ORMEC

Coordinate Transformations for a Two Axis SCARA Robot

Tech Note # 32

Introduction

A SCARA robot is simple mechanism frequently used for high speed pick and place applications. In a two axis SCARA robot, the axes control the angles of the two joints. The position of the end effecter moves in the X, Z plane and is the resultant of the angles and the lengths of the arms. Many applications require the end effecter to move in straight lines in the X, Z coordinate system. To accomplish this one needs to solve the coordinate transformation equations to convert a desired X, Z position to the joint angles required.

Coordinate Transformation Solution

The following diagram shows a simplified mechanism. We need to state the equations for X and Z then solve them for A and B. The equations for X and Z are,

$$x = p \cdot \cos(A) + q \cdot \cos(B) \qquad (1)$$

$$z = p \cdot \sin(A) + q \cdot \sin(B) \qquad (2)$$

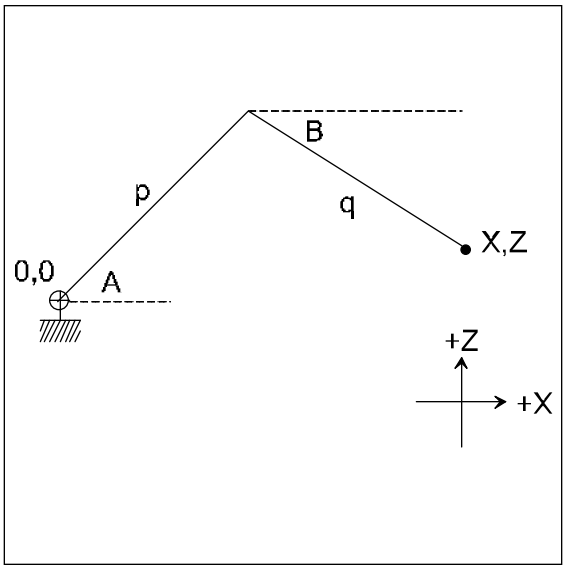


Figure 1, Simplified Mechanical Diagram

Now,

$$\sin(\alpha) = \sqrt{1 - \cos^2(\alpha)} \qquad (3)$$

so,

$$z = p \cdot \sqrt{1 - \cos^2(A)} + q \cdot \sin(B) \qquad (4)$$

Solving equation 1 for cos(*A*) we have,

$$\cos(A) = \frac{x - q \cdot \cos(B)}{p} \qquad (5)$$

Substituting this in equation 4,

$$z = p \cdot \sqrt{x - \frac{(x - q \cdot \cos(B))^2}{p^2}} + q \cdot \sin(B) \qquad (6)$$

Solving equation 6 for B gives,

$$B = \arctan\left[\frac{2 \cdot z \cdot q \pm \sqrt{2 \cdot (y^2 \cdot q^2 + x^2 \cdot p^2 - x^2 \cdot y^2 + x^2 \cdot q^2 + p^2 \cdot y^2 + p^2 \cdot q^2) - x^4 - p^4 - y^4 - q^4}}{(x^2 - p^2 + y^2 + q^2 + 2 \cdot x \cdot q)}\right]$$

Likewise,

$$\sin(\alpha) = \sqrt{1 - \cos^2(\alpha)} \tag{8}$$

so substituting this in equation 2,

$$z = p \cdot \sin(A) + q \cdot \sqrt{1 - \cos^2(B)} \tag{9}$$

Solving equation 1 for cos(*B*), we have,

$$\cos(B) = \frac{x - p \cdot \cos(A)}{q} \tag{10}$$

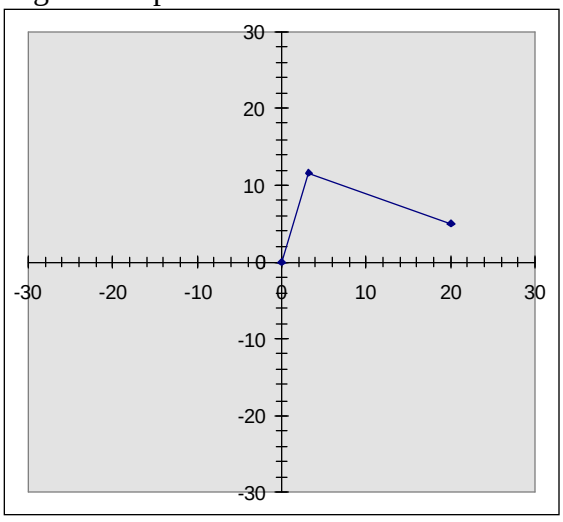
Which when substituted in equation 9 gives,

$$x = p \cdot \sin(A) + q \cdot \sqrt{\frac{1 - (x - q \cdot \cos(A))^2}{q^2}} \tag{11}$$

