CHAPTER

9

TRIO BASIC PROGRAMMING EXAMPLES

Example Programs

1 - Fetching an Integer Value from the Membrane Keypad

The subroutine “getnum” fetches an integer value from the membrane keypad in variable “num”. The routine prints the number on the display bottom line at cursor position 70, although this can be set to other values. Only the number keys, the “CLR” key and the ENTER key are used. Other keys are ignored.

```
' Demonstrate integer number entry via Membrane Keypad:
getnum: pos=70
  num=0
  PRINT#4,CHR(20);
  REPEAT
    PRINT#4,CURSOR(pos);num[6,0];
    GET#4,k
    IF k=69 THEN GOTO getnum
    IF k>=59 AND k<=61 THEN k=k-7
    IF k>=66 AND k<=68 THEN k=k-17
    IF k=71 THEN k=48
    IF k>47 AND k<58 THEN
      k=k-48
      num=num*10+k
    ENDIF
  UNTIL k=73
RETURN
```

Example 2 - Fetching a Real Value from the Membrane Keypad

This similar routine also fetches a number from the membrane keypad, but this number can have up to 2 decimal places. Note how this example uses the emulated keypad from *Motion* Perfect.

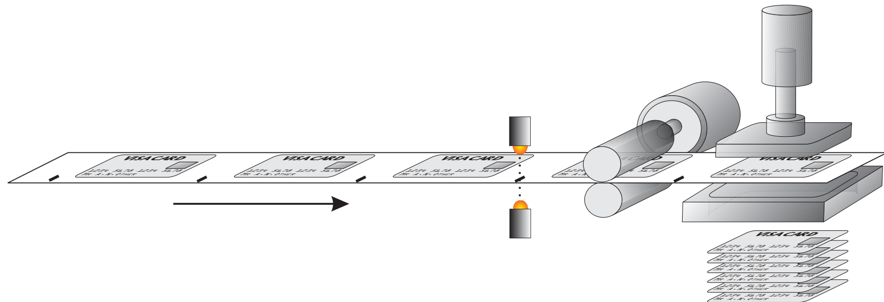
```
getnum:
  pos=40
  dpoint=0
  num=0
  negative=1
  PRINT#5,CHR(20);
  REPEAT
    PRINT#5,CURSOR(pos);num*negative[8,2];
```

```
        GET#5,k
        IF k=72 AND dpoint=0 THEN dpoint=1
        IF k=70 THEN negative=-negative
        IF k=69 THEN GOTO getnum
        IF k>=59 AND k<=61 THEN k=k-7
        IF k>=66 AND k<=68 THEN k=k-17
        IF k=71 THEN k=48
        IF k>47 AND k<58 THEN
          k=k-48
          IF dpoint>0 THEN
            dpoint=dpoint/10
            IF dpoint>=0.01 THEN num=num+k*dpoint
          ELSE
            num=num*10+k
          ENDIF
        ENDIF
      UNTIL k=73
      num=num*negative
    RETURN
```

Example 3 - ATM Card Production

Key Features Used: REGIST, MOVEMODIFY

An automated die-cutting machine, is designed to punch out pre-printed plastic cards for use in ATM machines etc.



There is one servo axis which is connected to the draw rollers which feed the card into the machine. A printed registration mark appears once per card and is sensed by an optical sensor connected to the Registration input of the MC2xx's Servo Daughter Board.

The operation of the machine is quite simple, the cards are printed at a known fixed-pitch. Each cycle, the draw rolls must feed the card into position, an output is then fired to operate the punch. An input signals that the punch is clear of the cards and the cycle can repeat.

