SIMPLE AGENTS

Introduction

- This lecture will start looking at AI techniques.
- We will start with one that is relatively new, but also very simple.
- This kind of system is simple to program.
- It is also what we will use for the first robotics exercise.

Behaviour-based AI

We can distinguish two approaches to AI:

- Classic AI:
  - Symbolic representations;
  - "Good Old Fashioned AI" (GOFAI).
- Behaviour-based AI:
  - Representation-free;
  - "Nouvelle AI".

Classical models are *deliberative*. They involve what we recognise as thinking.

- Sense-Plan-Act:
  - Sense the world and figure out where we are;
  - Generate a plan to get where we want to go;
  - Translate plan into actions.
- Iterate until goals are achieved.
- Need some kind of world model, notion of goal etc.

Behaviour-based approaches just act:

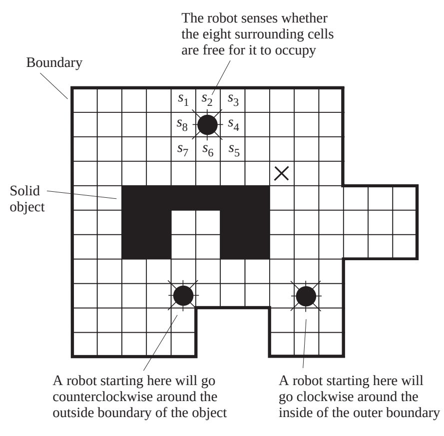
> When you think on the field, you've automatically lost that down. The time you should be thinking is during the course of the week in practice. That's when the light should go on. When you get in the game, it's all about reacting to what you see.

(Albert Lewis, Oakland Raiders cornerback).

Hypothesis is:

- Most activity isn't planned out; it is just reaction.
- Complex behaviours are just combinations of simple behaviours.
  - If we can string together enough simple behaviours we will get complex behavior.
- Can get further with this "bottom-up" approach than with the classical approach.
  - An artificial cockroach that works is better than an artificial human that doesn't.
- Elephants don't play chess.

Example



- Task:
  - Go to a cell adjacent to a boundary or object and follow its perimeter.
- Sensors:
  - Can sense if adjacent cells are occupied.
  - Each s_i has value 0 when that cell can be occupied. 1 otherwise.
- Thus at X, the sensors have value:

$$(0, 0, 0, 0, 0, 0, 1, 0)$$

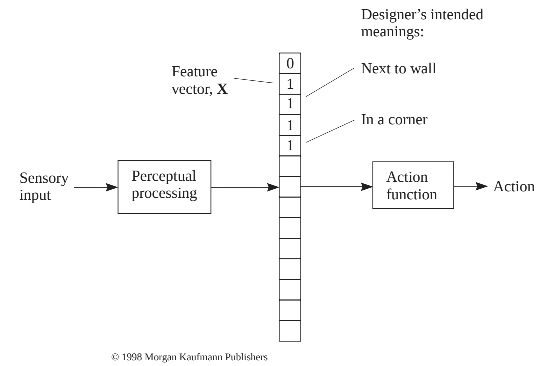
- In general we write $S = (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$

- Actions:
  - north move up in grid.
  - east move right in grid.
  - south move down in grid.
  - west move left in grid.
- We write the set of all actions as A.
- These work provided the cell into which the robot tries to move is free.
- The task is then to come up with a function from a set of s_i to some action:

$$f : S \mapsto A$$

Perception & Action

In general, the situation is:



Features can be numerical, categorical, or binary-valued.

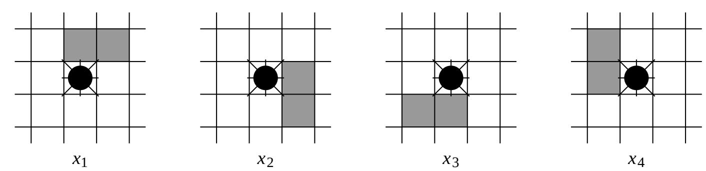
- The split between action and perception is arbitrary.
- Could make everything perception or everything action.
- The split is driven by the feature vector (just change the action function to get a different behaviour).
- Once the split is decided, we have to:
  - Map sensor data to feature vector;
  - Map feature vector to actions.
- Thus we have split the function f above into:

$$g : S \mapsto X$$

and

$$h : X \mapsto A$$

- How do we define g? We start with the feature set.
- There are 256 different possible sets of sensor values.
- We distinguish the important sets of these as *features*.
- For boundary following, the following set turn out to be sufficient:



In each diagram, the indicated feature has value 1 if and only if at least one of the shaded cells is *not* free.

