DIY robotics and sensors on the Commodore computer:
practical projects for control application
by John Billingsley, 1984
ISBN 0 946408 30 0

CHAPTER 8
Interfacing a Robot

A fully-fledged robot has seven degrees of freedom, that is to say it requires seven independent motors to drive it. The 'end effector' (a fancy term for 'hand') must be able to move in three dimensions, and for any given position it should be able to swivel about three more axes. A further channel is needed for 'open' or 'close', although this is often a simple on/off valve working a pneumatic gripper. Educational robots, such as the Armdroid, sacrifice one of the 'wrist' axes, but give continuous grip movement. This reduces the number of channels to six. If these are driven by stepper motors, how can we interface them to the computer? In the last chapter, two channels of stepper motor were interfaced to the user port with the use of all eight bits, so for six channels we must find some new technique to command them all. It is necessary to include an address as part of the user-port data, which can be decoded within the robot itself.

Multi-stepper control

The user port provides eight bits: a stepper motor needs four bits to define a half-step position (unless you are happy interfacing using the scale of three: N, off, S). That leaves four bits for housekeeping. From three of these bits, an address can be constructed to address eight channels. The addressed channel will now capture the motor signals in a four-bit latch, and carry on driving the motor lines until told to do otherwise. Now we need a 'strobe' signal as well, so that we can tell the circuitry 'the motor lines have finished changing, the address lines are settled, catch this data now and use it'.

The connections to the somewhat ageing Armdroid on which the programs of this chapter have been tried out are as follows:

PB0	PB1	PB2	PB3	PB4	PB5	PB6	PB7
Strobe		------Channel------		N----------	E----------	W----------	S

Note that the 'compass' bits are shuffled in comparison with the last chapter, and so the data statements for the codes are different. Putting the 'channel' bits in a shifted position does complicate the code calculation a little, but everything comes out in the wash.

Since the strobe will be active when low, the procedure for outputting a new command is as follows:

1. Look up the code for the desired motor position.
2. Add on 2*(channel number), hold the result in CO, say.
3. Set the strobe bit (bit 0) high in CO.
4. Output CO to the user port.
5. Set bit 0 of CO low; output CO to the user port.
6. Set bit 0 of CO high again; output CO to the user port.

Putting the algorithm into software

Let us adopt our usual technique of defining data and arrays 'up in the sky', with housekeeping at 10000:

```
10000 DD=59459:PO=59471:KB=547:KO=255:REM ANTIQUE
      PET ******
10000 DD=59459:PO=59471:KB=166:KO=255:REM 8000, 4000
      PET ****
10000 DD=56579:PO=56577:KB=197:KO=64 :REM
      COMMODORE 64 *****
10010 DIM DR(7): REM DRIVE VALUES WITH STROBE ALREADY
      HIGH
10020 FOR I=0 TO 7: READ J: DR(I)=16*J+1: NEXT
10030 DATA 1,3,2,6,4,12,8,9: REM N-E-S-W
10040 MA=254: POKE DD,255 :REM SET TO OUTPUTS
```

Now we can look at a subroutine for driving one of the motors. If the channel number is set in variable CH, whilst the step angle is held as value V (measured as number of steps from the switch-on position), then the following routine will output the required code to the motor:

```
9000 CO=DR(V AND 7)+CH*2
9010 POKE PO,CO: REM OUTPUT THE CODE, STROBE BIT HIGH
9020 POKE PO,CO AND MA :REM 'AND' WITH MASK — STROBE
     LOW
9030 POKE PO,CO: REM STROBE HIGH AGAIN
9040 RETURN
```

This routine takes one or two short cuts from the algorithm above, and will simply set up one motor drive to command a given position.

Troubleshooting the connections

For the Armdroid's end of the connections, you will have to refer to the handbook — the edge-connector has been modified in recent issues. If you have constructed your own circuit (perhaps from the one given later in this chapter) then it should already be familiar to you.

Apart from reassuring yourself that something will really happen, the next test will establish which channel numbers control which axes of the robot, and in which direction. Enter the following simple test program. Since it will get progressively overwritten as you work through the chapter, you might like to save it at each stage for future use.

```
   10 GOTO 10000
  100 INPUT "CHANNEL NUMBER, DISTANCE":CH,DI
  110 FOR V=0 TO DI STEP SGN(DI)
  120 GOSUB 9000:NEXT
  130 GOTO 100
10900 GOTO 100:REM AT END OF HOUSEKEEPING
```

Now check out each of the channels in turn, entering values from 0 to 7 for the channel number, and around 50 or −50 for the distance. Two of the eight possibilities will of course have no effect, since only six channels are used. Make a careful note of the axis and the direction, in terms of up, down, pivot left, right, forwards, backwards, gripper open and close. The Armdroid uses two motors at a time to rotate the wrist, and to swivel the wrist up and down. Make a note of which does which. A program with a machine-code output routine can afford the time to unscramble separate commands to twist and tilt the wrist. In BASIC, this makes the system rather slow, so here we will at first drive just one motor at a time.

