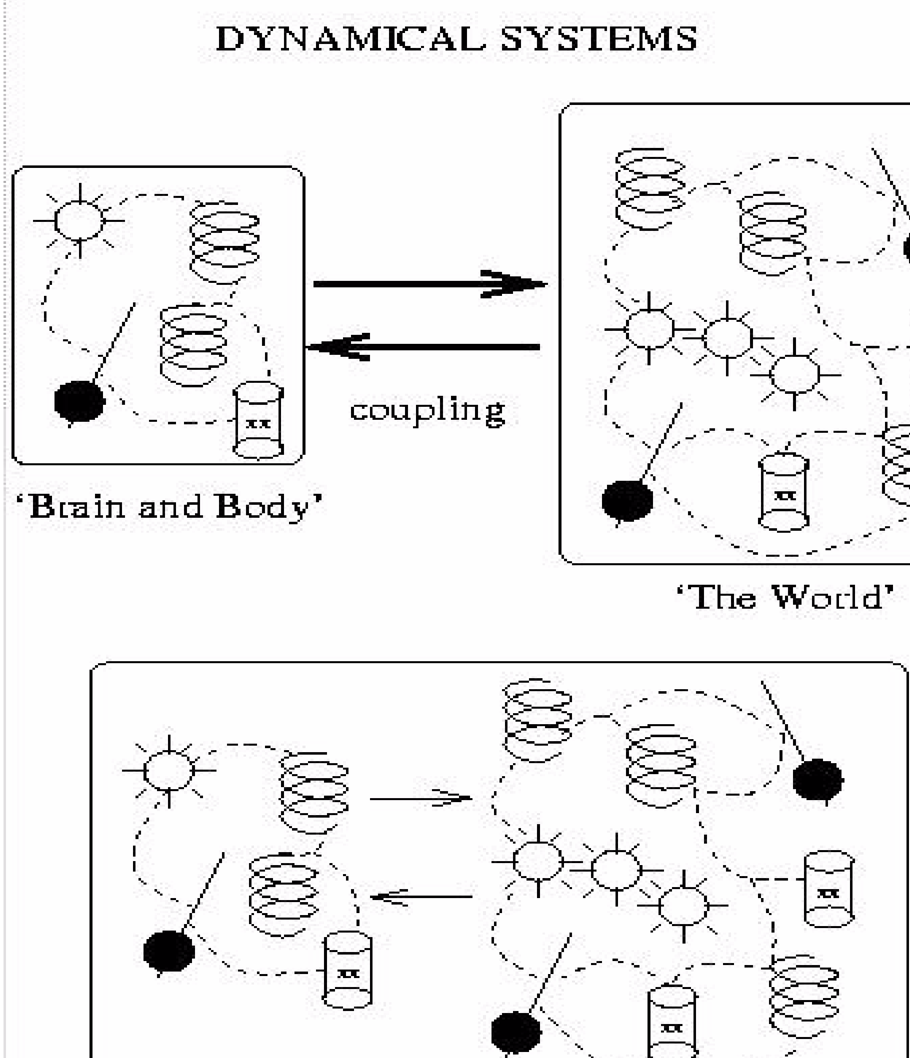


This is just ONE possible DRNN, which ONE specific genotype specified

Think of it as ...