In an ideal situation we would simply datum the first card and then move a fixed pitch every cycle,

```
loop:
  MOVE(card_pitch)
  WAIT IDLE
  OP(punch,ON)
  WAIT UNTIL IN(punch_clear)=OFF
  WAIT UNTIL IN(punch_clear)=ON
  OP(punch,OFF)
GOTO loop
```

In the real world we must allow for mechanical slippage and any inconsistencies which may occur in the printing. Therefore we will use the registration mark to synchronise the position of the draw each cycle

```
loop:
  DEFPOS(0)
  REGIST(3)
  MOVE(card_pitch)
  WAIT UNTIL MARK
  MOVEMODIFY(REG_POS+20)
  WAIT IDLE
  OP(punch,ON)
  WAIT UNTIL IN(punch_clear)=OFF
  WAIT UNTIL IN(punch_clear)=ON
  OP(punch,OFF)
GOTO loop
```

The above example shows only the simplest form of the main loop. It allows for a fixed offset value of 20, but there is no provision for error handling etc. An example where the code might be expanded to check for registration errors would be:

```
loop:
  DEFPOS(0)
  REGIST(3)
  MOVE(card_pitch)
  WAIT UNTIL MARK OR MTYPE=0
  IF MTYPE=0 THEN
    ' Indicate error to user
    PRINT #3,"Registration Error!"
    errors=errors+1
    if errors>max_errors then GOTO reg_failed
    OP(error_lamp,ON)
  ELSE
    OP(error_lamp,OFF)
    MOVEMODIFY(reg_pos+20)
    WAIT IDLE
  ENDIF
  ' Rest of loop as before
GOTO loop

reg_failed:
  PRINT #3,CURSOR(00);"Too many reg errors!"
  PRINT #3,CURSOR(20);"Press any key...    "
  GET #3,k
GOTO start
```

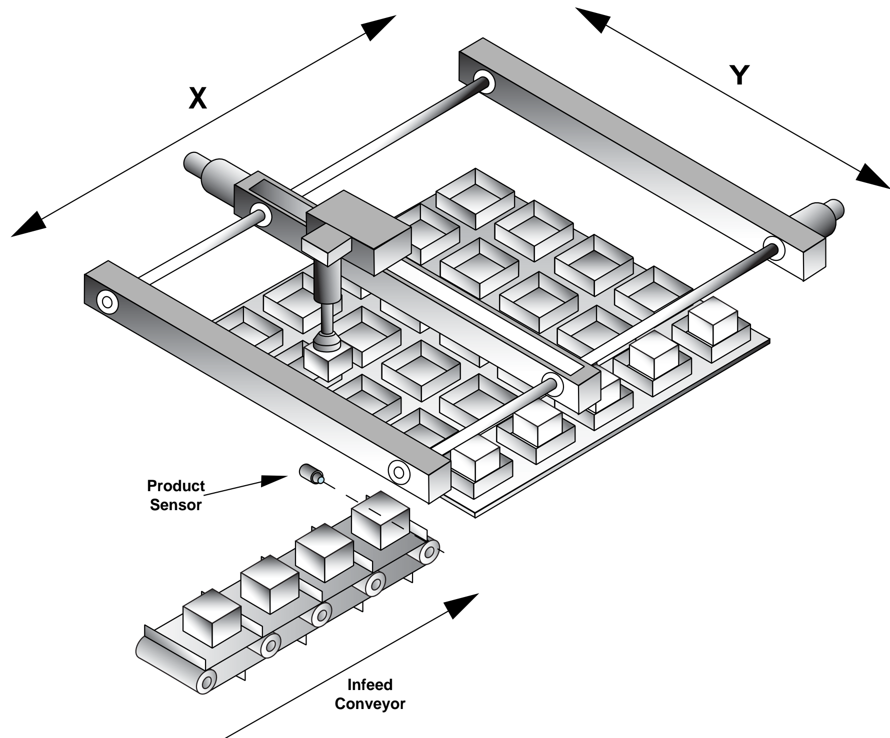
Example 4 - 2 Axis Pick & Place System

Overview

A square palette has sides 1200mm long.