Keypress commands

Now let us add a routine so that holding down a key will drive a motor. It is important to display a menu of keys on the screen, showing which key does what — nothing is more frustrating than having to guess. To get smart-looking upper and lower cases legends, put the machine into lower-case mode before typing in the program. Of course, when listed in this mode all the commands in the software will appear in lower case, but it is clearer to show them here in the usual capitals.

Now we must allocate a key to each movement, up, down, left, right, forwards, back, open and close, plus wrist up and down and rotate. The keys U, D, L, R, F, B, O and C are obvious enough as choices, but we must choose four more for the wrist — how about W, Q, T and Y? To each key will correspond a channel number and a direction. We must add a set of data statements to the housekeeping to sort them out — use your results from above to correct the values below, to give the correct movements. The values given here are the ones which Richard has sorted out for his particular Armdroid:

```
10100 DIM CH(5),HE(5):FOR I=0 TO 5:READ J
10110 CH(I)=2*J:NEXT:REM CHANNEL CODES
10120 DATA 1,3,5,4,2,6:REM U/D, L/R, F/B, O/C, WRIST
10130 C$="UDLRFBOCWQTY":REM COMMAND KEYS
```

You should swap the pairs of letters in C$ as necessary so that the first corresponds to a positive direction, the second negative. The array HEre(I) is used to remember the current position of each motor, and will become very useful later. Now we patch in a routine at 1000 to display the menu, read the keypress, interpret the command and execute it:

```
1000 PRINT CHR$(147); "Up         Down"
1010 PRINT"Left            Right"
1020 PRINT"Forward         Back"
1030 PRINT"Open            Close"
1040 PRINT"Wrist —         Q"
1050 PRINT"Twist —         Y"
1100 GET A$:K=PEEK(KB):IF K=KO OR A$="" THEN 1100
1150 FOR I=1 TO LEN(C$):IF A$<>MID$(C$,I,1) THEN
     NEXT:GOTO1000
1160 J=I:I=LEN(C$):NEXT: REM CLOSE LOOP NEATLY
1170 CH=INT((J-1)/2):DI=2*(J AND 1)-1:V=HE(CH)
1180 V=V+DI:GOSUB9000:REM TAKE A STEP
1190 IF PEEK(KB)=K THEN 1180:REM KEEP STEPPING IF
     PRESSED
1200 HE(CH)=V:GOTO1000
```

To make this work, we need to change line 9000 to make use of our channel unscrambler:

```
9000 CO=DR(V AND 7)+CH(CH)
```

We must also remove the old 100–130, writing instead:

```
100 PRINT CHR$(14):GOTO 1000
```

As it now stands, the program should run reasonable quickly. There is, however, a very simple dodge to speed it up a little. To access a variable, BASIC must scan down the list of variables in the order that they were first encountered until it comes to the correct one. Thus the variables declared earliest will operate the fastest. Looking through the program, we see that CO, PO, V, DI, CH, KB and K are used every step, some several times. If we change line 10 to the strange-looking form:

```
10  DIM CO,PO,V,DI,CH,I,R,KB,K:GOTO 10000
```

then there should be a satisfying increase in speed. (I and R have been included for later use.)

Programmed movements

When every move must be controlled from the keyboard, the robot is just a toy. If, however, we can program into it a sequence of movements which it can perform automatically, then we can start to explore its serious use. We therefore need to be able to record each target point, and we need a second command level which will let us switch between the 'teach-mode' section (the program we have already tested) and the routines which will drive the robot automatically. For this, we use a second menu starting at line 100:

```
100  PRINT CHR$(147);"Teach,         Perform,"
110  PRINT "Repeat,      Clear,"
120  PRINT "Save,          Input"
130  GET A$: IF A$ = ""THEN 130: REM WAIT FOR A KEY-PRESS
140  FOR I = 1 TO 6:IF A$< > MID$("TPRCSI",I,1) THEN
     NEXT:GOTO100
150  J = I:I = 6:NEXT: REM CLOSE THE FOR...NEXT LOOP
     NEATLY
160  ON J GOTO 1000,5000,5100,2000,6000,7000
```

So far, we have only written the 'teach' routine at 1000, and even this needs some modification. We must allow the selection of any point to be added to the manoeuvre, and we must allow a return to command mode. We therefore fill in the gaps in the program with:

```
1060  PRINT"Point          End teach"
1110  IF A$ = "E" THEN 100
1120  IF A$< > "P" THEN 1150
1130  IF NP = MP THEN 1000:REM TOO MANY POINTS
1140  NP = NP + 1:FOR  I = 0  TO  5:PT(I,NP) = HE(I):NEXT:GOTO
      1000
```

Now for some more housekeeping. The array PT(5,MP) must be declared to hold the points. The limit, MP, can conveniently be set at 20, but if you wish you can choose a much bigger number.

```
10300 NP=0:MP=20:DIM PT(5,MP)
```

To keep track of progress, we can display the present coordinates and the number of points set with:

```
1080 PRINT:PRINT NP;"POINTS"
1090 FOR I=0 TO 5:PRINT HE(I):NEXT
```

That completes the teach mode section of the program. We are left with the task of writing the routine to perform the set of actions.

