Take as an example a car assembly line, on which a team of robots works on the cars as they pass by. All you want to do is to program the robots so that they will assemble the cars in the correct manner. However, programming the robots takes time, and money is lost each time the production line is halted. You might decide to set up a dummy assembly line with some brand new robots with which to develop the new programs. But this, too, is expensive and can easily lead to another snag — the problem of robot choreography, which we considered earlier in this series. It is vital to ensure that robots working together do not interfere with each other's movements. This is not just a matter of convenience — a large industrial robot capable of moving heavy loads could easily damage another robot if it should run into it. And it's not only the robots that may be damaged — an improperly programmed robot that spends its time welding car doors shut would soon prove a problem!

Chorus Line
The complexity of movement and synchronisation required in real robotics applications can be clearly seen in this section of the Ford Sierra assembly line

COURTESY OF FORD

The obvious answer is to carry out computer simulations of each robot's actions so that the user can see how they will interact. This way the cost is low and nothing is damaged. Once the simulation is complete and everything looks satisfactory, the programs that have been developed can easily be transferred to the real robots, which can then be safely left to carry out their designated tasks.

In this article, we will demonstrate the principle of robot simulation with a program, Robot Arm, that simulates a 'pick and place' robot arm with two degrees of freedom. It has no sensors, so you must guide it yourself, controlling the shoulder and elbow joints and the grab mechanism in the end effector, in order to pick up an object and then place it somewhere else. Additionally, you should refer back to the maze-solving program detailed on page 722, which demonstrates how a robot may be progammed to find its way to the centre of a maze. This program is, in effect, a computer simulation of how a 'real' robot would attempt to reach its goal. It mimics the actions of a robot fitted with a simple touch sensor, and it finds the correct path by simply advancing into empty spaces until it meets a dead end, whereupon it returns to the last junction it encountered and then tries a new route. This is hardly a sophisticated model, but it does show how a computer program can be used to simulate a robot's movements. The 'robot' in the program obeys a fixed set of rules and 'maps out' the environment. If, within the program, the robot had direct and immediate access to the positions of all the maze components, it would be able to move directly to its goal. In our program, it does not have this information and so must use a trial and error technique.

Similarly, the Robot Arm program mimics the behaviour of a robot that has no sensors at all. This program contains a model of the robot's environment and a model of the arm itself, and you must ensure that these two models will interact only as they would in real life. So you cannot pick up an object with the arm unless the arm is positioned correctly. And you cannot move the arm below floor level, as that would be impossible if the robot arm was real. Although we are using computer graphics, in which one line (representing the arm) may easily cross another line (the floor), for an accurate simulation it is essential that these lines do not cross. And when the robot drops an object, that object must not remain in its current position — your simulation must allow gravitational effects. If this is ignored, you would certainly not be able to develop a safe simulation of a pick and place robot for handling eggs!

ADDING REALISM

There are very few limitations on what can be achieved using computer simulation — and, in most cases, the more complex the simulation is the more fascinating it becomes. Such a simulation can be even more entertaining than simply playing with 'real' robots, for the simple reason that, using a simulation, you can design any robot you like; programming the correct details of the robot and its world can lead to a better understanding of robots and of the way in which the physical world works. Look again at the Robot Arm program. You will see that when the robot drops an object it falls to the ground and stays there. To make the model even more realistic, the program could be altered so that the object accelerates as it falls, thus obeying the law of gravity. And perhaps, on hitting the ground, it should bounce? The possibilities are many, and the program is there for you to adapt, adding new and more realistic features to make your simulation as lifelike as possible.

Designing computer simulations can be very similar to developing computer games software. The big difference is that a simulation must represent the real world as accurately as possible. Achieving this accuracy may be difficult but, once achieved, the simulation can be considerably more satisfying than merely playing a computer game.