XY INTERPOLATION ALGORITHMS

Kenneth and Melvin Goldberg
3913 Pine Street, Apt. F
Philadelphia, Pennsylvania 19104

A microcomputer can be used to control the motion of numerical control machines. This article describes a straightforward method for approximating diagonal lines and circular motion on an XY plane.

Many numerical control machines are powered by stepping motors. When a pulse is sent to a stepping motor, the stepping motor alters its position by a unit step. Two motors can be used to control the XY movements of an arm or tool over a working plane.

If the pulses are generated by a device which can remember or generate a specified train of pulses, repetitive operations such as grinding, painting, or cutting can be performed hundreds of times with virtually no variation. A microcomputer is an obvious choice to generate and remember the pulses.

Since stepper motors can move only in discrete steps, we must approximate the actual curve by a series of small XY motions. Many algorithms rely upon parametric functions such as sine and cosine to perform the necessary calculations. Parametric functions, however, typically require a high degree of numeric precision. Calculating sine and cosine values with a microcomputer can be too time-consuming to be useful in a real-time application.

The following two algorithms require no parametric functions. This makes them ideally suited to the computation and memory capacities of microcomputers. Since these algorithms do not require a large amount of complex mathematical calculation, they are fast enough to be used in real-time applications. The program shown in listing 1 is written in C for fast execution, portability, and ease of modification. The program, as shown, does not actually control any stepping motors; rather, it provides a screen display consisting of +1, −1, and 0. In an actual control