Routines to perform a manoeuvre

Having remembered the moves, we can perform them by looking up target points in turn, and then calling a routine at 8000 to move from HEre(I) to TArget(I). If we wish to perform them just once, then we can GOTO 5000, and:

```
5000 IF NP=0 THEN GOTO 100:REM NO POINTS
5010 FOR P=1 TO NP
5020 FOR I=0 TO 5:TA(I)=PT(P,I):NEXT
5030 GOSUB 8000
5040 NEXT P
5050 GOTO 100
```

If we wish to repeat the cycle until a key is pressed, then we can use a section of program at 5100:

```
5100 IF NP=0 THEN GOTO 100
5110 FOR P=1 TO NP
5120 FOR I=0 TO 5
5130 TA(I)=PT(P,I):NEXT
5140 GOSUB 8000
5150 NEXT P
5160 GET A$:IFA$=""THEN 5110: REM NO KEY, ROUND AGAIN
5170 GOTO 100
```

This has still not resolved the problem of what to put at 8000. For best speed of response, we will at first move just one axis at a time, although we will go on to consider diagonal movements. After declaring the array TArget(5) with:

```
10200 DIM TA(5)
```

we can compare each channel of TArget against HEre, and, if necessary, move accordingly.

```
8000  FOR CH = 0 TO 5
8010  IF TA(CH) = HE(CH)THEN NEXT:RETURN
8020  FOR V = HE(CH) TO TA(CH) STEP SGN(TA(CH)-HE(CH))
8030  GOSUB 9000:NEXT:HE(CH) = TA(CH)
8040  NEXT:RETURN
```

Now, to complete the command routines we can add a 'clear' one-liner:

```
2000  NP = 0:GOTO100
```

and we should also add routines at 6000 and 7000 to record and retrieve a set of movements. For now, plus the lines with:

```
6000  GOTO 100
7000  GOTO 100
```

and create your own routines when you have checked out the rest of the program.

As it stands, the program will act as a quite acceptable robot driver, but the execution of the manoeuvres, one axis at a time, will seem inelegant. It would be far better to interpolate the movements with all axes firing together. Unfortunately, the program which follows in the next section is excruciatingly slow in its performance, and the only really satisfactory way to achieve the result is by using machine code.

Simultaneous movements

We want a routine which will set up all six channels of the robot, and will interpolate a manoeuvre so that all channels can be made to move at once. This is another task for the binary-rate-multiplier, this time firing on six cylinders. Our starting position is HEre, an array with six elements, one for each motor axis. The destination is held in TArget, and we can work through all six axes, finding which one demands the greatest change. Now it is a straightforward job to calculate the ratios, and to make a step according to the overflow of a REgister, just as in the last chapter. We need some more arrays along the way, and the housekeeping routine at 10200 becomes:

```
10200  DIM TA(5),RA(5),WA(5),RE(5)
10210  FOR CH = 0 TO 5:HE(CH) = 0:RE(CH) = .5
10220  V = 0:GOSUB9000:NEXT:REM ZERO MOTORS
```

Now we can write a subroutine to move all six channels from HEre to the TArget position:

```
8000 REM MOVE FROM HERE TO TARGET : FIRST FIND LONG-
     EST MOVE
8010 RM=0:FOR CH=0 TO 5
8020 RA(CH)=ABS(TA(CH)-HE(CH)):REM   WORK   OUT   DIS-
     TANCE AND
8030 WA(CH)=SGN(TA(CH)-HE(CH)):REM   DIRECTION   FOR
     EACH AXIS
8040 IF  RA(CH)>RM  THEN  RM=RA(CH):REM  FIND  MAX
     DISTANCE
8050 NEXT:IF RM=0 THEN RETURN : REM NO MOVE
8060 FOR CH=0 TO 5
8070 RA(CH)=RA(CH)/RM:NEXT:REM RATES NOW IN
     RANGE 0 TO 1

8100 FOR R=1 TO RM: REM NOW WE ARE READY TO MOVE
8110 FOR CH=0 TO 5
8120 RE(CH)=RE(CH)+RA(CH)
8130 IF RE(CH)< 1 THEN 8160
8140 RE(CH)=RE(CH)-1:HE(CH)=HE(CH)+WA(CH)
8150 V=HE(CH):GOSUB 9000 :REM MOVE MOTOR
8160 NEXT CH
8170 NEXT R
8180 RETURN : REM NOW HERE=TARGET
```

Lines 8100 to 8180 are written with the aim of being easy to understand. Some improvement in speed can be made at the expense of clarity. After trying the first version, try substituting this:

```
8100 FOR R=1 TO RM:FOR CH=0 TO 5:I=RA(CH):IF I=0 THEN
     8130
8110 I=I+RE(CH):IF I< 1 THEN RE(CH)=I:NEXT:NEXT:
     RETURN
8120 RE(CH)=I-1:V=HE(CH)+WA(CH):HE(CH)=V:GOSUB 9000
8130 NEXT:NEXT:RETURN
```

8140–8180 are now redundant and should be deleted.

