Aug-07

Application Note #3415

Methods of Addressing Applications Involving Inverse Kinematics

I. Introduction

Mechanical systems are often engineered in such a way that motors and actuators are not aligned in a Cartesian configuration. However, the machine will need to follow motion paths and positions based on real 3-dimensional coordinates. The machine programmer is required to convert from 3-D Cartesian 'real' space into axis position information, based on the mechanical layout of the machine. The process of conversion from real space into machine coordinates is called Inverse Kinematics.

Examples of applications that might require inverse kinematics:

- Anthropomorphic welding robot that must follow a complex 3-D path. This is a 'human-like' mechanism with a shoulder, an elbow, and an optional wrist joint.
- SCARA-type robot arm placing components on an X-Y grid. Selective Compliant Assembly Robot Arms are common in semiconductor and other pick-and-place applications.
- Medical patient positioning gurney that must rotate around a virtual pivot
- Hexapod positioning device. A Hexapod is a six linear-axis robot that allows for complex pan/tilt/displacement motion.

There are three general ways to address inverse kinematics applications using Galil motion controllers.

1. Inverse kinematics equations solved by host computer
2. Inverse kinematics equations solved in DMC code on the motion controller
3. Inverse kinematics equations solved on the controller within the servo loop

Details of the solution methods are listed in Section II.

II. Solution Methods

1. Inverse Kinematics Equations Solved by Host Computer

If the machine is extremely complex, or if the mechanical arrangement of components can vary, it is sometimes best to use a host PC to convert 3D commands into machine data. A host application, written in C++ or other language, takes the 3D data, runs the equations, and transmits motor position data to the Galil controller in the form of Contour data. The Galil controller has no knowledge of the configuration of the machine, and simply moves the motors in accordance with the data. Any multiple solutions, keepout zones, and boundary conditions must be controlled by the host PC.

2. Inverse Kinematics Equations solved in DMC Code

Galil motion controllers have the ability to process kinematics equations within an on-board DMC program. This is an excellent way to prove the equations and boundary conditions. On Galil's Accelera-series motion controllers, it is possible that the equation processing time is fast enough such that position updates can be performed in only a few milliseconds. This all depends on the complexity of the equations and the number of axes involved. An example of a simple 2-D transformation can be found in Section III below.

3. Inverse Kinematics solved in Firmware

If the machine equations are complex, or very fast update of positions is required, but a host computer is not available, the equations can be ported into the Galil's servo update loop. This firmware modification will involve detailed discussions with a Galil Applications Engineer in defining the scope of the Non-Recurring Engineering (NRE) project. It is necessary that the equations and boundary conditions be tested in DMC code (Option 2 above) before translating to firmware. As the equations are ported in to the firmware, the flexibility is limited to variables associated with these equations. Only an additional NRE can change the equations. The benefits of this solution are that the equations run within the servo sample. DMC program space and command processing time is left free for other programmatic functions.

III. Simple 2-D Example Solved in DMC Code

An application uses a 2-axis anthropomorphic robot arm as shown in Figure 1. There are two joints; a shoulder and an elbow. The programmer would like to command the axis in XY coordinates. The controller has on-board inverse kinematics equations that translate XY points into position commands for the two rotary motors. Figure 1 shows an example of such a robot.

