Interpolated Control VC++ Make Circles

Introduction

User Requirements

To run the Make Circles Program, the user must be familiar with QuickControl® Software and advanced SilverLode™ operation. If the user is creating a new program that uses this application example as a template, it is assumed the reader is familiar with the SilverLode™ Command Set, QuickControl Software and Windows C/C++ programming.

Notice: Microsoft and Windows are registered trademarks of Microsoft Corporation. SilverLode and QuickControl are trademarks of QuickSilver Controls Corporation.

Application Setup

The Make Circles Application communicates with two servos in an XY configuration via a Windows PC COM port. The application commands both servos to drive the X-Y table in circles defined from user specified parameters. Other shapes can be defined by reducing the number of segments in the arc definition setting.

The following items are required before using the driven X-Y table with Make Circles:

1) Each servo must be initialized with a unique Device ID.
2) Both servos & the PC's COM port are configured for: 57600 Baud, No parity, 8 data bits, 2 stop bits.
3) Both servos are initialized using QuickControl with 57600 Baud, 8 Bit ASCII Protocol
4) I/O #1 from the X-axis needs to be connected to I/O #1 of the Y-axis.
5) The serial network (RS-485 or RS-232 multi-drop) is communicating correctly from the PC COM port to both servos independently on the network.
6) Each servo must home to the midpoint of the axis (center of travel), X-Y location (0,0). See Application Note QCI-AN001 Homing Techniques for details.
7) Download Leader Queued Interpolation.qcp into the X- axis servo.
8) Download Follower Queued Interpolation.qcp into the Y-axis servo.

Supplied Files

The programming used in Make Circles is compatible with Microsoft® Visual C++ 6.0.

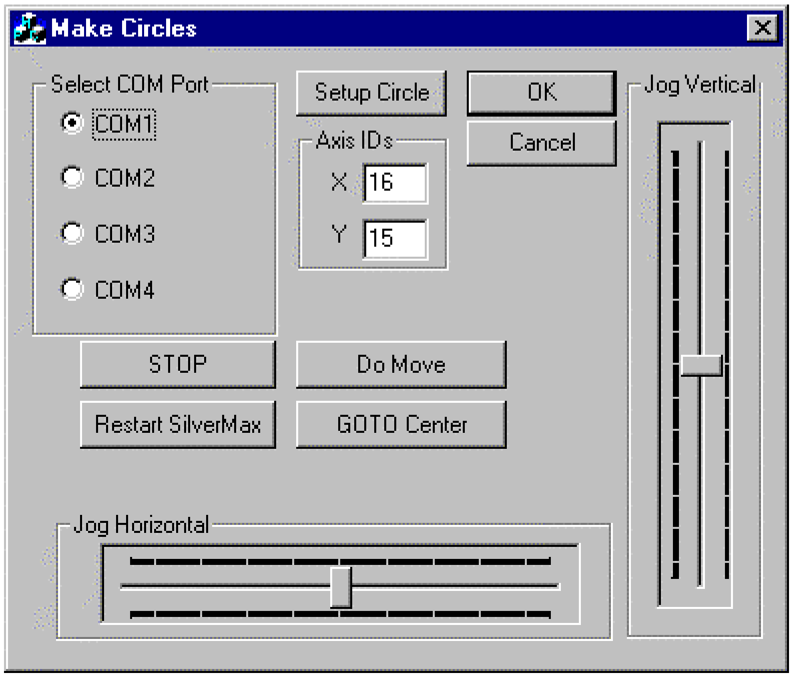
The Make Circles Application requires about 2 MB of hard disk space for all files and includes the following folders.

Directory/Folder	Description
\MakeCircles\	C++ Source Code files for the Make Circles Application
\MakeCircles Support\	Application Support files including: MakeCircles.xls – spreadsheet that calculates circle data Leader Queued Interpolation.qcp – X axis QuickControl program Follower Queued Interpolation.qcp. – Y axis QuickControl program
\Setup	Install the Make Circles Application in Windows.