- We could choose a different set of features.
- A different set might change the behaviors that we can program the robot with.
- The set of features tells us what the x_i are.
- This in turn gives us a way of defining g — for each x_i we can say what s_i define it.
- Of course, in real life, identifying features is not so easy…

- Now we have to define h.
- If all the x_i are 0, then the robot can move in any direction.
- We will make it go north if this is the case.
- Otherwise there is a boundary to follow.
- We follow it by:
  - If $x_1 = 1$ and $x_2 = 0$ then east
  - If $x_2 = 1$ and $x_3 = 0$ then south
  - If $x_3 = 1$ and $x_4 = 0$ then west
  - If $x_4 = 1$ and $x_1 = 0$ then north

- We can write these conditions as Boolean expressions.
- The condition for the robot to move east is:

$$x_1.\overline{x_2}$$

- And the condition for it to move north is:

$$\overline{x_1}.\overline{x_2}.\overline{x_3}.\overline{x_4} + x_4.\overline{x_1}$$

- We can also express the x_i as Boolean combinations of the sensor signals:

$$x_4 = s_1 + s_8$$

Production systems

- How do we represent the action function?
- We think about this because we need to find a way to code it.
- One convenient representation is as a *production system*, a collection of *production rules*.
- Each rule is written as:

$$c_i \longrightarrow a_i$$

  with a *condition* part and an *action* part.
- A production system is a list of such rules:

$$\begin{aligned} c_1 &\longrightarrow a_1 \\ c_2 &\longrightarrow a_2 \\ &\vdots \\ c_n &\longrightarrow a_n \end{aligned}$$

- The condition can be any binary-valued function of the appropriate feature vector.
- For our example it is just a simple Boolean function.
- To select an action, we look through the rules until we find a c_i which evaluates to 1.
- Then we execute the associated a_i.
- The a_i can be a primitive action, a set of actions, or a call to another production system.
- Usually the last rule in the system has condition 1 (ie. it is an "else" production).

- Thus, for our example, we could have the production system:

$$\begin{aligned} x_4\overline{x_1} &\longrightarrow \text{north} \\ x_3\overline{x_4} &\longrightarrow \text{west} \\ x_2\overline{x_3} &\longrightarrow \text{south} \\ x_1\overline{x_2} &\longrightarrow \text{east} \\ 1 &\longrightarrow \text{north} \end{aligned}$$

- This system will then run forever.
- This is pretty typical for agent/robot controllers — its job is just to keep running, it doesn't try to achieve anything.

- Another kind of production system will have an overall goal.
- Imagine that we want the robot to follow the boundary until it finds a north-east corner (like the top-left corner in the example) and then stop there.
- We can define another item in the feature vector:

$$x_5 = s_1 s_2 s_3 \overline{s_4 s_5 s_6} s_7 s_8$$

and then write the production system:

$$\begin{aligned} x_5 &\longrightarrow \text{nil} \\ 1 &\longrightarrow \text{b-f} \end{aligned}$$

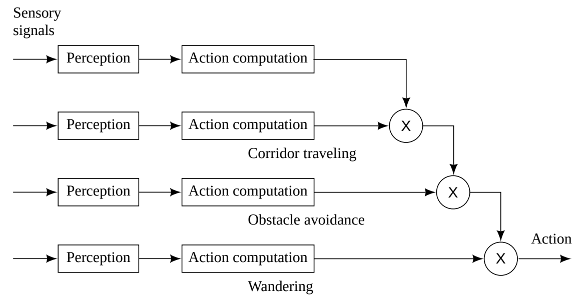
where nil is an action which does nothing, and b-f is a call to the previous production system.

- There are three points to make about this.
- First, in goal-achieving production systems, the topmost rule identifies the situation we are aiming for.
- Once this is acheived, we need do nothing more.
- Second, conditions and actions lower down the production system lead towards the achievement of the topmost condition.
- Indeed, action a_i is intended to bring about c_j where $j < i$.
- Third, we can build up a hierarchy of production systems, where systems lower in the hierarchy move the robot towards meeting the conditions of productions in systems higher up.
- This gives us a means of procedural abstraction.

- Systems of rules like this are called *teleo-reactive* (T-R) programs.
- Every action in a T-R program works towards the achievement of a condition higher in the program.
- It is typically easy to write such programs.
- T-R programs are also very robust.
- Even in the face of faulty sensor readings, carefully constructed T-R programs will get back on track.

Subsumption Architecture

- Another approach to combining simple sensory-driven behaviour:



- Each module receives sensory information directly from the world.
- If the sensory inputs match the preconditions of a module, it executes.
- Modules can *subsume* each other (in the picture upper modules can subsume lower ones).
- When module *i* subsumes *j*, then if *i*'s precondition is met, the program of *i* replaces that of *j*.
- So in the example:
  - The robot wanders until it has to avoid an obstacle;
  - Avoids an obstacle until it is travelling in a corridor.

- Subsumption architecture started with Brooks.
- Idea is that:
  - Build basic behaviour;
  - When that is refined, add a subsuming behaviour;
  - When that is refined, add another;
  - . . .
- So far as I know, the maximum “stack height” is not *that* high.
- However, there are other ways of making the approach more sophisticated.

- We can make the approach more flexible:
  - Rather than having a fixed set of behaviours, construct a task specific set.
  - (Plan, but in terms of behaviours not actions.)
- We can improve on subsumption.
  - Rather than having one behaviour replace another, merge behaviours.
  - (Imagine being able to do a weighted sum of actions.)
- Both these features are available in Saffiotti's THINKING CAP.

- How could we program this?
- As follows:

```
if <some condition>
   then <some action>
   else if <another condition>
            then <another action>
            else ...
```

- Here actions higher up in the compound if statement take precedence.

Summary

- This lecture introduced stimulus-response agents.
- These do not think; they just act.
- We looked at two approaches to implementing such systems.
  - Production rule systems.
  - Subsumption architecture.
- The next step will be to think about how to go from programming our robot this way to having to learn how to do this for itself.