Even if you use only the program listed here, you will be able to teach the robot a routine which it will perform with interpolated movements. I still don't claim that the execution will be fast — you will soon want to perform the binary-rate-multiplier functions in machine code, and include a ramp

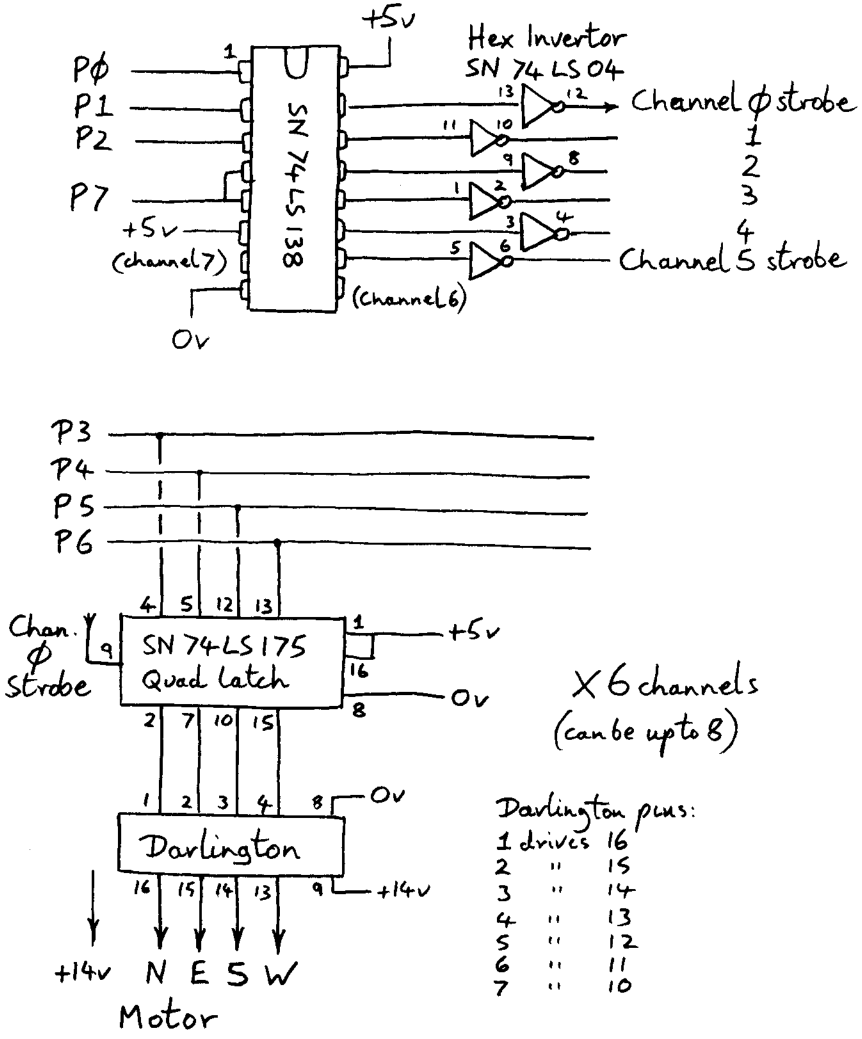

Figure 8.1 Circuit for 6-axis robot — decoder, latches, drivers

speed-up routine to achieve top speed without breaking away. Some elegant programs (eg MEMROB, written in collaboration with Tim Dadd and distributed by Colne for using an Armdroid with a PET) link the interpolation and output functions to the computer's interrupt, and communicate between BASIC and machine code by planting values in an array of variables. In this way, the BASIC part can do its 'thinking', working out the speed ratios for the next move at the same time as the present move is being made. The listing of MEMROB has baffled multitudes, and has little teaching value.

Building your own robot

You will first need half-a-dozen stepper motors. The ID35 mentioned in the last chapter will do nicely — it is the one used in the Armdroid. The Darlington drivers have been covered pretty thoroughly in Chapter 6. That leaves only the channel decoders and the four-bit latches — and the power supply. The supply detailed in Chapter 1 should be adequate, and will cost less than a single stepper motor. A circuit diagram for the 'innards' of the robot is drawn in **Figure 8.1**, and this should do all that you need.

For a do-it-yourself mechanical design, you can either follow close on the heels of the commercial robots, or you can be more adventurous. When you start to examine the number of ways you can link six motors together, the choice is amazing. Your geometry can be cartesian, polar, cylindrical-polar or a variety of strange hybrids. Let us start by looking at the 'conventional' robots.

Robot anatomy

The first axis of movement is a rotation of the whole assembly about the vertical. You can 'humanise' this by thinking of it as swivelling about the waist. Next, the shoulder joint allows the arm to tilt up and down, so that, using these two motors alone, the hand could reach any point on the surface of a sphere. Next comes the elbow joint. As this bends, the arm is effectively shortened, although the hand now moves in a way which needs more and more trigonometry to describe it. In principle, the robot should now be able to reach any point within a sphere, but if, for example, the upper arm is not the same length as the forearm, there will be some unreachable zones. The wrist joint should now be able to swivel both up-and-down and left-to-right. The Unimation Puma instead uses a movement like the human wrist, where the up-and-down hinge is an axis which can in turn swivel about the line of the forearm. The Armdroid leaves one of these movements out. Now the Puma can in effect line up a screwdriver with a screw in any position; the final axis twists the screwdriver to drive the screw.