Make Circles Overview

Make Circles Main Dialog Box

Below is the main application screen of Make Circles. This dialog box is displayed upon execution of the Make Circles application. The box is a control panel to select the Make Circles serial port, control start/stop motion, check independent axis motion, homing the axes and rebooting the servos.



Select COM Port: [✓ COM1] COM2 COM3 COM4

Axis IDs

X 16

Y 15

Functions available in the Make Circles Main Dialog Box

Select COM Port

Select the PC serial COM port to be used for network communication to both servos. The four COM ports listed (1 – 4) use the port settings for IRQ & I/O address defined by Windows.

Required PC/Windows serial port settings: 57600 baud, no parity, 8 data bits, 2 stop bits.

Axis IDs

Enter the servo Device ID numbers for the X Axis and Y Axis into the correct fields. Each servo must be initialized with a unique device ID number. Both servos should be initialized using QuickControl.

STOP

This button sends a Stop (STP) command to both servos causing all motion to stop and all program execution to end. The Leader & Follower programs running will be stopped.

NOTE: This button is also a good way to test serial communications. If the servos red LED Comm indicator flashes when the **STOP** button is pressed, then communication is correct to that motor.

Restart

This button sends a Restart (RST) command to both servos. The motors will stop motion, halt program execution, then restart the initialization and any other program set to auto-run on power up.

Jog Slider Bars

When the Jog Sliders are dragged with the mouse, move commands are sent to the X and Y axes to move the specified jog speed. To move an axis, click on the slider, hold the mouse button down and drag the slider with the mouse. The jog velocity of the motor is proportional to the slide bar position. The maximum velocity of the slider is user specified in the **Setup Circle Dialog Box** (see CSetupCircle).

Setup Circle Button

Pressing this button launches the Setup Circle Dialog Box. (see CSetupCircle). The Setup Circle Dialog Box provides the user with data windows to enter the desired parameters of the circular motion from the X Y table.

Do Move

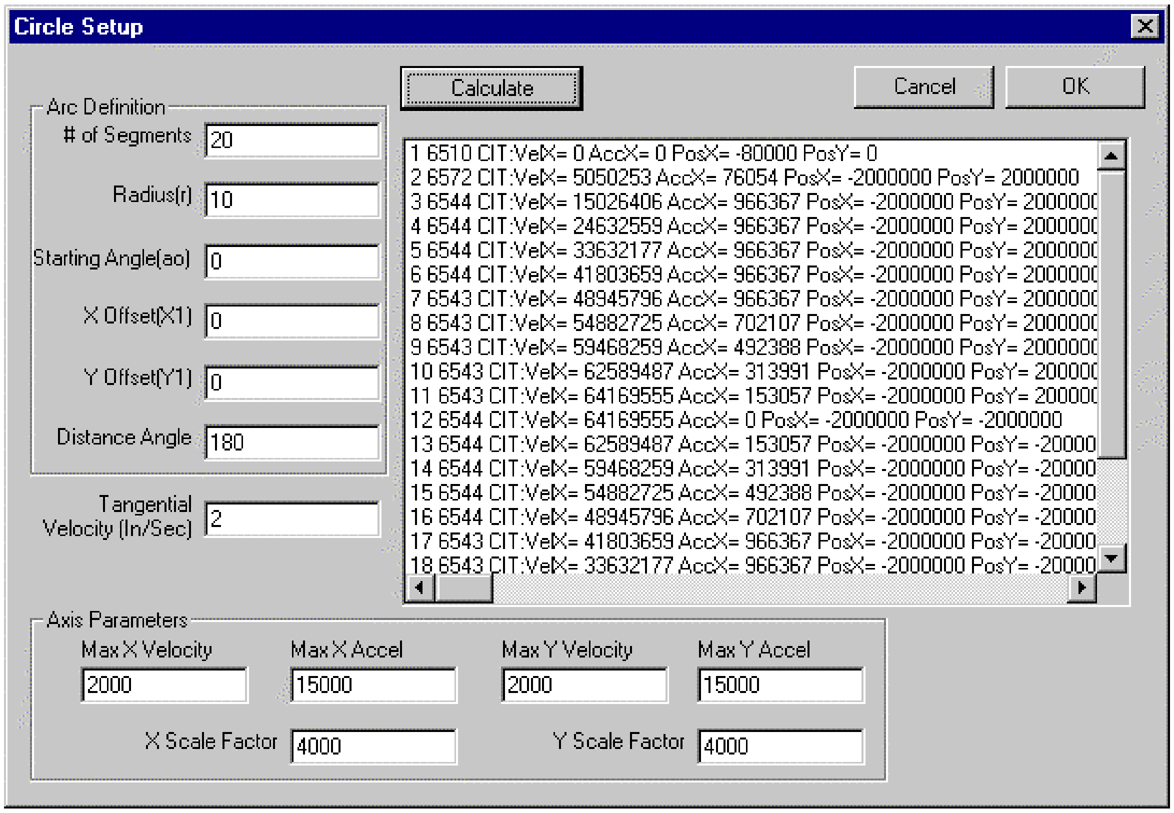
The Do Move button begins the serial transmission of the interpolated motion data from the PC to the servos. This data is the processed for execution with the IMS command. However, the interpolated motion data points must be calculated **FIRST** using the **Setup Circle Dialog Box** before the Do Move button can begin the execution correctly.