It must be divided into a grid, each of these positions on the palette contains a box into which a widget must be placed:

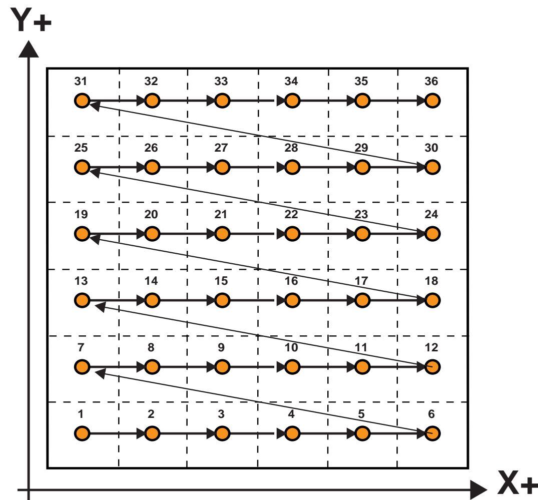
A vacuum operated pick-up mechanism collects objects from a conveyor and fills each of the boxes in turn.



Additional Information:

Whist the palette size is fixed at the maximum size of 1.2m square, the program should be flexible enough to allow for a user-defined number of boxes on the palette. The grid could contain up to 10 divisions in each direction and they may be combined in any ratio, i.e. 2x4, 6x3, 1x10, 9x4 etc.

The illustration below shows a sample palette with a 6x6 grid of boxes. The numbers/arrows show the order in which the boxes are filled. Note that we step through the rows (Y axis) in turn, filling each box (move along X axis) before moving onto the next row.



Structuring the program

This example can be solved with a very simple structure using two nested `FOR..NEXT` loops.

Firstly we create a loop to step through each row (Y) in turn:

```
FOR y=0 to ydiv-1

NEXT y
```

'ydiv' is the number of rows, and the -1 is because we start counting at 0 rather than 1.

Now, within this loop we create another for the 'X' direction:

```
FOR y=0 to ydiv-1
  FOR x=0 to xdiv-1
    '
  NEXT x
NEXT y
```

Calculating the box positions

So now we have a sequence which steps sequentially through each row, and then through each position on that row in turn. We can use the absolute move (**MOVEABS**) command to position the axes at an absolute position in our X/Y coordinate system in the form

```
MOVEABS(x,y)
```

The x and y variables with the **FOR..NEXT** loop are simply logical box coordinates and therefore need to be scaled to the correct positions.

If we know the palette size (1200) and the number of divisions in each direction (xdiv/ydiv) then we can simply calculate an appropriate scaling, thus for the x axis:

```
xscale = 1200/xdiv
```

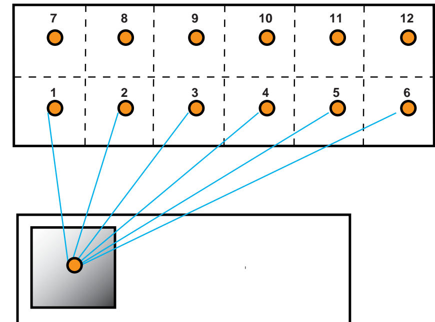
The actual position (of the box's corner) would therefore be (x*xscale), we could adjust this for the centre of the box by adding half the box size (xscale/2). So the position would be:

```
(x*xscale)+(xscale/2)
```

Our final consideration is that for each box we first have to move to the preset pick-up position, fetch the product and then move to the appropriate empty box to place the product in.

The pick up point is at a known absolute position and so we can simply use a pair of constants (pick_x and pick_y) to reference this point.

