Contouring and Coordinate Transformation

Introduction

Some applications require the servos to move an object in straight lines in a real world Cartesian coordinate system (X, Y and Z). If each axis is directly related to X, Y and Z, this can be quite simple. But if the servo axes do not correspond directly to X, Y and Z, you must employ a technique known as coordinate transformation. Consider the following two axis example. The diagram shows a robot like mechanism which can move on two axes, Boom extension and Elevator extension. As you can see, independent changes in either of these two axes will not result in the load moving in straight line along the X or Z axes. Changes in Boom extension will result in a diagonal line. Changes in Elevator extension result in an arc. Clearly to achieve straight line motion parallel to either the X or Z axes, both the Boom and Elevator must move together. This technote explains how to accomplish this in an ORMEC ORION Motion Controller using MotionBASIC®.

Developing the Equations

First you must develop a set of equations that, given any X, Z position, will give you the corresponding positions for the Boom and Elevator. To do this, you start by developing a simplified mechanical diagram as shown here. In this case the length of the line $EC$ represents the vertical (Z) position and the length of $EF$ represents the horizontal (X) position. Likewise, $CF$ is the Boom extension and $BA$ is the elevator extension.

There is no trick to this, the only thing that will help you is a knack for trigonometry. In this example, the solution is;

First, given the desired values for X and Z ($EF$ and $EC$), using triangle $ECF$,

$$\angle ECB = \arctan\left(\frac{EF}{EC}\right)$$

Since angle $DCA$ is known and fixed,

$$\angle BCA = \pi - \angle DCA - \angle ECB$$
\[ \angle BCA = \pi - \angle DCA - \arctan \left( \frac{EF}{EC} \right) \]

Now, applying the cosine rule to triangle \(ABC\),

\[
\cos(\angle BCA) = \frac{CA^2 + CB^2 - BA^2}{2 \cdot CA \cdot CB}
\]

Rearranging this to solve for \(BA\),

\[
2 \cdot CA \cdot CB \cdot \cos(\angle BCA) = CA^2 + CB^2 - BA^2
\]

\[
BA = \sqrt{CA^2 + CB^2 - 2 \cdot CA \cdot CB \cdot \cos(\angle BCA)}
\]

This gives us the Elevator extension. For the Boom extension we simply apply Pythagorus's theorem to triangle \(ECF\),

\[
CF = \sqrt{EF^2 + EC^2}
\]

Having developed the equations to convert \(X\) and \(Z\) to Elevator and Boom extension, we must now develop the reverse transforms that allow us to take Elevator and Boom extension and calculate the real world coordinates \(X\) and \(Z\),

\[
\angle ECB = \pi - \angle DCA - \arccos \left( \frac{CA^2 + CB^2 - BA^2}{2 \cdot CA \cdot CB} \right)
\]

Unfortunately, MotionBASIC\textsuperscript{®} does not have an arccos function, so we have to use the following trig identity to solve for angle \(ECB\),

\[
\arccos(\theta) = \arctan \left( \frac{\sqrt{1 - \theta^2}}{\theta} \right)
\]

Using the substitution

\[
\theta = \frac{CA^2 + CB^2 - BA^2}{2 \cdot CA \cdot CB}
\]

then

\[
\angle ECB = \pi - \angle DCA - \arctan \left( \frac{\sqrt{1 - \theta^2}}{\theta} \right)
\]

Having found angle \(ECB\), it is easy to find \(EC\) and \(EF\) which correspond to \(Z\) and \(X\) respectively,

\[
Z = EC = CF \cdot \cos(\angle ECB)
\]
\[
X = EF = CF \cdot \sin(\angle ECB)
\]

**Programming**

It is tempting to believe that you can simply implement these formula as a CAM or PROFILE and simple plug in the desired positions and have the axes move to them. However, it is not that
easy. As each axis moves, it affects the values used to calculate the position of the other. This means the equations must be calculated in real time as the values are needed to command the motion.

To do this, the program requires two transformation routines, one to handle the forward transforms and one the reverse transforms.