GOTO Center

The GOTO Center button will cause both servos to move to the home location on both axes. The motion command sent to the motors will move the axes to the 0 location on that axis. The servo actual position register defines this location. The home location (0,0) is the center location on the X-Y table.

Circle Setup Dialog Box

Below is the Circle Setup Dialog Box after the **Calculate** button is selected. The Make Circles Application calculates the interpolated motion data listed in the window.



Arc Definition

# of Segments	20
Radius(r)	10
Starting Angle(ao)	0
X Offset(X1)	0
Y Offset(Y1)	0
Distance Angle	180
Tangential Velocity (In/Sec)	2

```
1 6510 CIT:VelX= 0 AccX= 0 PosX= -80000 PosY= 0
2 6572 CIT:VelX= 5050253 AccX= 76054 PosX= -2000000 PosY= 2000000
3 6544 CIT:VelX= 15026406 AccX= 966367 PosX= -2000000 PosY= 200000
4 6544 CIT:VelX= 24632559 AccX= 966367 PosX= -2000000 PosY= 200000
5 6544 CIT:VelX= 33632177 AccX= 966367 PosX= -2000000 PosY= 200000
6 6544 CIT:VelX= 41803659 AccX= 966367 PosX= -2000000 PosY= 200000
7 6543 CIT:VelX= 48945796 AccX= 966367 PosX= -2000000 PosY= 200000
8 6543 CIT:VelX= 54882725 AccX= 702107 PosX= -2000000 PosY= 200000
9 6543 CIT:VelX= 59468259 AccX= 492388 PosX= -2000000 PosY= 200000
10 6543 CIT:VelX= 62589487 AccX= 313991 PosX= -2000000 PosY= 20000
11 6543 CIT:VelX= 64169555 AccX= 153057 PosX= -2000000 PosY= 20000
12 6544 CIT:VelX= 64169555 AccX= 0 PosX= -2000000 PosY= -2000000
13 6544 CIT:VelX= 62589487 AccX= 153057 PosX= -2000000 PosY= -20000
14 6544 CIT:VelX= 59468259 AccX= 313991 PosX= -2000000 PosY= -20000
15 6544 CIT:VelX= 54882725 AccX= 492388 PosX= -2000000 PosY= -20000
16 6544 CIT:VelX= 48945796 AccX= 702107 PosX= -2000000 PosY= -20000
17 6543 CIT:VelX= 41803659 AccX= 966367 PosX= -2000000 PosY= -20000
18 6543 CIT:VelX= 33632177 AccX= 966367 PosX= -2000000 PosY= -20000
```