Listing 1.

```
/*********************************************************/
/*              PROGRAM STEER                            */
/*                                  (C)M.M. and K.Y. GOLDBERG */
/*      This Numerical Control Program uses linear       */
/*      interpolation to provide a flow of output        */
/*      pulses which can be used to steer an xy table    */
/*********************************************************/
#include <bdscio.h>     /* micro 'C' function package */

char temp1[5],temp2[5],temp3[5],temp4[5],temp5[5];
char eia[10];
int feedrate,drag,oil;  /* vars dealing with feedrate and delay func. */
int x1,y1;      /* starting point */
int x2,y2;      /* relative position */
int x3,y3;      /* endpoint */
int xo,yo;      /* direction of output: +1, -1,  or 0 */
int dx,dy;      /* differentials of x and y */
int stepnum;
int fxy;        /* value of function */

int rad,radrad,f,a,b,d;
main(argc,argv)
        int argc;
        char *argv[];
{
        printf("  STEER:  A NUMERICAL CONTROL PROGRAM\n");
        printf("\nInput Command line (ie, 'GO1 (0,0,10,20) 100')\n: ");
        scanf("%s (%s,%s,%s,%s) %s",eia,temp1,temp2,temp3,temp4,temp5);
        feedrate = atoi(temp5);
        x1 = atoi(temp1);
        y1 = atoi(temp2);
        x3 = atoi(temp3);
        y3 = atoi(temp4);

        if(!strcmp(eia,"GO1"))doline();
                else docircle();

}

doline()        /* interpolates impulses for a straight line */
{
        printf("\n\nFor EIA code '%s' with feedrate = %d",eia,feedrate);
        printf("\nGoing from (%d,%d) to (%d,%d):\n",x1,y1,x3,y3);

        stepnum = x2 = y2 = fxy = 0;
        drag = 100; oil = 1;
        setdirection();
        printf("\nStep\tFXY\tX2\tY2\t\tXO\tYO");
        while((x2 != dx) || (y2 != dy)) /* at endpoint? */
                {
                delay();
                printf("\n%d\t%d\t%d\t%d\t\t",stepnum++,fxy,x2,y2);
                if(fxy > 0)
```

```
54:                             { printf("%d",xo);   ++x2; fxy = fxy - dy; }
55:                     else
56:                             { printf("\t%d",yo); ++y2; fxy = fxy + dx; }
57:
58:                     }
59: }
60:
61: setdirection()  /* sets output directions and initial fxy value  for line */
62: {
63:         dy = y3 - y1;
64:         if(dy < 0)   yo = -1;
65:                     else yo = 1;
66:         dy = abs(dy);
67:
68:         dx = x3 - x1;
69:         if(dx < 0)   xo = -1;
70:                     else xo = 1;
71:         dx = abs(dx);
72:
73:         fxy = dx - dy;
74: }
75:
76: docircle()      /* Circle Routine */
77: {
78:         stepnum = 0;
79:         x2 = x1; y2 = y1;
80:         d = x3;  rad = y3;
81:         radrad = rad * rad;
82:
83:         printf("\n\nFor EIA code '%s' with feedrate = %d",eia,feedrate);
84:         printf("\nCircling from (%d,%d) in direction %d with radius %d:",
85:                 x1,y1,d,rad);
86:
87:         printf("\nStep\tx2\ty2\tradius\tfxy\tdx\tdy\tf  a  b  d  xo  yo\n");
88:         do
89:                 {
90:                 delay();
91:                 fxy = (x2*x2) + (y2*y2) - (radrad);
92:                 dx = 2*x2;
93:                 dy = 2*y2;
94:                 f = (fxy < 0) ? 0 : 1;
95:                 a = ( dx < 0) ? 0 : 1;
96:                 b = ( dy < 0) ? 0 : 1;
97:
98:                 getdir();
99:
100:        printf("\n%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d  %d  %d  %d   %d   %d",
101:                stepnum++,x2,y2,rad,fxy,dx,dy,f,a,b,d,xo,yo);
102:
103:                x2 = x2 + xo;
104:                y2 = y2 + yo;
105:                }
106:        while((x2 != x1) || (y2 != y1));
107:
108: }
109:
110:
111: delay() /* delay loop : feedrate approx = # steps/minute */
112: {
113:         int i;i = 0;
114:         while (++i != ((feedrate + drag) / 30))continue;
115:         if(drag > 0)    /* drag increases the delay at the beginning */
116:                         /* to allow for inertia in machine startup */
117:                 {
118:                 drag = drag - (oil * oil);      /* falls off exponentially */
119:                 --oil;if (drag < 0) drag = 0;
120:                 }
121: }
122:
123:
124: getdir()        /* creates 'mock' binary representation of d,f,a,b */
125:                 /* and uses this to determine best output */
126: {
127:         int binrep;
128:         binrep = 0;
129:         xo = yo = 0;
130:         if(d)binrep = binrep + 8;
131:         if(f)binrep = binrep + 4;
132:         if(a)binrep = binrep + 2;
133:         if(b)binrep = binrep + 1;
134:
135:         switch(binrep)
136:                 {
137:                 case 0:         yo = -1 ;break;
138:                 case 1:         xo = -1 ;break;
139:                 case 2:         xo = 1  ;break;
140:                 case 3:         yo = 1  ;break;
141:                 case 4:         xo = 1  ;break;
142:                 case 5:         yo = -1 ;break;
143:                 case 6:         yo = 1  ;break;
144:                 case 7:         xo = -1 ;break;
145:                 case 8:         xo = -1 ;break;
146:                 case 9:         yo = 1  ;break;
147:                 case 10:        yo = -1 ;break;
148:                 case 11:        xo = 1  ;break;
149:                 case 12:        yo = 1  ;break;
150:                 case 13:        xo = 1  ;break;
151:                 case 14:        xo = -1 ;break;
152:                 case 15:        yo = -1 ;break;
153:                 }
154: }
```

situation, these values would be transferred to stepper motors or a graphic display.

Linear Interpolation. Approximating diagonal lines with unit steps in two dimensions can be accomplished with the following algorithm.

1. Define the starting position (X1,Y1) and ending position (X3,Y3). Define the feed rate (f). Feed rate is the speed at which the tool being controlled moves. The Electronic Industries Association (EIA) recommends the following notation for linear interpolation:

G01 (X1, Y1, X3, Y3) f

where

X1,Y1 is starting position
X3,Y3 is ending position
f is feed rate

2. Initialize variables. Set the current relative position (X2,Y2) of the tool to (0,0). This effectively sets the current tool position to the starting point. Set the step count number to zero.

3. Calculate the direction in which to move the tool. When following a straight line from one point to another, all X motion is in the same direction, as is all Y motion. The direction is determined by the sign of the difference (DX and DY) between the ending and the starting positions. DX = X3 − Y1. DY = Y3 − Y1.

4. Calculate the difference between the absolute values of DX and DY. This determines FXY, a variable which is used to control the movements along the X and Y axes. FXY = |DX| − |DY|.