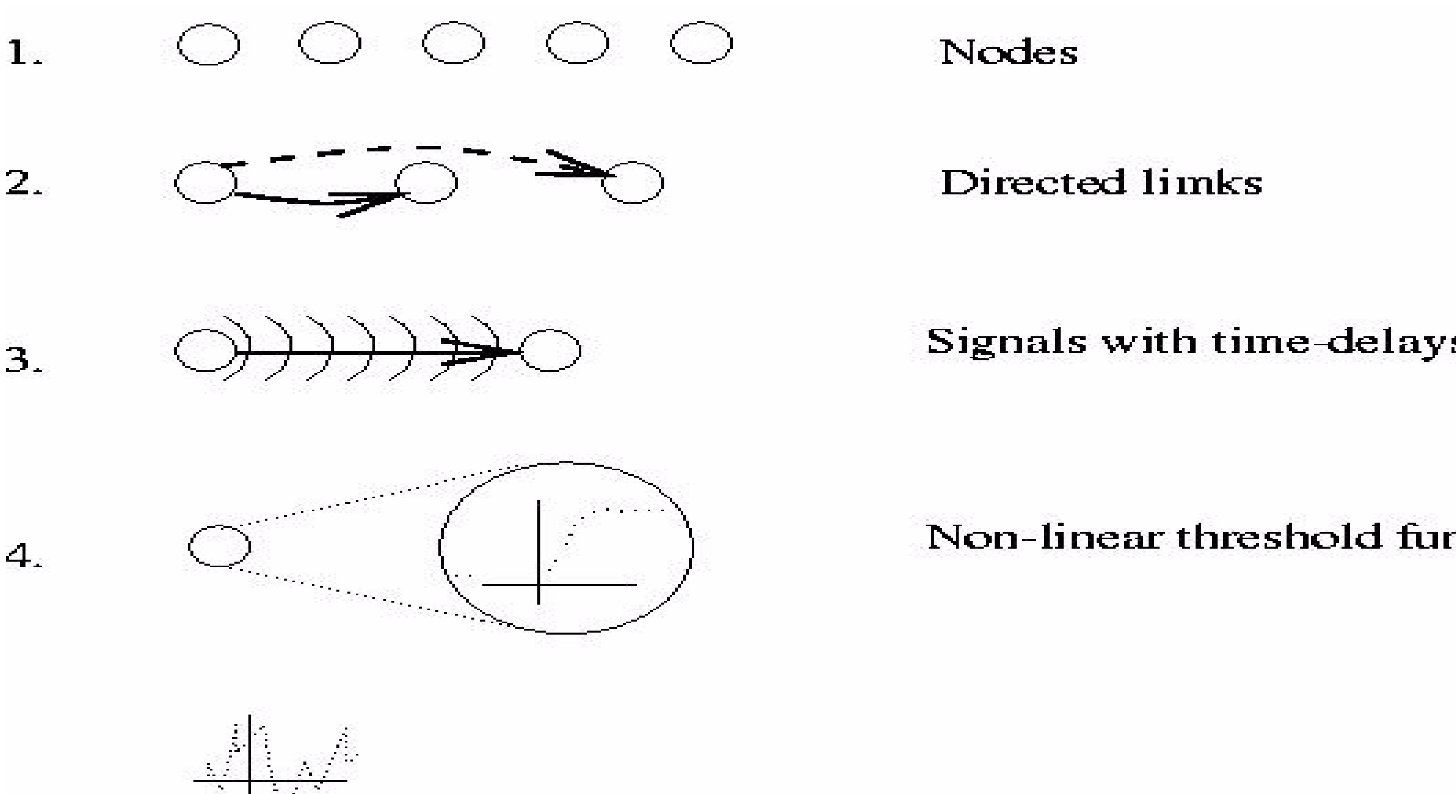
Think of this as a nervous system with its own Dynamics.

Even if it was not connected up to the environment (I.e. it was a 'brain-in-a-vat’), it would have its own dynamics, through internal noise and recurrent connections)



DRNN Basics

The basic components of a DRNN are these
(1 to 4 definite, 5 optional)



ER basics

ER basics

The genotype of a robot specifies
(through the encoding genotype->phenotype that WE decide on as appropriate)
how to 'wire these components up' into a network connected to sensors and motors.

(Just as there are many flavours of feedforward ANNs, there are many possible versions of DRNNs – in a moment you will see ju[s]t one.)

Then you hook all this up to a robot and evaluate it on a task.

Evaluating a robot

When you evaluate each robot genotype, you

- ✓ Decode it into the network architecture and parameters
- ✓ Possibly decode part into
  body/sensor/motor parameters
- ✓ Create the specified robot
- ✓ Put it into the test environment
- ✓ Run it for n seconds, scoring it on the task.