Axis Parameters

Max X Velocity	Max X Accel	Max Y Velocity	Max Y Accel
2000	15000	2000	15000

X Scale Factor	4000
Y Scale Factor	4000

SilverLode Servo Interpolated Motion Data Format

In Make Circles, there are 6 data parameters for each data segment of X Y interpolated motion:

- segment number
- calculated segment move time
- velocity of the X axis segment move
- acceleration of the X axis segment move
- velocity of the Y axis segment move
- acceleration of the Y axis segment move

Each servo receives 4 data parameters for that particular axis specifying the motion profile of one segment. All 4 data parameters are required by the IMS command for every segment of the interpolated move. Otherwise, the motion stops and halts the program execution.

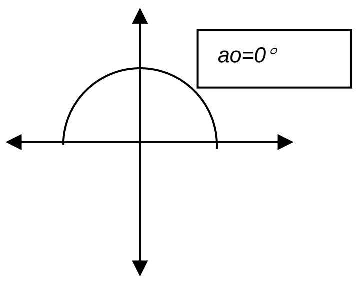
Please see the SilverLode Command Reference for details on all the commands implemented in this application.

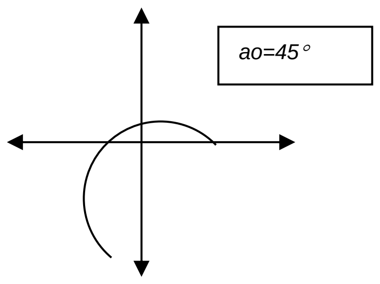
Arc Definition – Parameters for Making Circles
The following parameters define the arc to be scribed by the X Y table into circles.

# of Segments: The number of straight segments the arc is divided by.

Radius (*r*): The arc radius is the circle radius defined by the scale factor (revs, counts, etc.)

Starting Angle (*ao*): Offset starting angle of the circle (in degrees) of arc. Positive values move counter clockwise from zero (12 on the clock face).





Note: X1 and Y1 stay the same as *ao* is changed.

X,Y Offset(X1,Y1): Offset starting distance from the 0,0 position. This location is the first point on the arc.

Distance Angle: Length of arc (in degrees). 360 degrees is the distance angle of a complete circle.

Tangential Velocity
Tangential velocity of arc being scribed.

Axis Parameters
Axis Parameters include the velocity limits, acceleration limits and scaling factors for both axes.

Calculate
After designating all the parameters of the desired circle, the data points for the interpolated motion are calculated. All data points in the Circular Interpolation Table (CIT) are displayed in the Circle Setup Dialog Box center window.

Make Circles Classes
Since the Microsoft framework generates some of the code for you, only those classes with QCI specific code will be described.

All important QCI specific code is tagged with the comment character “///QCI”. This allows the programmer to quickly search the application for the “important” stuff.

CCircleSetup
Circle Setup dialog box class.

CMakeCirclesApp
Main application class.

CMakeCirclesDlg
Main dialog box class.

Circular Interpolation Table (CIT)
The following members make up the CIT:

```
long m_motorTime[MAXSEGMENTS];
long m_motorVelX[MAXSEGMENTS];
long m_motorVelY[MAXSEGMENTS];
long m_motorAccX[MAXSEGMENTS];
long m_motorAccY[MAXSEGMENTS];
long m_motorPosX[MAXSEGMENTS];
long m_motorPosY[MAXSEGMENTS];
```

Note: CCircleSetup puts the data into the CIT when the Circle Setup dialog box is exited after Calculate has been pressed. CMakeCirclesDlg accesses the CIT to send the data out to the motor

Theory of Operation
Background
The profiled move command routines generate a trapezoidal velocity profile limited by the provided acceleration, deceleration, velocity and final position, as well as the present target position and target velocity.

