
CLOSED-LOOP CONTROL OF A SINGLE AXIS BY COMPUTER PROGRAM
EXERCISE 3

Object To develop initially a closed-loop program for controlling the shoulder of the robot, and use it to demonstrate the effectiveness of feedback.

To extend that program to provide sequential control of each axis in turn.

Exercise 3 – Introduction Exercise 2 showed how to command the robot to move, and how to read the position feedback signals it provides. In this exercise we will get the robot to perform automatically a sequence of movements. This involves organising all the required sequences of commands into a computer program.

The general form of these commands will be as detailed in Exercise 2. You will have to select them and organise them to produce the required effects.

In the first part, Exercise 3.1, we shall do this for just one axis, for simplicity. However, by thinking ahead we can do it in such a way that it is easy to alter the program later to work with all six axes.

This applies especially in the case where programs will be written in BASIC. Suppose that variables, for example S denoting set-position (the target position to which we want the robot to move), and constants such as the value of the axis-selecting bit of a port A output, are wanted for one axis. Suppose too that they depend on which axis that is. The trick then is simply to recognise that if we define an index X and use the indexed variable S(X) instead of just S, and so on, all that need to be done to use the same segment of program for each axis in turn is to change the value of X.

Exercise 3.1 – Defining the algorithm

Writing a program is a great deal easier if you start with a clear idea of exactly what you want it to do, how the program will know when and how to do it, and so on. The collection of statements which defines the behaviour for every possible condition the robot will meet is called its algorithm.

It should be clear from Exercise 2 that initialisation of the robot is necessary, and you may like to leave the initialisation routine given in Exercise 2.1 unchanged, to be sure that it will work. The robot will need to be told where to go to, how to decide which way to go, and when to stop. Consider the following

Initialise

```

REPEAT
  input (Set-position) (*from keyboard*)
  REPEAT
    form Error-signal (*need to read feedback signal to do this*)
    IF Error-signal > 0 THEN move in +ve direction
    IF Error-signal < 0 THEN move in -ve direction
    (* can implement this by forming the corresponding
    command bytes*)
    ELSE stop
  UNTIL Error-signal = 0
UNTIL forever

```


This would certainly get the axis in question to move to the correct position. The next question is what happens when it gets there. In practice nothing stops instantly, so that the robot will overshoot, find a reversed position error, and so hunt backward and forward trying to find the zero-error position. It may do this indefinitely.

A safer algorithm will therefore remember in some way in which direction the motion started, and will exit the inner REPEAT loop when the measured position has reached the set position or passed it in that direction.

Second stage of algorithm

REPEAT

Halt-command := false

input (Set-position) *(from keyboard*)

form Error-signal

IF Error-signal > 0 THEN Direction := +ve
(*use port B command byte*)

IF Error-signal < 0 THEN Direction := -ve

ELSE Halt-command := true

REPEAT

form Error-signal

IF (Direction = +ve) AND (Error-signal > 0)
THEN move in +ve direction

IF (Direction = -ve) AND (Error-signal < 0)
THEN move in -ve direction

(*can implement this by forming the corresponding command bytes*)

ELSE stop; Halt-command := true

UNTIL Halt-command = true

UNTIL forever

Repeated overshoot of desired position is avoided by using an inner repeat loop as it is described above. The magnitude of this overshoot will depend upon the speed with which an axis is moving when the desired position is reached, i.e. more speed, more overshoot. To improve matters, use can be made of the dual speed capability.

If an axis is far from its set position it may be moved rapidly to reduce the error. When it reaches a preset distance from its goal, the speed of movement can be reduced.

To implement this, another conditional or test statement, included within the inner repeat loop, should be made to check position error and reduce speed when appropriate. The limits for positive and negative position errors need not be the same.

You will need to develop the algorithm further to specify how the error signal is to be formed. Since this has to be done several times a subroutine is appropriate which can be called repeatedly. It may involve further subroutines to reinitialise port A to input and/or output, and to operate the analogue-digital converter. You may find it convenient to avoid the complication of a separate Halt-command, and simply make Direction = 0 equivalent to the halt command. A useful refinement is as follows. Suppose that movements are to take place in various axes, but one axis remains still. It is convenient to choose a particular set-position value, such as 0, to have the effect of causing no motion. This is easily done by letting (set-value = 0) force a zero error signal. The benefits are that it is not necessary repeatedly to remember and insert the coordinate for the stationary axis, and it will stay still instead of hunting back and forth in search of the ideal position. (The position designated 0 is no longer available, but will often be off the scale anyway).