5. Generate output pulses to move the tool until the endpoint is reached. This is the heart of the program. The proportionate stream of XY pulses is generated by manipulating variable FXY. Each time a

```
B>steer
  STEER:  A NUMERICAL CONTROL PROGRAM

Input Command line (ie, 'G01 (0,0,10,20) 100')
: G01 (0,0,3,-7) 300


For EIA code 'G01' with feedrate = 300
Going from (0,0) to (3,-7):

Step    FXY     X2      Y2              XO      YO
0       -4      0       0                       -1
1       -1      0       1                       -1
2       2       0       2               1
3       -5      1       2                       -1
4       -2      1       3                       -1
5       1       1       4               1
6       -6      2       4                       -1
7       -3      2       5                       -1
8       0       2       6                       -1
9       3       2       7               1
```

Table 1. A sample program run which draws a line between points (0,0) and (3, −7). The tool moves with a feed rate of 300.

D	FXY	DX	DY	XOUT	YOUT
0	0	0	0	0	−1
0	0	0	1	−1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	−1
0	1	1	0	0	1
0	1	1	1	−1	0
1	0	0	0	−1	0
1	0	0	1	0	1
1	0	1	0	0	−1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	−1	0
1	1	1	1	0	−1

Table 2. The 16 possible arrangements of values which are generated by the circular interpolation algorithm.

```
B>steer
  STEER:  A NUMERICAL CONTROL PROGRAM

Input Command line (ie, 'G01 (0,0,10,20) 100')
: G03 (25,0,0,25) 100


For EIA code 'G03' with feedrate = 100
Circling from (25,0) in direction 0 with radius 25:
Step    x2      y2      radius  fxy     dx      dy      f  a  b  d  xo  yo
0       25      0       25      0       50      0       1  1  1  0  -1   0
1       24      0       25      -49     48      0       0  1  1  0   0   1
2       24      1       25      -48     48      2       0  1  1  0   0   1
3       24      2       25      -45     48      4       0  1  1  0   0   1
4       24      3       25      -40     48      6       0  1  1  0   0   1
5       24      4       25      -33     48      8       0  1  1  0   0   1
6       24      5       25      -24     48      10      0  1  1  0   0   1
7       24      6       25      -13     48      12      0  1  1  0   0   1
8       24      7       25      0       48      14      1  1  1  0  -1   0
9       23      7       25      -47     46      14      0  1  1  0   0   1
10      23      8       25      -32     46      16      0  1  1  0   0   1
11      23      9       25      -15     46      18      0  1  1  0   0   1
12      23      10      25      4       46      20      1  1  1  0  -1   0
13      22      10      25      -41     44      20      0  1  1  0   0   1
14      22      11      25      -20     44      22      0  1  1  0   0   1
15      22      12      25      3       44      24      1  1  1  0  -1   0
16      21      12      25      -40     42      24      0  1  1  0   0   1
17      21      13      25      -15     42      26      0  1  1  0   0   1
18      21      14      25      12      42      28      1  1  1  0  -1   0
19      20      14      25      -29     40      28      0  1  1  0   0   1
20      20      15      25      0       40      30      1  1  1  0  -1   0
21      19      15      25      -39     38      30      0  1  1  0   0   1
22      19      16      25      -8      38      32      0  1  1  0   0   1
23      19      17      25      25      38      34      1  1  1  0  -1   0
24      18      17      25      -12     36      34      0  1  1  0   0   1
25      18      18      25      23      36      36      1  1  1  0  -1   0
```

Table 3. Output generated by the circular interpolation algorithm for a circular curve.

step is taken in the X direction, the absolute value of DY is subtracted from FXY. When FXY becomes negative, a step is taken in the Y direction, and the absolute value of DX is added to FXY. The sign of FXY determines the appropriate step needed to approximate a straight line.

6. A delay loop controls the feed rate. This loop may include extra delay for the initial steps. "Ramping up" the feed rate in this manner is useful in real-world situations where the inertia of a machine may have a significant effect on the system.

Table 1 shows the output generated when a starting point of (0,0), an ending point of (3, −7), and a feed rate of 300 are given to the program shown in listing 1.

Circular Interpolation. A conceptually similar nonparametric algorithm can provide the necessary XY steps for approximating a circular path. The equation for a circle is:

$FXY = X^2 + Y^2 - R^2$

FXY = positive when (X,Y) is outside circle
0 when (X,Y) is on circumference
negative when (X,Y) is inside circle

DX = 2X

DY = 2Y

The variable FXY determines the direction in which the tool is moved at each point on the circle. The motion is always perpendicular to the instantaneous circular radius. The tangent to a circle is always perpendicular to the radius. The X and Y components of the radius are defined by the partial derivatives of FXY.