If the command position is far beyond the target position in the direction of movement, a simple trapezoid target velocity is realized, with the new target velocity being incremented (or decremented) by the command acceleration (deceleration) until the new command velocity is reached. If the command position comes within the deceleration distance as limited by the present target velocity and the target and command positions, then the system will decelerate in such a way to reach and stop at the desired target without overshooting. The deceleration will be the deceleration that is calculated to start 1 cycle (120us) early so as to keep the deceleration within the command deceleration while preventing overshoot for distances that would not exactly be reached using exact velocity and acceleration values. If the command position and/or deceleration are changed such that the target is forced to overshoot, the deceleration limit will be used down to zero velocity, then the acceleration limit will be used until the point determined by either the command velocity or limited by distance and the deceleration value so as to stop at the command position.

Calculations
The MakeCircles.xls spread sheet calculates the time segments, command acceleration/deceleration (same for the queued profile move to reduce the number of bytes sent), and command velocity by limiting the minimum time slice to greatest of:

1) The value given by the user for minimum update time (L2) to allow the user communications to keep up with the process;
2) The time for the slowest axis to cover the distance;
3) The maximum time required to accelerate any axis.

The times are first calculated as if the axis had infinite acceleration, next the acceleration times are considered, with ½ of the acceleration time pushed into the prior motion period. This offsets the starting points for the time slices such that each time slice contains only a single acceleration/deceleration segment and a single constant velocity slew segment. Once the axis with the longest segment time requirement has been determined, then the acceleration and velocity for all of the axis are re-calculated so as to use the required time slice.

Note that the spread sheet was done for a 3 axis system, where two axis were used to move a carriage in an x-y format, while the third axis was carried by the carriage and used to position a knife blade so as to keep it aiming into the direction of the cut. The MakeCircles example only implements two axis as a simplified general example.

MakeCircles Walk Through
We will walk through the MakeCircles.xls Excel spreadsheet and the MakeCircles C++ program in an attempt to tie the two together and clarify the theory behind them. The equations found in the spreadsheet are performed in CCircleSetup::BuildTable() of the MakeCircles program.

Basic Circle Equations
The circumference of the circle is:

$2\pi r$ [C32 of spreadsheet].

The arc of the circle(distance) that we will scribe is:

Circumference * distance angle / 360 ° [C33 of spreadsheet]

The time that it will take to traverse that distance is:

Distance / tangential velocity [C34 of spreadsheet]

And the time slice is:

Time / # of segments [C35 of spreadsheet]

In CCircleSetup::BuildTable() the above equations are represented in the following line of code:

```
TimeSlice =    fabs((2*m_radius*PI)
             *  fabs(m_distanceAngle/360)/m_maxVel/m_numCITSegments
```

The time slice is then rounded (converted to whole number of ticks, there are 8333 ticks in one second):

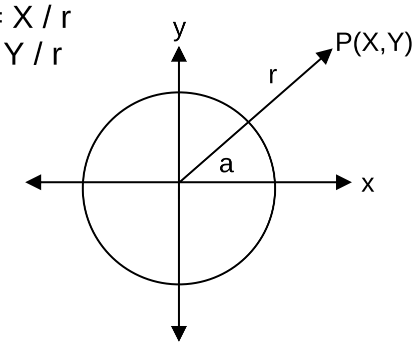
TimeRounded = TimeSlice * 8333

The rounded time slice is then scaled (converted back to sec with the resulting number being divisible by 120 microseconds (one servo cycle)):

Time Slice Scaled (Ts)
Ts = TimeRounded * 0.00012 [M2 of spreadsheet (rounded and scaled)]

Given a circle, we know from trigonometry that for any angle with measure *a* (in radians) and point (X,Y) on the circle:

cos(a) = X / r
sin(a) = Y / r



Therefore, knowing the angle we are able to find the corresponding X,Y coordinates:

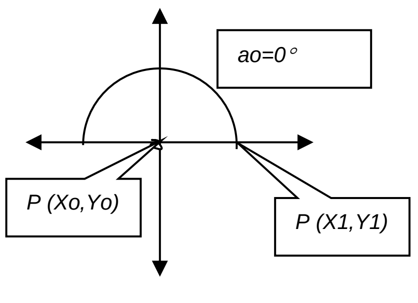
X = r cos(a)
Y = r sin(a)