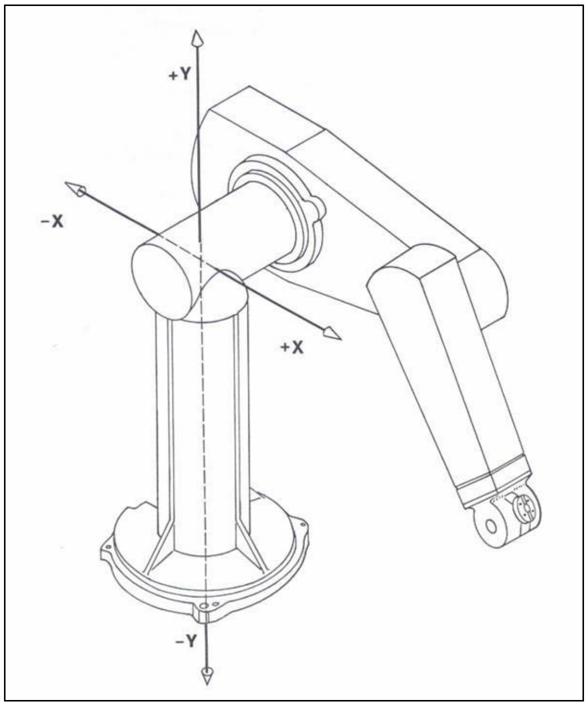


Figure 1- 2 Axis Robot

Figure 2 shows the same mechanics reduced to a 2-D representation.

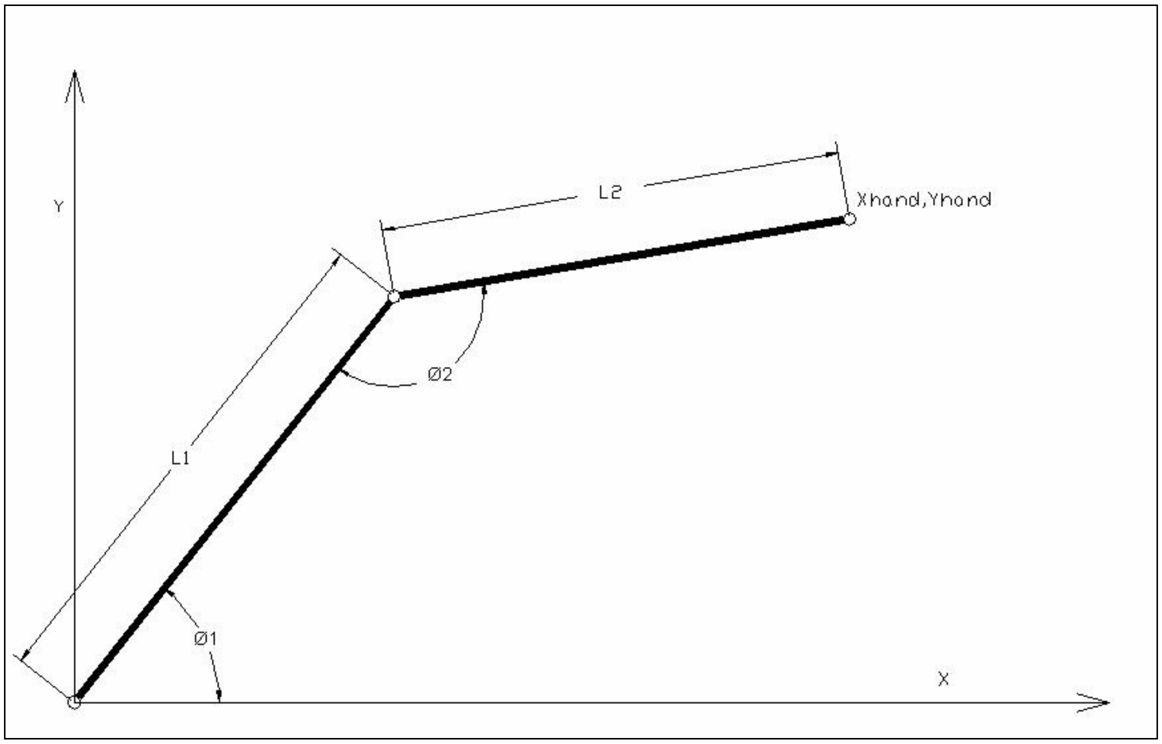


Figure 2- 2-Axis Robot shown in 2D

Definitions:
L1 = length of first joint, from shoulder pivot to elbow pivot
L2 = Length of second joint, from elbow pivot to end-effector
Xhand = X axis position commanded by operator
Yhand = Y axis position commanded by operator
θ1 = Angle of shoulder motor
θ2 = Angle of elbow motor

Given:
Xhand, Yhand, L1, L2

Find:
θ1, θ2

To assist in solving this example, add an imaginary straight line from the shoulder to the wrist. Figure 3 shows such a visual aide.

