Computer Science 773

Robotics and Real-time Control

SUBSUMPTION ARCHITECTURE

Programming complex robots – or the equivalent, programming robots to perform complex tasks – is a difficult job. The job rapidly gets harder as the tasks become more ambitious. This is often not primarily directly because of the intrinsic complexity of the task to be accomplished, but because there are more terms to consider and, because it's a combinatorial phenomenon, increasingly more possible combinations of terms which might have to be taken into account. The whole system acquires more and more possible modes of behaviour, each of which must be dealt with somehow in the final controller.

The conventional way of programming robots, which is perhaps the obvious way, is to decompose the task into a set of functions : perhaps primarily a vision function and a control function, which are decomposed further into primary vision, world view maintenance, world view interpretation, motion planning, motion execution, and so on.

While this is a rather appealing factorisation from a familiar top-down view, there is a sense in which it encourages complex interactions. Even though the robot might eventually be intended to do just one rather limited task, it is very difficult to reflect that limitation down to the level of the individual components. For example (an example which will turn up again later), if you are building a mobile arm robot which will only ever be required to wander round a room identifying and picking up Coca Cola tins and putting them away in the rubbish basket, how does this help you to simplify the world view ? If you think a lot about it, you will probably be able to do it, but the result isn't a very natural way of solving the problem.

An alternative view of the programming task is to think of it in terms of behaviour : if the required behaviour expected of the robot is not very complex, perhaps this view will lead to a more tractable analysis of the problem and a more effective solution. This view of robot programming is called *behaviour-based robotics*, and is generally regarded as originating in the work of Rodney Brooks around 1985. You might like to consider whether the "servo processes" which were used in the RSS programming language might perhaps have get somewhere near there a little sooner.

Brooks was particularly interested in developing intelligent robotic systems. In this field of research, the programming problem is extreme; if we can't even satisfactorily programme robots to perform well defined tasks, how can we hope to make them define the tasks as well ? He asserts that cognition can only develop when entities are required to solve problems for themselves in changing environments, which adds another level of complexity. Within this context, a model based on behaviour fits rather well. In practice, we don't learn new problems starting from the ground and working up; we learn to use the information we receive through the sensors we've got in new ways.

Brooks's ideas led to his development of the *"subsumption" architecture*. This is an attempt to mechanise the idea that complex behaviour can be regarded as the result of a collection of elementary behaviours operating at different levels of sophistication, and interacting in various ways. Generally, low-level behaviours provide services which can be used by high-level behaviours, and the high-level behaviours can inhibit the operation of the lower levels when they think they know better, replacing them with behaviours of their own. This combination of inhibition and replacement is called *suppression.* Action is initiated at the appropriate level by information from sensors, typically with high-level behaviours activated by very specific sensory information, and then inhibiting lower-level activities to make some appropriate response to the high-level stimulus. When the response is complete, the high-level procedure can lift the suppression, leaving it to the lower levels to carry on.

As an example, consider this set of behaviours (not tested in practice – I made them up) for a robot which clears up rubbish from the floor.

Behaviour	*Responding to*	*Description*
0	Switching on	Start the motors
1	Switching on	Wander about at random
2	Front high touch sensor	Go backwards
3	Rear high touch sensor	Go forwards
4	Small object seen on floor	Move towards small object
5	Low level touch sensors	Push
6	Expected motion detected	Lift the small object
7	Carrying an object	Go to the rubbish basket
8	At the rubbish basket	Put the object into the basket

As a first approximation, the behaviour number represents the level, so higher numbers suppress lower numbers. If you work it out, you find a few refinements are necessary (what happens if the front high touch sensor comes on when the robot is moving towards a small object ?), but apart from those a robot following that prescription would clear bits from the floor quite well.

The specification also identifies just what each sensor must be able to do. The touch sensors simply have to notice when they hit something; the vision system can be restricted to something which inspects the floor close to the robot; and so on. There's no need for a sophisticated world model, though there must be some way to find the rubbish basket. (Brooks describes this observation with the phrase "The world is its own model".)

Of course, there are some less attractive features of the approach. (Brooks doesn't say a lot about these.) It is pretty certain that my set of behaviours above wouldn't be adequate in practice to do the job; odd interactions and curiosities would turn up, and other behaviours might have to be invented to cope with these. In a more complicated system, a lot of effort can go into finding a suitable set of behaviours, and interactions between the behaviours, which will do the job you require, and it's at least arguable that the net difference in complexity between a behaviour-based system and a more traditional version might not be as great as one might expect.