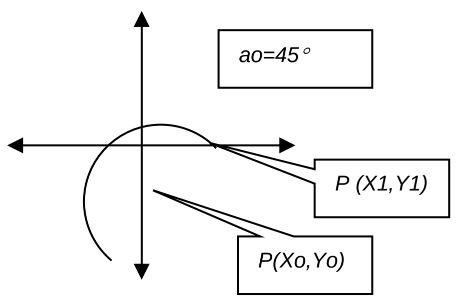
Converting the angle from degrees to radians:

$a_{rad} = a_{deg}\ 2\pi/360$ [C37 of spreadsheet]

Taking into account the starting position (as defined by the starting angle), the equations for the Center of Arc (Xo,Yo) can be found in CCircleSetup::BuildTable() as:

Center of Arc (Xo,Yo)
Xo = X1 – r * cos(2*PI*ao/360) [B of spreadsheet]
Yo = Y1 – r * sin(2*PI*ao/360) [C of spreadsheet]





Note: X1 and Y1 stay the same as *ao* is changed.

Calculations for the Arc Segments

In CCircleSetup::BuildTable() we calculate parameters for each of the segments around the arc. In the following equations, lowercase *i* represents the current arc segment. The following parameters are calculated for each segment of the arc:

The cos(a) and sin(a) (combined with the radius will give you X and Y):

cosa[i] = cos (2 * PI * a / 360) [column starting at E38 in spreadsheet]
sina[i] = sin (2 * PI * a / 360) [column starting at F38 in spreadsheet]

The position (based on infinite acceleration):

X[i] = (cosa [i] * r + Xo) * m_XscaleFactor [B6 in spreadsheet]
Y[i] = (sina [i] * r + Yo) * m_YscaleFactor [C6 in spreadsheet]

The change in position:

Delta X (dx) and Delta Y (dy)
dx[i] = X [i] – X[i-1] [E in spreadsheet]
dy[i] = Y [i] – Y[i-1] [F in spreadsheet]

The total distance:

XtotalDistance += dx [x]
YtotalDistance += dy [x]

The time it takes to travel the above distance based on the maximum velocity:
(Note: to change the velocity from RPM (revs/minute) to counts/sec
we multiply the velocity in revs/min * 1 min/60 sec * 4000 counts/rev)

Time Vel (Tvx) and Time Y (Tvy):
Tvx = fabs(dx [x] / (Vmx / 60 * 4000)) [H in spreadsheet]
Tvy = fabs(dy [x] / (Vmy/ 60 * 4000)) [I in spreadsheet]

The maximum of *TimeSliceScaled (Ts), Time Vel X (Tvx) and Time Vel Y (Tvy) are found* and placed in *Time Vel Max (Tvm*) [K in spreadsheet]

The velocities are then recalculated based on the maximum time slice:

Velocity due to Time Vel Max (Vvx and Vvy)
Vvx[i] = dx[i] / Tvm[i] [L in spreadsheet]
Vvy[i] = dy[i] / Tvm[i] [M in spreadsheet]

The acceleration times are recalculated using the above velocities:
(Note: to change the acceleration from RPM/sec to counts/sec/sec
we multiply the acceleration in revs/min/sec * 1 min/60 sec * 4000 counts/rev)

Time Acc (Tax and Tay):
Tax = fabs(Vvx[i]/(m_XmaxAccel/60*4000)) [O in spreadsheet]
Tay = fabs(Vvy[i]/(m_YmaxAccel/60*4000)) [P in spreadsheet]

The maximum of *Tax and Tay* is placed in *Time Accel Max(Tam[i])* [R in spreadsheet]

The maximum of *Tvm*, *Tam and (Tam[i+1]/2+Tam[I]/2)* is placed in *Time Min (Tmin)* [S in spreadsheet]. *Time Min* is is the minimum time slice for this segment.