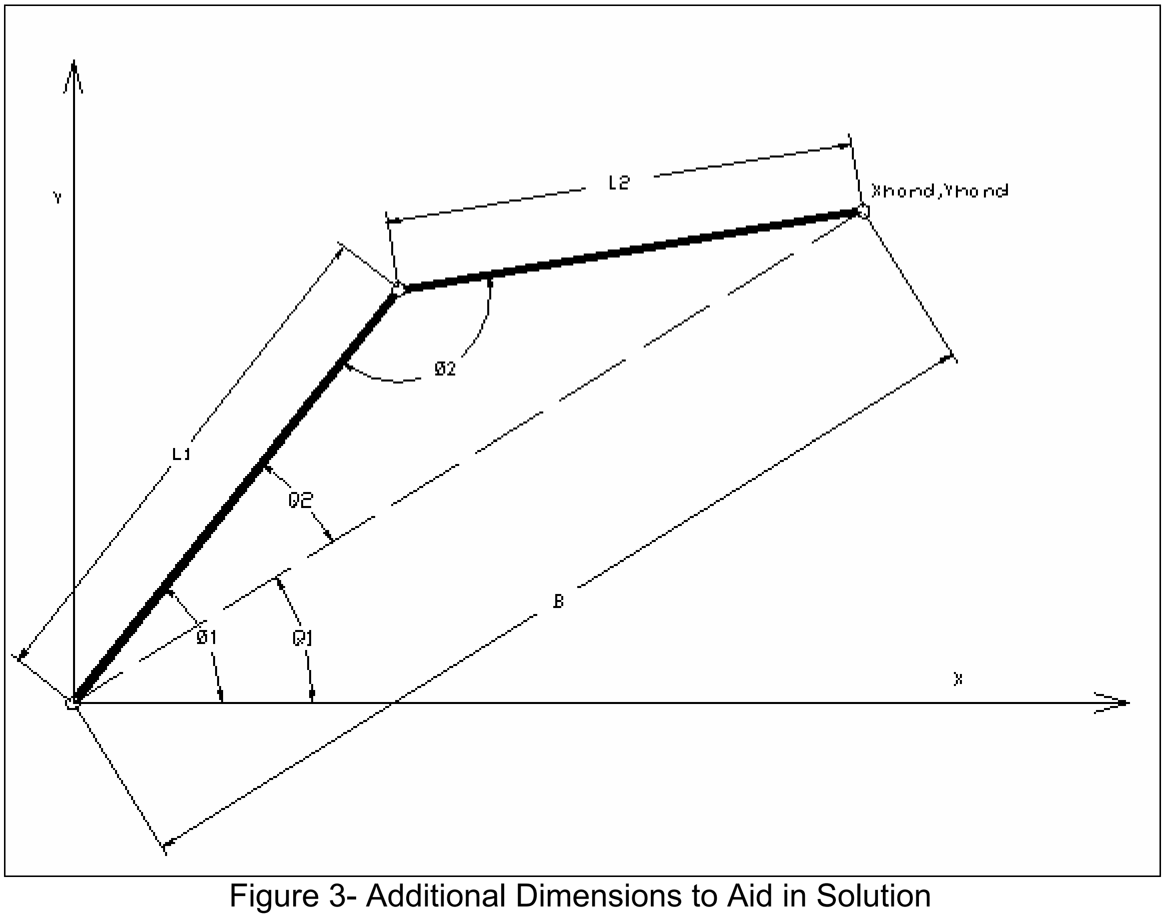


Figure 3- Additional Dimensions to Aid in Solution

Additional Definitions:
B: Length of imaginary line
Q1: Angle between X axis and imaginary line B
Q2: Interior angle between imaginary line B and link L1

Equations:
B^2 = Xhand^2 + Yhand^2
Q1 = atan(Yhand/Xhand)
Q2 = acos[(L1^2 – L2^2 + B^2)/(2*L1*B)]
θ1 = Q1 + Q2
θ2 = acos[(L1^2 + L2^2 –B^2)/(2*L1*L2)]

This example will focus on a specific move. The operator wishes to move from (Xhand, Yhand) location of (150,100) to a new position of (250,200). Figure 4 shows the motion trajectory.