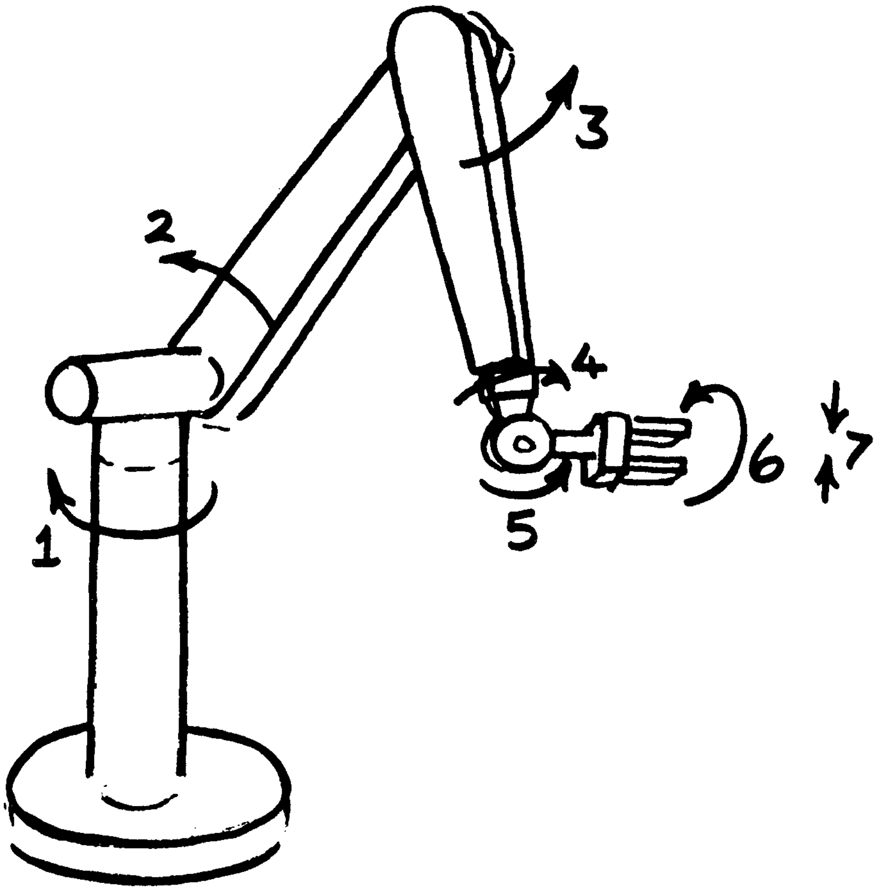

Figure 8.2 Axes of a 'conventional' robot

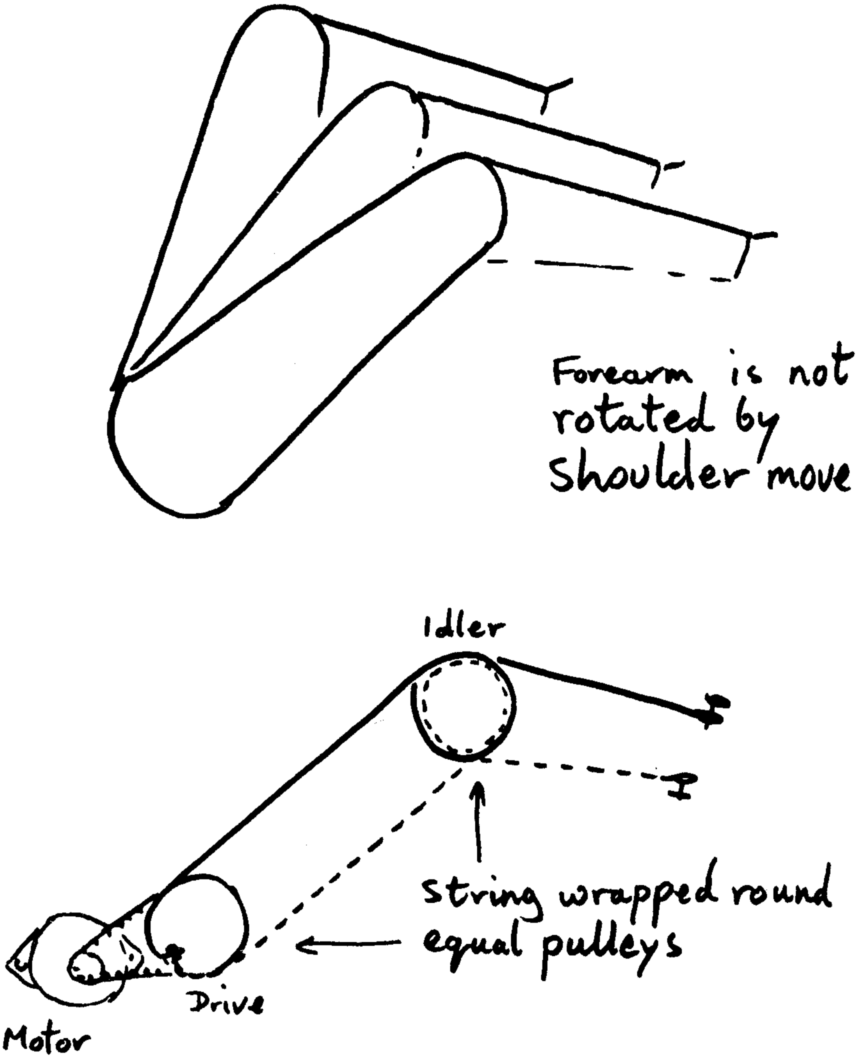
Forearm is not rotated by shoulder move
Idler
string wrapped round equal pulleys
Drive
Motor