Any evolutionary approach needs a selection process, whereby
different members of the population have different chances of
producing offspring according to their **fitness**

Robot evaluation

(Beware - set conditions carefully!)

Eg: for a robot to move, avoiding obstacles -- have a number of obstacles in the environment, and evaluate it on how far it moves forwards.

Have a number of trials from random starting positions

- ✓ take the average score, **or**
- ✓ take the worst of 4 trials, **or**
- ✓ (alternatives with different implications)

DSs -> Behaviour

The genotype specifies a DS for the nervous system

Given the robot body, the environment, this constrains the behav

The robot is evaluated on the behaviour.

The phenotype is (perhaps):

- ✓ the architecture of the nervous system(/body)
- ✓ or ... the behaviour
- ✓ or even ... the fitness

Robustness and Noise

For robust behaviours, despite uncertain circumstances, noisy t
are neeeded.

Internal noise (deliberately put into the network) affects the dyna
(eg self-initiating feedback loops) and (it can be argued) makes
'evolution easier'
-- 'smooths the fitness landscape'.

Summarising DSs for Robot Brains

They have to have **temporal** dynamics.
Three (and there are more...) possibilities are:

(1) Brook's subsumption architecture

(2) DRNNs as covered in previous slides

(3) Another option to mention here: Beer's networks

see Beer ref. cited earlier, or "Computational and Dynamical Languages for Autonomous Agents", in Mind as Motion, T van Gelder & R. Port (eds) MIT Press

Beer's Equations

Beer uses **CTRNNs** (continuous-time recurrent NNs), where for each node (i = 1 to n) in the network the following equation hold

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{n} w_{ji}\sigma(y_j - \theta_j) + I_i$$

y_i = activation of node i

τ_i = time constant, w_{ji} = weight on connection from node j to node

$\rho(x)$ = sigmoidal = $(1/1+e^{-x})$

η_i= bias,

I_i = possible sensory input.

Applying this for real

Applying this for real

One issue to be faced is:

- Evaluate on a real robot, or
- Use a Simulation ?

On a real robot it is expensive, time-consuming -- and for evolution you need many many evaluations.

Problems of simulations

On a simulation it should be much faster
(though note -- may not be true for vision)
cheaper, can be left unattended.

BUT AI (and indeed Alife) has a history of toy, unvalidated simulations, that 'assume away' all the genuine problems that m
be faced.

Eg: grid worlds "move one step North"

Magic sensors "perceive food"

Principled Simulations ?

How do you know whether you have included all that is necessa
a simulation?

-- only ultimate test, **validation,** is whether what works in simula
ALSO works on a real robot.

How can one best insure this, for Evolutionary Robotics ?

'Envelope of Noise' ?

Hypothesis: -- "if the simulation attempts to model the real world fairly accurately, but where in doubt extra noise (through variatio driven by random numbers) is put in, then evolution-in-a-noisy-simulation will be more arduous than evolution-in-the-real-world"

Ie put an envelope-of-noise, with sufficient margins, around cruc parameters whose real values you are unsure of.

"Evolve for **more robustness** than strictly necessary"

Problem: some systems evolved to rely on the existence of nois that wasnt actually present in real world!

Jakobi's Minimal Simulations

Jakobi's Minimal Simulations

See, by Nick Jakobi:

(1) Evolutionary Robotics and the Radical Envelope of Noise Hypothesis and
(2) The Minimal Simulation Approach To Evolutionary Robotics

available on http://www.cogs.susx.ac.uk/users/nickja/

Minimal simulation approach developed explicitly for ER -- the problem is often more in simulating the environment than the rob

Minimal Simulation principles

Work out the minimal set of environmental features needed for th
job -- the **base set**.

Model these, with some principled envelope-of-noise, so that wh
uses these features in simulation will work in real world
-- '**base-set-robust**'

Model everything ELSE in the simulation with wild, unreliable no
- so that robots cannot evolve in simulation to use anything other
trhan the base set
-- '**base-set-exclusive**'