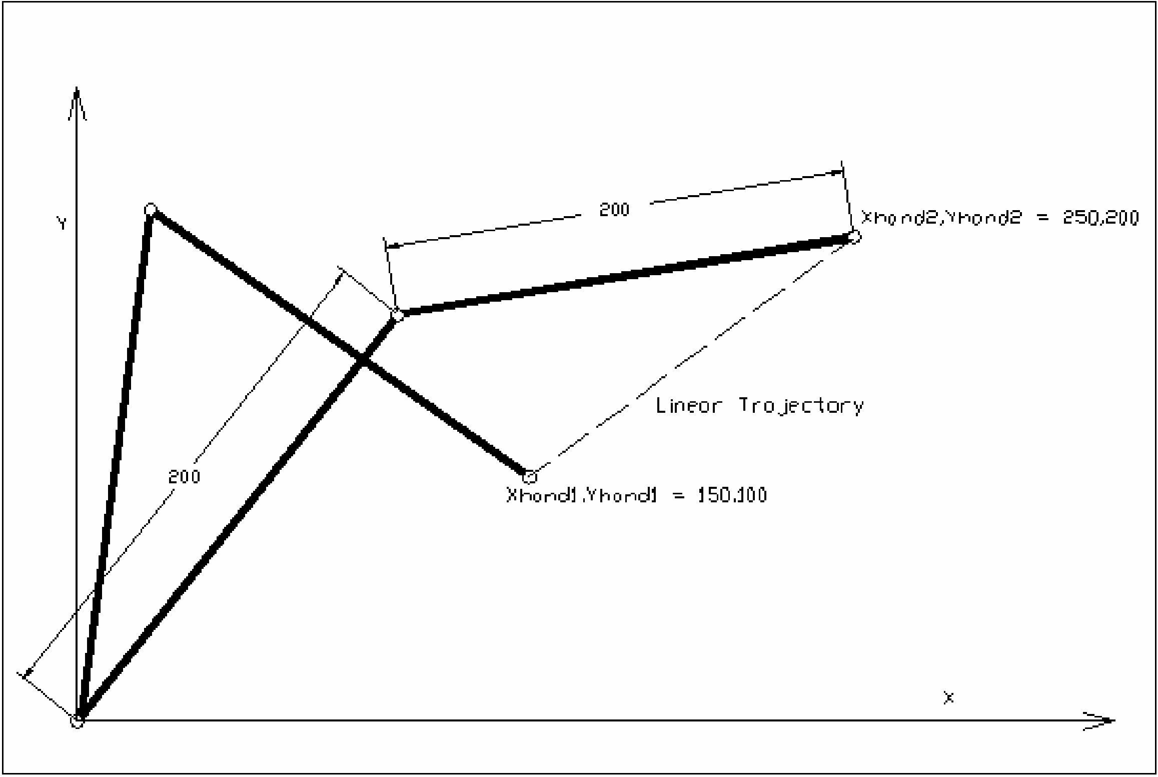


Figure 4- Motion Trajectory

This example uses virtual axes in order to profile along the linear trajectory. As the controller profiles along this path, X-Y points are captured and stored in an array. This data is used later by the controller in the inverse kinematics equations. To calculate XY trajectory, set up Vector Mode and Auto-data capture. Capture the commanded XY positions every 2 milliseconds.