We propose to step the machine tool around the circle by comparing the current tool position to the ideal radius. We perform this comparison by tracking the value of FXY. We know that the tool has crossed the circumference and must be corrected when the sign of FXY changes. The appropriate correction (±X, ±Y) depends on the quadrant in which the tool is located.

This algorithm's simplicity lies in the fact that the only information required to determine the proper output is the sign of FXY, its derivatives, and the direction of rotation. (0 = clockwise, 1 = counterclockwise). If we denote positive by 1 and negative by 0, then we can organize the 16 possible combinations of values as shown in table 2. Table 3 shows the output generated for a typical circular approximation.

Figure 1 demonstrates the raggedness found in part of an enlarged path of a typical circle. The raggedness is greatly decreased in large-diameter circles. With a radius of 1000 steps, a circle will appear smooth to the naked eye.

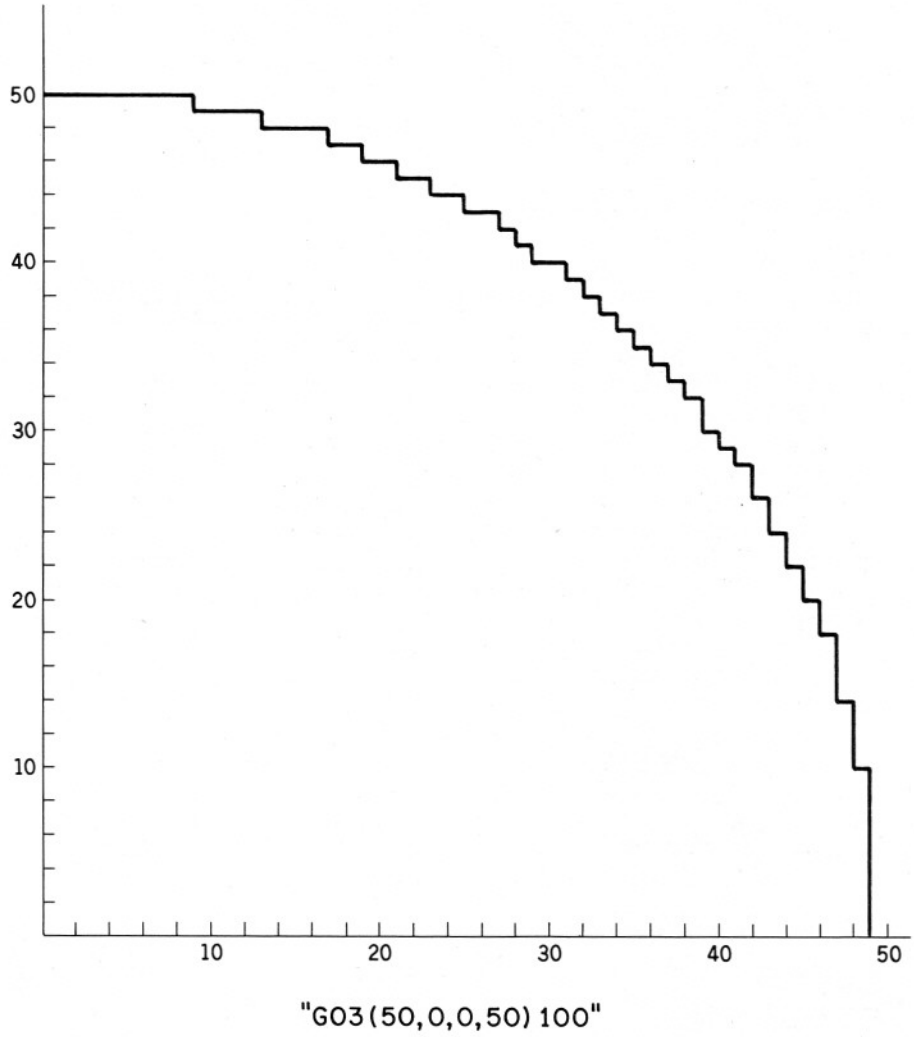

Figure 1. A portion of a circular path generated by only xy movements.

Summary. These two examples demonstrate that regular figures can be approximated by simple, nontrigonometric algorithms. The algorithms presented here can be extended to other forms such as an ellipse. The examples also show that the best approach to a real-world problem may well be an approximate arithmetic solution, rather than a mathematically precise solution.

Acknowledgements

The authors are indebted to Frank Francisco for program assistance, Alan Krigman of ICON Information for providing hardware facilities, and to Eric Gray for supplying a typewriter.