Try to work out your own version before turning the page. At this stage also start defining the names of the variables you will use, remembering the point made earlier about using indexed variables. For example you might decide to use $S(X)$ as the setpoint for axis X , and allocate to index X values which ascend in the order of most probable movement, thus:

$X = 0$	swivel
1	shoulder
2	elbow
3	wrist flexure
4	wrist rotation
5	grip

On this basis your set point for the first part of this exercise would be $S(1)$.

(PROGRAM Closed-loop-shoulder
 (* 'X' is short for indexX))

X := 1

REPEAT

Halt-command (X) := false

input (Set-position (X)) (*from keyboard*)

form Error-signal (X)

IF Error-signal (X) > 0 THEN Direction (X) := +ve
 (*use port B command byte*)

IF Error-signal (X) < 0 THEN Direction (X) := -ve

ELSE Direction (X) := halt

REPEAT

form Error signal (X)

IF (magnitude of Error-signal > preset value) THEN (set fast speed
 parameter for use by Move subroutine)

IF (Direction (X) = +ve) AND (Error-signal (X) > 0)
 THEN Move (Direction (X), Axis (X))

IF (Direction (X) = -ve) AND (Error-signal (X) < 0)
 THEN Move (Direction (X), Axis (X))

ELSE Direction (X) := Halt

UNTIL Direction (X) = halt

Halt (Direction (X), Axis-X)

UNTIL forever

SUBROUTINE Move (Direction (X), Axis (X))

Port B = (select all restrictor bypass valves)

Port A = Axis (X)

Port B = : Direction (X)

Port A = Axis (X)

END

SUBROUTINE Halt (Direction (X), Axis (X))

Port B := (select all valves)

Port A := 0 (*closes the valves*)

END

SUBROUTINE Form-error-signal (Set-position(X))

Reinitialise port A to input

Set up A-D converter to read position detector (X)

start A-D conversion

Error-signal (X) := Set-position (X) — port A

Reinitialise port A for output

END

END (*Closed-loop-shoulder*)

At this stage it should be possible to start programming.

Loading the program

Depending on the availability of computers, you may have to write your program initially on paper; or it may be written directly into a computer (which for this purpose need not have the robot connected to it), and (again subject to availability) it may be saved on cassette or disc.

When it comes to running the program to control the robot, the experimental set-up is the same as for exercise 2. If electrical connections have to be made or broken, switch the power off first. Check that a ribbon cable is plugged into the connector labelled I/O on the back of the processor box, and into the computer interface (which may be inside the computer).

When the equipment is ready, connect its supply cables to the supply line and switch on the power to the computer, but NOT at this stage to the robot.

Remember that the robot must be initialised the moment after it is switched on. If your program is being typed in while the robot is connected, start by entering just the initialisation routine (as exercise 2.1), and then switch the robot on. If you have a whole program already written, enter an END statement after the initialisation, then switch on. One or two direct (un-numbered) instructions on the lines of Exercise 2.2 can then be used to check that the robot is working correctly if desired, before the END statement is removed to enable the rest of the program.

If odd behaviour is observed, the use of temporary instructions inserted in the program (e.g to display the value of a variable at a particular stage) can be very helpful in diagnosing problems.

Exercise 3.2

If your algorithm and program have used indexed variables in the manner suggested, it should not be very difficult to convert the fixed index denoting the shoulder axis into a FOR . . . NEXT loop which will cause each axis in turn to be positioned. It is as well to leave the grip axis out of the loop, to be dealt with by a separate routine after the FOR . . . NEXT loop is concluded.

This is because, first the position of the arm will have to be correct before the grip is operated and second, its control function will be different. It is suggested that the grip is closed slowly and its position repeatedly monitored to determine when the jaws have stopped (presumably around an object to be lifted). A simple time delay can be used to allow sufficient time for the jaws to open (at fast speed) when desired.

It is suggested that a more interesting performance will be given by the robot if it does not stop for a keyboard entry every time one axis has been positioned. An alternative is to load a batch of data with the program. In a BASIC computer this is done by having in the program a DATA entry in the form DATA a, b, c, d, e, f, g, h. . . where a, b, etc denote setpoint values in groups of six. The last in each group will control the grip action.