The run time slice is offset from the starting point of the time slice by pushing one-half of the acceleration time into the prior motion period:

RunTime = Tmin– Tam[i+1]/2 + Tam[i]/2 [T in spreadsheet]

The velocity for each of the axis are then recalculated based on the *Run Time* slice:

Velocity (Vx,Vy)
Vx = dx[i] / Tmin [U in spreadsheet]
Vy = dy[i] / Tmin [W in spreadsheet]

The acceleration for each of the axis are then recalculated based on the Time Acc Max (Tam) time slice:

Acceleration (Ax,Ay)
Ax = (Vvx[i] – Vvx[i-1]) / Tam[i] [V in spreadsheet]
Ay = (Vvy[i] – Vvy[i-1]) / Tam[i] [X in spreadsheet]

Building Circular Interpolation Table (CIT)

The time, velocity and acceleration are converted to SilverLode native units. These parameters are stored into a Circular Interpolation Table (CIT) and called upon when the circle motion is to be performed:

The time is converted to servo cycles (ticks) by multiplying
the time (in seconds) by 1 tick / 0.00012 seconds

m_motorTime[x] = RunTime / 0.00012 [AA of spreadsheet]

The velocity is converted to SilverLode Velocity Units (SVU).

SVU = Vx (counts/sec) * 0.000120 sec/tick * 2^{26} SVU/(counts/tick)

SVUs are based on the maximum velocity value of 2,147,483,647 *[2^{31}]* being equal to 32 *[2^{5}]* counts/120 microseconds. [2^{31} / 2^{5} / .00012 is equal to 0.00012 * 2^{26}].

In the CIT, velocity and acceleration are made absolute as required by the servo. The sign of the motor position CIT entry holds the direction information for both velocity and acceleration. The spreadsheet omits the ABS function to make graphing easier.

m_motorVelX[x] = fabs(Vx* 0.00012 * 2^{26}) [AB of spreadsheet w/o ABS]
m_motorVelY[x] = fabs(Vy * 0.00012 * 2^{26}) [AD of spreadsheet w/w ABS]

The acceleration is converted to SilverLode Acceleration Units (SAU).

SAU = Ax (couts/sec^{2}) * (0.00012 sec/tick)^{2} * 2^{26} SAU/(counts/tick^{2})

SAUs are based on the maximum acceleration value of 1,073,741,823 *[2^{30}]* being equal to 16 *[2^{4}]* counts/120 microseconds/120 microseconds. [2^{30} / 2^{4} is equal to 2^{26}].

m_motorAccX[x] = fabs(Ax * 0.00012^{2} * 2^{26}) [AC of spreadsheet w/o ABS]
m_motorAccY[x] = fabs(Ay * 0.00012^{2} * 2^{26}) [AE of spreadsheet w/o ABS]

The final entry into the table is the motor position (*m_motorPosX, m_motorPosY*). The motor position is arbitrarily set to a large number (+ or – depending on the sign of the velocity) so that the motor will never get there. This causes the motion of the motor to be driven by the velocity.

In the spreadsheet, the actual motion that the motor should make in the new adjusted time slice is calculated. These calculations are provided to give the reader more insight into what the servos are doing.

accel time X[x] = (velocity X[x] – velocity X[x-1]) / acceleration X[x] [AH of spreadsheet]
slew time X[x] = delay[x] – accel time X [x] [AI of spreadsheet]
covered dist acc X[x] = velocity X[x-1] * accel time X[x] + accel X[x] * accel time X[x]
    * (accel time X[x] + 1)/2 *32 / pow(2,31) [AJ of spreadsheet]
covered dist slew X[x] = slew time[x] * velocity X[x] * 32 / pow(2,31) [AK of spreadsheet]
total covered distance X[x] = covered dist acc X[x]
    + covered dist slew X[x] [AL of spreadsheet]
total time X[x] = total time X[x-1] + delay count[x] [AM of spreadsheet]
final position X[x] = final position X[x-1] + total distance X[x] [AN of spreadsheet]

accel time Y[x] = (velocity Y[x] – velocity Y[x-1]) / acceleration Y[x] [AO of spreadsheet]
slew time Y[x] = delay[x] – accel time Y[x]   [AP of spreadsheet]
covered dist acc Y[x] = velocity Y[x-1] * accel time Y[x] + accel Y[x] * accel time Y[x]