Figure 8.3 Stringing to obtain parallel forearm movement

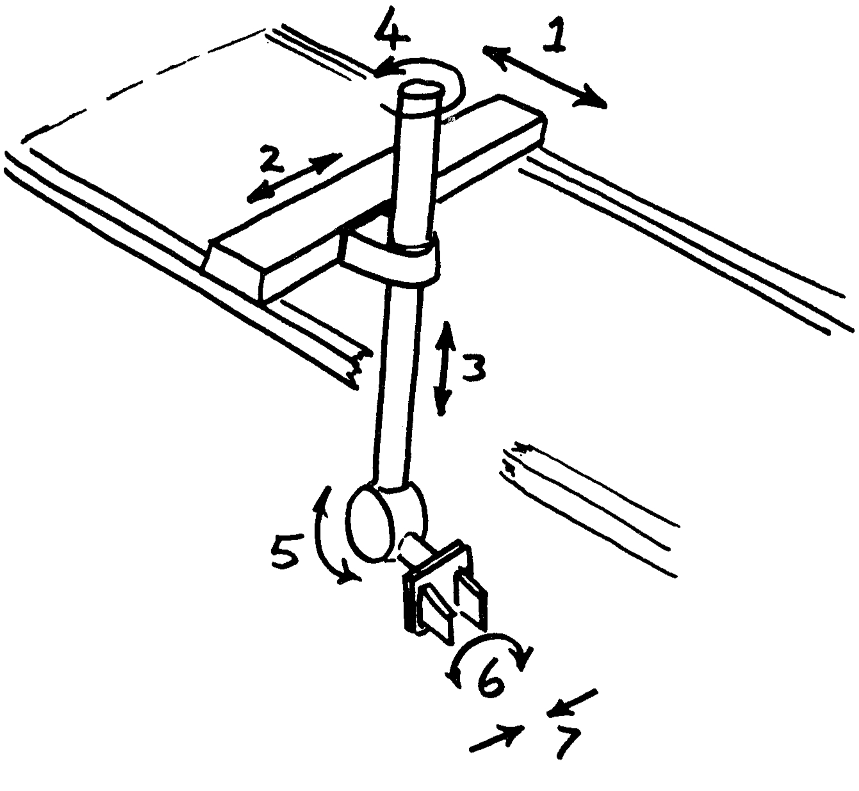

Figure 8.4 Cartesian robot arrangement

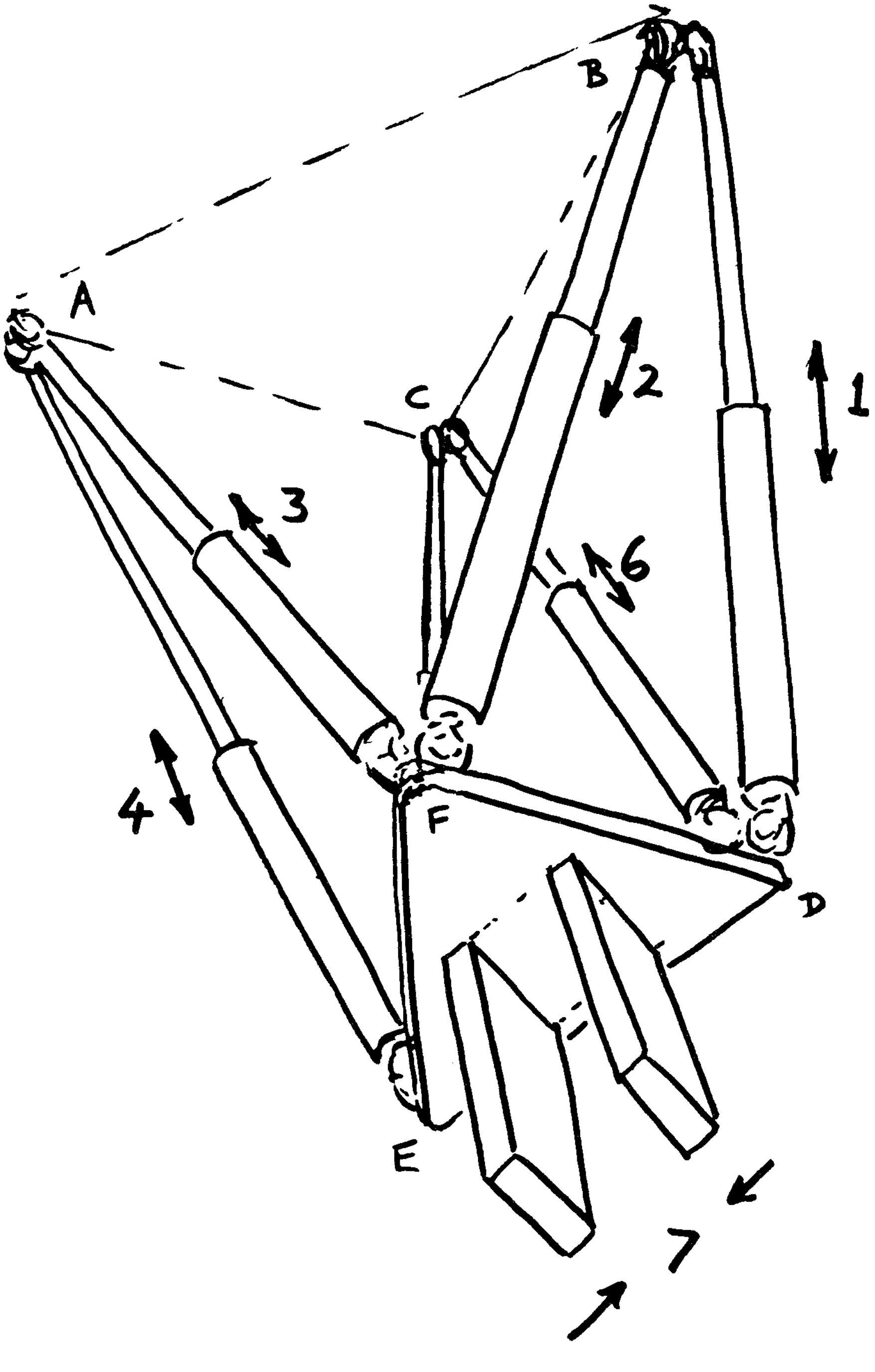

Figure 8.5 A variation on the 'Gadfly'

Some sophisticated robots, such as the Puma, perform laborious computations, allowing the user to specify that the hand should move in a straight line and that the tool should not rotate in space. New positions are calculated for the motor axes up to forty times per second, and the movement is then smoothed out by 'rate control', similar to the techniques described earlier. It is a challenging exercise to try!

Even when the geometry is settled, there are many ways to connect the motors to the axes. The Puma uses brute force, so that the entire leverage of the arm and its load will appear at the shoulder joint. The Armdroid on the other hand uses a cunning bit of string-work, so that as the shoulder rotates the upper arm, the forearm remains parallel to its former position. This effectively halves the leverage of the load on the shoulder motor — although it does not do a lot for the string which is annoyingly apt to break.

The IBM robot is cartesian, and bears a strong resemblance to an overgrown graph plotter for controlling the X and Y axes. The Z axis is an even more overgrown pen-lift, raising and lowering a bar which can rotate to provide the first of the wrist axes. All the problems of straight-line movement are solved at a stroke, but tracks and pulleys are now needed in place of pivots and levers.

When computing power is let loose, anything goes. Considerable industrial research is being put into developing a device using six extending rods, driven by motors and leadscrews. Imagine a triangle ABC fixed to the floor, with the tool attached to a movable triangular plate DEF. The plate is held up by six rods AE, AF, BF, BD, CD and CE. As these vary in length, so the plate can be moved in three dimensions and rotated about three axes. If you have an idle moment, calculate the relationship between the lengths and the position of the plate, or more especially the lengths required to place the plate in any particular position — a prize is offered for the simplest solution! No wonder it is called the 'Gadfly'.

These variations hardly scratch the surface of the possible combinations. If you connect up the six stepper motor channels, you can try any number of 'lash-ups' using cardboard, string and balsa wood before immortalising your design in aluminium or steel. Good luck!