Even so, it's an interesting idea, and worthwhile even if it just sets people thinking in different directions.

AN EXAMPLE.

This example is not too unlike my primitive case study; it picks up Coca Cola tins from benches and disposes of them. The main difference is that this one works.

The robot arm used is very simple; it has only two joints, and the linkages are such that the end effector always points in the same direction (downwards, with the gripper used in this exercise). The trunk does not rotate, but as the whole body can move that isn't a problem.

The gripper is the clever bit, so far as the machinery is concerned. It has several sensors :

- The *tip switches* and *wrist switch* detect when the gripper hits something when descending.
- The *finger beam* ("beam" in the diagrams which follow) is an infra-red beam from one finger to the other, and detects opaque objects between the finger.
- The *crossed IR* beams ("XIR" in the diagrams) are implemented by two infra-red emitter and detector sets, one at each side of the gripper. They are used in two ways : by detecting the light reflected back from anything at all they act as general fairly short range object detectors, while by using the detector of one set to detect light reflected from the emitter of the other, they give specific information on whether there is an object within the shaded area.

The "programme" is shown in the diagrams below. The first part controls the gripper, while the second controls the arm. (The controller for the movement of the robot as a whole around the room is not described in the paper from which most of this is taken; it says that similar methods are used, and there's a reference to another publication, which I haven't inspected.)

Each rectangular box is a behaviour, taking input from a sensor. What the box does with the sensor input is entirely up to the box, so you can't just assume that a box does something obvious with its input; notice that (for example) three boxes receive the "tips" input, and use it in different ways. It is interesting, though, that the box is a finite-state machine, so it is less versatile than a Turing machine, and much more realistic as something which could be implemented with neurons.

Circular boxes denote suppression. In a box labelled S, a signal along the line with the arrow suppresses a signal along the line without; in a box labelled D, the line signal suppresses the arrow signal.

Sensor inputs not marked on the gripper diagram are :

- force : monitors the force exerted by the gripper when it grips.
- pos : the robot's current position, as determined by its joint sensors.

Notice the several levels of behaviour, each capable of interfering in one way or another (usually by suppressing) with the layer below.

Where, you might ask, is the vision ? Ah, well, there isn't any. At the time of writing, you rolled the robot to a position in which there was a Coca Cola tin (or practically anything of the same or similar dimensions) in a plausible place in front of it, and set it going. Then the robot would pick up the tin. But it still isn't a bad trick.

And, so far as it goes, it works. Here are some traces of the robot's performance at repeatedly picking up a tin from a bench, picking up a tin stood on a block on a bench, and picking up a tin on the other side of a block on the bench.

WHAT'S WRONG WITH IT ?

Not much, really. It's an interesting and noticeably different way of programming a robot, and if only as a stimulus for thought is well worth while. But neither is it the solution to all our problems, which one might have been forgiven for believing Books thought.

It's possible to get something going with the minimum of specialised equipment designed to do the job – so the robot of the example will pick up tins from fairly arbitrary positions using a few rather simple sensors, and certainly without anything resembling vision or a world view.

But it won't recognise tins. Once you've added the bits to recognise a tin, you might well have enough additional sensory abilities to do away with some of the rather appealing but low-level features like bouncing along a table top searching for the tin. Unfortunately, there's no easy way to plug in the new abilities – the bouncing behaviour just has to go. (Better, perhaps, it has to be suppressed by the new clever version – then if the new clever version breaks down, we can easily revert to the old bouncing method.) Generally, the penalty for separating behaviours is that you're stuck with separate behaviours, and they can't easily cooperate or share resources (like, say, a world view) without at least bending the model quite a bit.

Neither is it clear how you'd go about designing a behaviour-based system for something very big. One advantage of traditional methods is that we do know ways of analysing problems of arbitrary size and producing programmes which are reasonably likely to work reasonably well. It isn't clear that you can do the same by analysing behaviours.

It does look as though Brooks has retreated a bit from his rather uncompromising pro-subsumption (not to say nothing-but-subsumption) position of some years ago. I think it's fair to say that his position has evolved. Now he's trying to build a humanoid robot called Cog (http://www.ai.mit.edu/projects/cog/).

REFERENCE.

J.H. Connell : "A behavior-based arm controller", *IEEE Trans. Robotics Automation* **5**, 784 (1989)

Alan Creak,
April, 1997.