* (accel time Y[x] + 1)/2 *32 / pow(2,31) [AQ of spreadsheet]
covered dist slew Y[x] = slew time Y[x] * velocity Y[x] * 32 / pow(2,31) [AR of spreadsheet]
total covered distance Y[x] = covered dist acc Y[x]
+ covered dist slew Y[x] [AS of spreadsheet]
total time Y[x] = total time Y[x] + delay count[x] [AT of spreadsheet]
final position Y[x] = final position Y[x-1] + total distance Y[x] [AU of spreadsheet]

accel time theta[x] = (velocity theta[x] – velocity theta[x-1]) / acceleration theta[x]
[AV of spreadsheet]
slew time theta[x] = delay[x] – accel time theta[x] [AW of spreadsheet]
covered dist acc theta[x] = velocity theta[x-1] * accel time theta[x] + accel theta[x]
* accel time theta[x] * (accel time theta[x] + 1)/2 * 32 / pow(2,31) [AX of spreadsheet]
covered dist slew theta[x] = slew time theta[x] * velocity theta[x] * 32 / pow(2,31)
[AY of spreadsheet]
total covered distance theta[x] = covered dist acc theta[x] + covered dist slew theta[x]
[AZ of spreadsheet]
total time theta[x] = total time theta[x-1] + delay count[x] [BA of spreadsheet]
final position theta[x] = final position theta[x-1] + total distance theta[x] [BB of spreadsheet]

Spread Sheet Graphs

There are 5 graphs provided in the spreadsheet. The main graph is on the “XY Plot” page and shows the X Y Plot simulating two servos drawing the described arc. X starts at B6 and Y starts at C6.

The graph in col M (page “Calcs”) is a repeat of the XY Plot. The graph is col R is an XY Plot of the actual motion (X=Col AN, Y=Col AU).

The final two graphs (col AI and AP), show the X and Y motion of a single axis. Note for a circular arc the two graphs show sinusoidal movement as expected.

Performing the Circle Move

When the ‘Do Move’ button is pressed in the Make Circle window, CmakeCircleDlg::OnDoMove() first checks to see if a Comm port is setup and the CIT(Circular Interpolation Table) has been built. The .qcp programs are then loaded into the motors. There are two .qcp programs, one for the leader (Leader queued interpolation.qcp used for the X-axis) and one for the follower (Follower queued interpolation.qcp used for the Y-axis). At this point the programs in the motors are running, but just waiting in a loop for an input from the Make Circles program.

For each segment of the circle, CmakeCircleDlg::OnDoMove() sends the motor on each axis the information in the Circular Interpolation Table (CIT), i.e. the appropriate time, position, acceleration and velocity. This information is stored into a queue. The data is written using the IMW (Interpolated Move Write Queue) command. After two points in the table have been received, a one is written to register eleven.

The Leader queued interpolation.qcp has been waiting in a loop for register eleven to be written to. When register eleven is no longer zero, the leader program clears output bit one. The Follower queued interpolation.qcp has been waiting in a loop for output bit one to be cleared. Both programs then perform an IMS (Interpolated Move Start) using the data in the queue.

CmakeCircleDlg::OnDoMove() keeps sending the data from the CIT to the motors and the motors are moving according to the data in the queue. This keeps going until all of the data in the queue has been sent. At that time, CmakeCircleDlg::OnDoMove() writes a zero to register eleven. This causes the leader program to set the output bit one and go back to the beginning of the program, where is again waits for register eleven to be written to. The follower program in turn sees the output bit set and jumps back to the beginning of its program and waits for the output bit to be cleared.