Galil Code to generate XY coordinates along motion path

```
#CALC
TL0,0;ER-1,-1;'For pure calculation.  No motion possible
VMXY;'Virtual 2-D coordinate plane
VS1000;VA20000;VD20000;'Vector parameters
DP150,100;'Define virtual axes as starting point of 150,100
DA,*[];'De-allocate array space
DMXPOS[100],YPOS[100];'Arrays for trajectory data
RAXPOS[],YPOS[];'Define arrays for data capture
RD_RPX,_RPY;'Record Commanded position of virtual axes
VP100,100;'Command a delta position change
VE;'End of vector sequence
RC1;'Begin recording at 2 msec intervals
BGS;'Begin virtual motion
AMS;'After profiled motion complete
MG"DONE";'Message
EN;'End Program
```

If the programmer uploads the arrays XPOS and YPOS and displays in a graph, one can see the linear motion with the points along the trajectory. Figure 5 shows an Excel plot of the motion path.

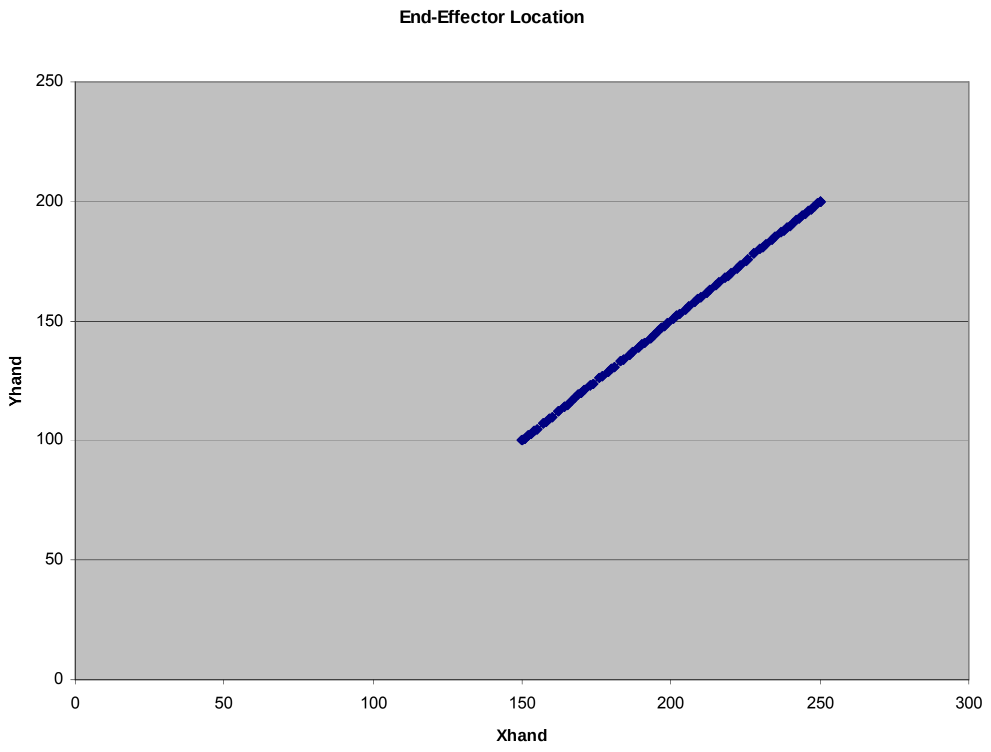


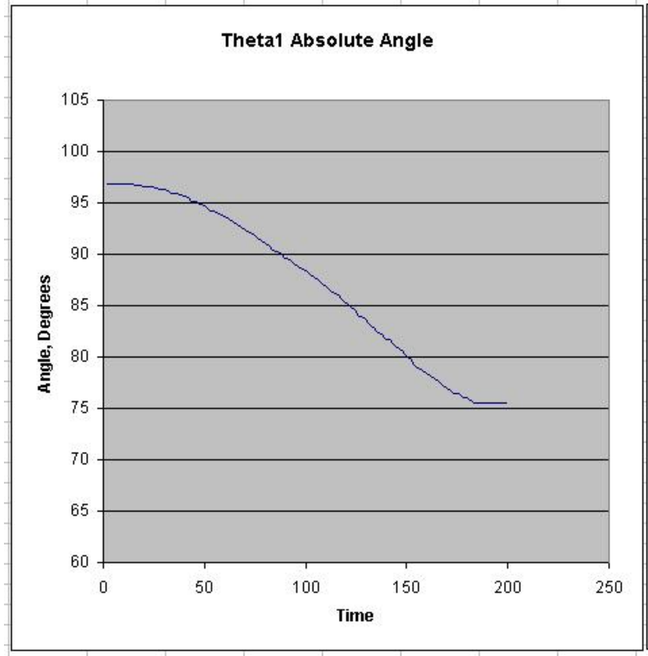
Figure 5- Plot of XY Coordinate Data

The next step involves taking those array points in XPOS and YPOS and applying the inverse kinematics equations on this data. This will result in the absolute angular positions of the joints.