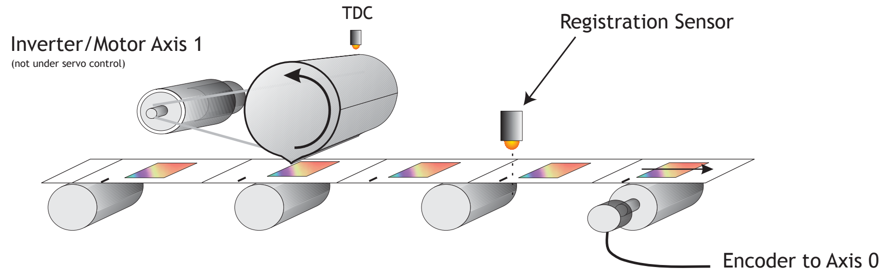
```
MOVEABS(pick_x, pick_y)
```



```
constants:
  nozzle=8 `    output - nozzle raise/lower
  vacuum=9 `    output - vacuum on / off

  xdiv=6
  ydiv=6

start:
  xscale=1200/xdiv
  xmid=xscale/2
  yscale=1200/ydiv
  ymid=yscale/2

  FOR y=0 TO xdiv-1
    FOR y=0 TO ydiv-1
    GOSUB pick
      MOVEABS((x*xscale)+xmid,(y*yscale)+ymid)
      WAIT IDLE
      GOSUB place
    NEXT x
  NEXT y
GOTO start

pick:
  MOVEABS(pick_x, pick_y)
  WAIT IDLE

  OP(nozzle,ON)
  OP(vacuum,ON)
  wa(500)
  OP(nozzle,OFF)
  wa(500)
RETURN

place:
  OP(nozzle,ON)
  OP(vacuum,OFF)
  wa(250)
  OP(nozzle,OFF)
  wa(500)
RETURN
```

Example 5 - Rotating Print Head with Registration

Description A rotating print head prints a number on a conveyor mounted product. During printing the print head must be synchronized with the conveyor. The print position must be registered to be relative to a registration mark.



The program achieves the motion profile by:

1) Making a synchronization gearbox connection between the conveyor and the print head with CONNECT. This will let the printer make a print on the conveyor. The distance between the prints will be the peripheral distance of the print head rotation.
2) Datuming and setting the repeat distance REPDIST for the print head rotation so that it has an absolute position of zero every time it touches the conveyor.
3) Superimposing a movement onto the print head. This movement has two functions: To adjust the printer position to keep the system in register and to adjust by the difference between the peripheral distance of the printer and the registration marks on the conveyor.
4) The superimposed movement is run on axis 3 of the controller (an “imaginary” axis) and the movement summed with the ADDAX command. The move is performed when the printer is going “over the top” of its stroke.

Program Listing

```
'  Rotating Head with Registration:
start: GOSUB initial

loop:
  WAIT UNTIL MPOS>100' Wait 100mm past print

  IF MARK THEN
    ' mark seen during last cycle:
```

```
        ’ Limit adjust to 10mm
        r_adj=REG_POS*0.5' Apply 50% of error
        IF ABS(r_adj)>10 THEN r_adj=SGN(r_adj)*10
        OP(8,OFF)
      ELSE
        ’ mark not seen last cycle: Set Zero adjust
        r_adj=0
        OP(8,ON)' light "no register" warning lamp
      ENDIF

      BASE(3)' Correction on axis 3
      MOVELINK(75-r_adj,150,25,25,1)
      ‘ 75 is the diff. between the mark spacing and
      ‘ the print head circumference
      ' Move linked to conveyor
      WAIT IDLE
      BASE(0)
      REGIST(3)
    GOTO loop

    initial:
      BASE(0)'   Setup axis 0
      UNITS=20'  Edges/mm
      P_GAIN=0.5
      REP_DIST=200' 200mm=180 degrees
      SERVO=ON

      BASE(1)'  Setup axis 1
      SERVO=OFF
      UNITS=15'Edges/mm on conveyor
      BASE(3)' Setup axis 3
      UNITS=20' Match axis 0 units

      ' Datum axis 0 keeping sync with paper:
      WDOG=ON
      BASE(0)
      CONNECT(20/15,1)
      WAIT UNTIL IN(0)=ON' Wait for prox
      DEFPOS(-150)
      ADDAX(3)'  Add moves on axis 3
    RETURN
```

Example 6 - *Motion Coordinator* programs sharing data

Description These two programs run multi-tasking on a *Motion Coordinator*. The Motion Cycle program performs a movement. The Operator Interface program communicates with a membrane keypad to control the Motion Cycle program. In this simple example of multi-tasking the two tasks communicate via two global variables.