CALC.BOOM.ELEV:
'---- calculate Boom and Elevator extension given X! and Z!
   BCA!=PI!-DCA!-ATAN(X!/Z!)
   BA!=SQR(CA!^2+CB!^2-2*CA!*CB!*COS(BCA!))
   CF!=SQR(X!^2+Z!^2)
RETURN
'
CALC.XZ:
'---- calculate X! and Z! given Boom and Elevator extension
   THETA!=((CA!^2+CB!^2-BA!)^2)/(2*CA!*CB!)
   ECB!=PI!-DCA!-ATAN(SQR(1-THETA!^2)/THETA!)
   Z!=CF!*COS(ECB!)
   X!=CF!*SIN(ECB!)
RETURN

To start a move from the current position you first need calculate the current X! and Z! positions.

BA!=POS.ACT@(ELEV~)*1000
CF!=POS.ACT@(BOOM~)*1000
CALC.XZ

Now, knowing the target coordinates, XX! and ZZ!, we need to calculate the total distance between the starting point and the target.

XDIST!=XX!-X!                'total X distance
ZDIST!=ZZ!-Z!                'total Z distance
DIST!=SQR(XDIST!^2+ZDIST!^2) 'total length of vector

We are going to issue new position commands to the axes every 25ms, so we need to know how many command values we need to calculate. Let's use a speed in inches per second.

TICKS&=DIST!/(SPEED!*0.025)

The last step is to calculate the positions and command the moves.

XSTART!=X!                         'remember the X starting position
ZSTART!=Z!                         'remember the Z starting position
COUNT&=1                           'initialize the tick count
WHILE DIO@(MOVE)=ON AND COUNT&<=TICKS&
   X!=XSTART!+XDIST!*COUNT&/TICKS&  'new X position
   Z!=ZSTART!+ZDIST!*COUNT&/TICKS&  'new Z position
   CALC.BOOM.ELEV                   'new axis positions
   MOVE BOOM~ TO CF!*1000 IN 25,0,0 'command the Boom move
   MOVE ELEV~ TO BA!*1000 IN 25,0,0 'command the Elevator move
   COUNT&=COUNT&+1                 'increment the tick count
WEND
IF DIO@(MOVE)=ON THEN              'did the move get cancelled
X!=XX! :Z!=ZZ!                   'set X and Y to the target positions
CALC.BOOM.ELEV                   'calculate the target axis positions
MOVE BOOM~ TO CF!*1000 IN 25,0,0 'do a last fix up move incase the move
MOVE ELEV~ TO BA!*1000 IN 25,0,0 '  did not divide into an exact number
ENDIF                              '  of ticks

Axis Configuration

Normally, MotionBASIC® requires the axes to come to a complete stop between MOVE
TOs. If we allowed this, the motion would be very jerky. To prevent it, we need to turn the
velocity feedforward off and execute the moves with a zero acceleration and deceleration time.
This means we must also turn the acceleration and deceleration limit checking off and reduce the
position gain of the axes. We do all this as follows.

BOTH~==BOOM~+ELEV~
KVF@(BOTH~)=0
ACL.MAX@(BOTH~)=0
DCL.MAX@(BOTH~)=0

With Velocity Feedforward off, we must reduce the position gain to a value that allows the
axes to accelerate to their maximum speed without putting the drives in current limit. For a
typical mechanism this would be about 1 inch per minute/mil. Smaller more responsive
mechanisms could operate at higher gains while larger more sluggish mechanisms will require a
lower gain. It is important that both axes have the same position gain. To set the gain in inches
per minute/mil, use the following formula,

POS.GAIN=1
KP@(BOOM~)=POS.GAIN*LOOP.RATE@*VLTC@(BOOM~)/395
KP@(ELEV~)=POS.GAIN*LOOP.RATE@*VLTC@(ELEV~)/395

It may also be necessary to adjust the PERR.MAX@ parameter as well to avoid position
error faults due to the reduced position gain and the absence of velocity feedforward.