Galil code to compute angles of the joints over the distance traveled

```
#GENR8
'L1 = Length of first linkage
'L2 = Length of second linkage
'B= Length of hypotenuse
'Q1 = Angle between X axis and Line B
'Q2 = Interior angle from B to L1
'Theta1 = Absolute angular position of first motor
'Theta2 = Absolute angular position of second motor
L1=200;'Define first segment length
L2=200;'define second segment
DA,Theta1[];DA,Theta2[];'Define arrays for actual motor positions
DMTheta1[100],Theta2[100];'Build arrays for contour data
N=0;'Array increment
#CALC2;'Perform calculations based on equations
B=@SQR[(Xpos[N]*Xpos[N])+(Ypos[N]*Ypos[N])]
Q1=@ATAN[(Ypos[N])/(Xpos[N])]
Q2=@ACOS[((L1*L1)-(L2*L2)+(B*B))/(2*L1*B)]
Theta1[N]=Q1+Q2
Theta2[N]=@ACOS[((L1*L1)+(L2*L2)-(B*B))/(2*L1*L2)]
N=N+1
JP#CALC2,N<100
MG"DONE GENR8"
EN;'Done routine
```

If the programmer uploads the arrays Theta1 and Theta2 and displays in a chart, one can see the calculated actual motor angle. Figure 6 shows an Excel plot of the motor positions.



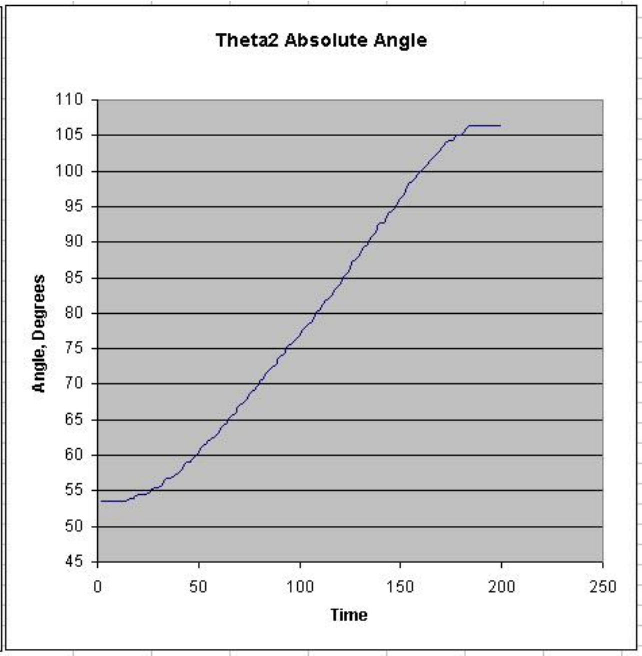


Figure 6- Calculated motor angles

This data is currently absolute motor angles. The next step involves calculation of motor delta values.

Galil Code to calculate incremental contour data from absolute angle data

```
#CONTCLC
NOTE THIS CONVERTS FROM ABSOLUTE ANGLES
NOTE TO QUADRATURE ENCODER COUNTS
NOTE ASSUME ENCODER VALUE OF ZERO LIES ON +X
NOTE AND CCW COUNTS UP
'Encres = ENCODER COUNTS/DEGREE SAME FOR BOTH MOTORS
Encres=4000/360
DA,Enc1[];DA,Enc[2]
DMEnc1[100],Enc2[100];'BUILD ARRAYS FOR ENCODER RELATIVE DATA
N=0;'REINITIALIZE COUNTER
#CALC3
Enc1[N]=Encres*((Theta1[N+1])-(Theta1[N]))
Enc2[N]=Encres*((Theta2[N+1])-(Theta2[N]))
N=N+1
JP#CALC3,N<99
MG"DONE CONTCLC"
EN
```

The final step is to command the actual motors with the Enc1 and Enc2 array data.