Robot Control Program

NB. Change line 10000 if using a PET.

```
10 DIM CO,PO,V,DI,CH,I,R,KB,K: GOTO 10000
100 PRINT CHR$(147);"TEACH,    PERFORM,"
110 PRINT"REPEAT,   CLEAR"
120 PRINT"SAVE,     INPUT"
130 GET A$: IF A$="" THEN 130:REM AWAIT KEYPRESS
```

```
140 FOR I=1 TO 6
145 IF A$<>MID$("TPRCSI",I,1) THEN NEXT:GOTO 100
150 J=I:I=6:NEXT: REM CLOSE 'FOR' LOOP NEATLY
160 ON J GOTO 1000,5000,5100,2000,6000,7000
1000 PRINT CHR$(147);"UP          DOWN"
1010 PRINT"LEFT          RIGHT"
1020 PRINT"FORWARD    BACKWARD"
1030 PRINT"OPEN          CLOSE"
1040 PRINT"WRIST   -   Q"
1050 PRINT"TWIST   -   Y"
1060 PRINT"POINT  END TEACH"
1080 PRINT:PRINT NP;"POINTS"
1090 FOR I=0 TO 5:PRINT HE(I): NEXT
1100 GET A$:K=PEEK(KB):IF K=KO OR A$=""THEN 1100
1110 IF A$="E" THEN 100
1120 IF A$<>"P"THEN 1150
1130 IF NP=MP THEN GOTO 1000:REM TOO MANY POINTS
1140 NP=NP+1
1145 FOR I=0 TO 5:PT(NP,I)=HE(I):NEXT:GOTO 1000
1150 FOR I=1 TO LEN(C$)
1155 IF A$<>MID$(C$,I,1) THEN NEXT: GOTO 1000
1160 J=I: I=LEN(C$): NEXT:REM CLOSE LOOP NEATLY
1170 CH=INT((J-1)/2): DI=2*(J AND 1)-1: V=HE(CH)
1180 V=V+DI:GOSUB 9000:            REM TAKE A STEP
1190 IF PEEK(KB)=K THEN 1180:  REM KEEP STEPPING
1200 HE(CH)=V:GOTO 1000
2000 NP=0:GOTO 100
5000 IF NP=0 THEN GOTO 100:         REM NO POINTS
5010 FOR P=1 TO NP
5020 FOR I=0 TO 5: TA(I)=PT(P,I): NEXT
5030 GOSUB 8000
5040 NEXT P
5050 GOTO 100
5100 IF NP=0 THEN GOTO 100
5110 FOR P=1 TO NP:PRINT P;
5120 FOR I=0 TO 5
5130 TA(I)=PT(P,I): NEXT
5140 GOSUB 8000
5150 NEXT P
5160 GET A$:IF A$="" THEN 5110
5170 GOTO 100
6000 GOTO 100:REM PUT 'SAVE' HERE
7000 GOTO 100:REM PUT 'INPUT' HERE
8000 REM MOVE FROM HERE TO TARGET
8010 M=0: FOR CH=0 TO 5
8020 RA(CH)=ABS(TA(CH)-HE(CH))
8030 WA(CH)=SGN(TA(CH)-HE(CH))
8040 IF RA(CH)>RM THEN RM=RA(CH)
8050 NEXT: IF RM=0 THEN RETURN:         REM NO MOVE
```

```
8060 FOR CH=0 TO 5
8070 RA(CH)=RA(CH)/RM: NEXT
8100 FOR R=1 TO RM
8105 FOR CH=0 TO 5:I=RA(CH):IF I=0 THEN 8130
8110 I=I+RE(CH):IF I<1 THEN RE(CH)=I:NEXT:RETURN
8120 RE(CH)=I-1: V=HE(CH)+WA(CH): HE(CH)=V
8125 GOSUB 9000
8130 NEXT: NEXT: RETURN
9000 CO=DR(V AND 7)+CH(CH)
9010 POKE PO,CO:POKE PO,CO AND MA:POKE PO,CO
9020 RETURN
10000 DD=56579:PO=56577:KB=197:KO=64: REM CBM 64
10010 DIM DR(7): REM DRIVE VALUES WITH STROBE HI
10020 FOR I=0 TO 7: READ J: DR(I)=16*J+1: NEXT
10030 DATA 1,3,2,6,4,12,8,9: REM N-E-S-W
10040 MA=254: POKE DD,255:    REM SET TO OUTPUTS
10100 DIM CH(5),HE(5): FOR I=0 TO 5: READ J
10110 CH(I)=2*J: NEXT:            REM CHANNEL CODE
10120 DATA 1,3,5,4,2,6
10130 C$="UDLRFBOCQWTY":          REM COMMAND KEYS
10200 DIM TA(5),RA(5),WA(5),RE(5)
10210 FOR CH=0 TO 5: HE(CH)=0: RE(CH)=.5
10220 V=0: GOSUB9000: NEXT:   REM ZERO THE MOTORS
10300 NP=0:MP=20:DIM PT(5,MP)
10900 GOTO 100
```

Simple Driver and Channel Identifier

NB. Change line 10000 if using a PET.

```
10 GOTO 10000
100 INPUT"CHANNEL NUMBER,DISTANCE";CH,DI
110 FOR V=0 TO DI STEP SGN(DI)
120 GOSUB 9000:NEXT
130 GOTO 100
9000 CO=DR(V AND 7)+CH*2
9010 POKE PO,CO:POKE PO,CO AND MA:POKE PO,CO
9020 RETURN
10000 DD=56579:PO=56577:KB=197:KO=64: REM CBM 64
10010 DIM DR(7): REM DRIVE VALUES WITH STROBE HI
10020 FOR I=0 TO 7: READ J: DR(I)=16*J+1: NEXT
10030 DATA 1,3,2,6,4,12,8,9: REM N-E-S-W
10040 MA=254: POKE DD,255:    REM SET TO OUTPUTS
10900 GOTO 100
```