`VR(start)`	This holds the start/stop signal
`VR(length)`	This holds the movement length

Operator Interface Program

```
' Motion Coordinator Demonstration Program

start: GOSUB initial

loop:
  PRINT #3,CHR(20);CHR(14);
  PRINT #3,CURSOR(0);">LENGTH:";in_length[0];
  IF VR(start)=ON THEN
    PRINT #3,CURSOR(15);"STOP<";
  ELSE
    PRINT #3,CURSOR(15);" RUN<";
  ENDIF

  GET #3,kp
  PRINT kp
  IF kp=53 THEN GOSUB input_length
  IF kp=54 THEN VR(start)=1-VR(start)
GOTO loop

input_length:
  PRINT #3,CURSOR(60);"New Length:";
  GOSUB getnum
  IF num>=smallest_len THEN
    in_length=num
  ELSE
    PRINT #3,CURSOR(60);"Min. Length=200mm";
    WA(1000)
  ENDIF
RETURN
getnum:
```

```
 pos=40
  num=0
  PRINT#4,CHR(20);
  REPEAT
    PRINT#4,CURSOR(pos);num[6,0];
    GET#4,k
    IF k=69 THEN GOTO getnum
      IF k>=59 AND k<=61 THEN k=k-7
      IF k>=66 AND k<=68 THEN k=k-17
      IF k=71 THEN k=48
      IF k>47 AND k<58 THEN
        k=k-48
        num=num*10+k
      ENDIF
  UNTIL k=73
RETURN

initial:
  PRINT #3,CHR(20);CHR(14);
  PRINT #3,CURSOR(0);
  PRINT #3," Demonstration of  "
  PRINT #3,"Motion Coordinator "
  WA(3000)

' Set Global Variable Pointers:
  start=0
  length=1

' Set any none zero local variables:
  in_length=VR(length)
  VR(start)=0
  RUN "cycle"
RETURN
```

Motion Cycle Program

```
'
' Motion Cycle demonstration program:
'
  GOSUB setvar
  GOSUB initial
```

```
loop:
  WAIT UNTIL VR(start)=ON
  MOVE(VR(length))
  MOVEABS(0)
GOTO loop

initial:
  WDOG=ON
  WA(100)

  BASE(0)
  P_GAIN=0.8
  FE_LIMIT=1000
  SERVO=ON
  ACCEL=1000000
  DECEL=100000
  SPEED=10000
RETURN

setvar:
  ' Define Global Variable Pointers:
  start=0
  length=1
RETURN
```

Example 7 - Handling Axis Errors

The *Motion Coordinator* controllers are designed to trap error conditions in hardware, and if required to automatically open the drive enable relay (watchdog) and to disable the output to the drives.

As this mechanism happens automatically, it may not be immediately apparent that an error has occurred and therefore we need a mechanism in the software to recognise it, and to set up the type of errors which will cause the controller to disable the drive / output.

The relevant parameters are:

- **AXISSTATUS**
- **ERRORMASK**
- **MOTION_ERROR**
- **ERROR_AXIS**

```
start:
   ' Monitor constantly until axis error occurs
   ' Set ERRORMASK so that Following Errors and Fwd/Rev limit
   ' switches will automatically trip the watchdog relay
  ERRORMASK = 256+16+32

REPEAT
  IF MOTION_ERROR<>0 THEN
    ax = ERROR_AXIS
    BASE(ax)
    PRINT #3,CURSOR(0);"Error on Axis ";ax[0]
    IF (AXISSTATUS AND 256)>0 THEN PRINT #3,"Fol. Error";
    IF (AXISSTATUS AND (16+32))>0 THEN PRINT #3,"H/W Limit";
    IF (AXISSTATUS AND 512+1024)>0 THEN PRINT #3,"S/W Limit";
  ENDIF

  WAIT UNTIL KEY#3
  GET #3,k
GOTO start
```