Galil code to command contour data

```
#CONTMOV
TL9.998,9.998;'Ensure ability of motors to move
DP0,0;'Move is relative so define positions as zero
NOTE THIS ROUTINE SENDS ALL 100 CONTOUR MOVES
CMXY
DT1
N=0;'INCREMENT RESET
#LOOP3
CDEnc1[N],Enc2[N]
N=N+1
JP#LOOP3,N<100
#WAIT;JP#WAIT,_CM<>511
CD0
DT0
EN
```

Based on the fact that the encoder positions were zeroed before the start of this segment, Figure 7 shows actual encoder positions for the real X and Y motors.

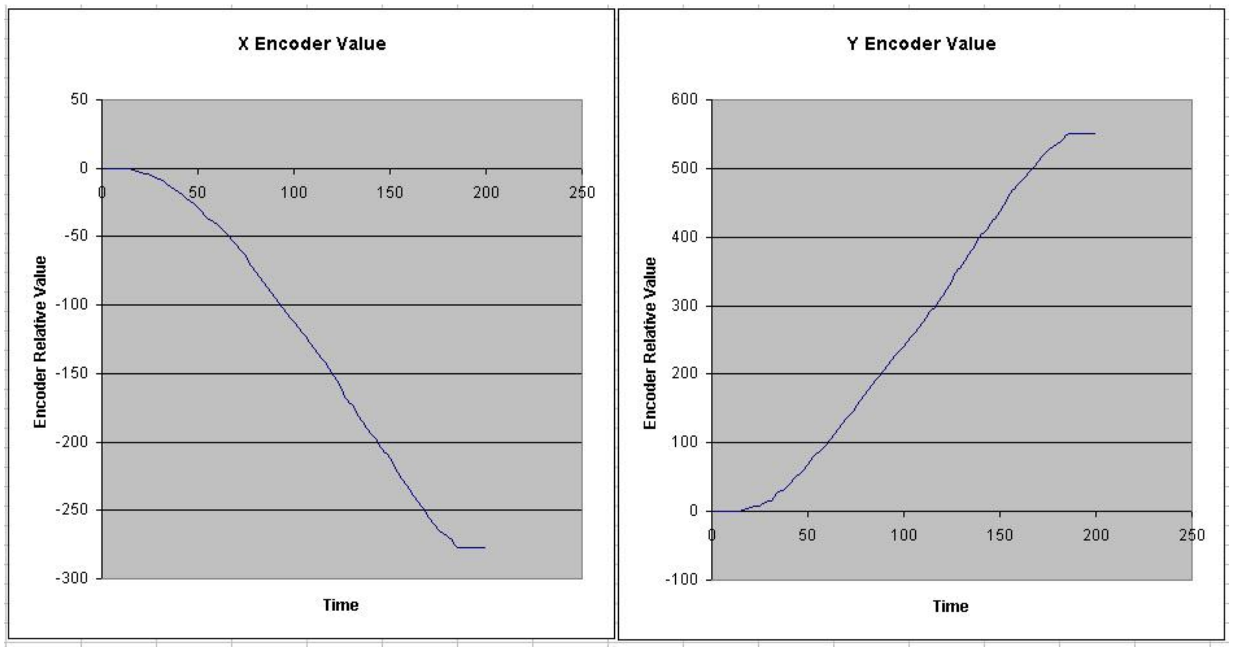


Figure 7- Actual Encoder Values

This example shows a relatively simple 2-axis inverse kinematics transformation. As axis count increases, the complexity of these equations goes up quickly. Feel free to contact an Applications Engineer at Galil Motion Control for assistance at 1.800.377.6329