Solving equation 11 for A gives,

$$A = \arctan\left[\frac{2 \cdot y \cdot p \pm \sqrt{2 \cdot (y^2 \cdot q^2 + x^2 \cdot p^2 - x^2 \cdot y^2 + x^2 \cdot q^2 + p^2 \cdot y^2 + p^2 \cdot q^2) - x^4 - p^4 - y^4 - q^4}}{(x^2 - q^2 + y^2 + p^2 + 2 \cdot x \cdot p)}\right]$$

By substituting values in the equations for A and B you can determine whether to use the negative or positive roots.



Positive root for A
Negative root for B

Figure 2, Solution 1

Negative root for A
Positive root for B

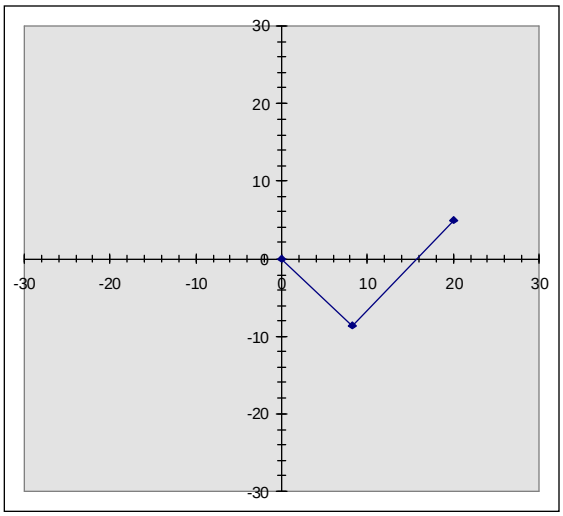


Figure 3, Solution 2

For most applications solution 1 will be the one used.

Having calculated the angle B, one small adjustment needs to be made. An angle of 0 degrees at B represents the situation when the outer arm is lined up exactly with the inner arm. Our calculations assume the angle B is the angle between the outer arm and the X axis. To adjust for this we simply subtract the angle A from B.

Program Implementation

This technique breaks the move up into small, 10 ms pieces, and commands a series of moves to each successive position. MotionBASIC® would normally turn each piece of the move into a trapezoidal move with an acceleration, cruise and deceleration. In our case we do not want the axes to decelerate to a stop between each piece of the move so we will command each piece with a zero acceleration and deceleration time. Since the speed change from each piece of the move to the next is quite small, instantaneous acceleration is not a problem during the move itself. It is however a problem at the start and end of the move when the axes would have to accelerate to and from rest instantaneously.

To avoid having these accelerations become a problem, you need to turn velocity feed forward off by setting KVF@=0 for both axes. You will also need to set ACL.MAX@=0 and DCL.MAX@=0 for both axes. Finally you will need to set the KP@ value to provide a reasonable, identical position gain for both axes. A reasonable value for position gain is 1 inch per minute/mil. You may use a higher gain for more responsive systems or a lower gain for heavier, more sluggish systems.

$$\text{Position gain} \quad G = \frac{395 \cdot KP@}{LOOP.RATE@ \cdot VLTC@} \quad \text{inches per min/mil}$$

$$KP@ = \frac{G \cdot LOOP.RATE@ \cdot VLTC@}{395} \ \%$$

Depending on the position gain you use and the characteristics of your mechanism, you may need to adjust or disable PERR.MAX@ position error checking to avoid nuisance position error faults. It is very important for accuracy to have both axes with identical position gains.

Program Listing

```
POWERUP:
  MP.CONFIG
  A~={1}
  B~={2}
  BOTH~=A~+B~
'---- set up the axis parameters for interpolation
  ACL.MAX@(BOTH~)=0
  DCL.MAX@(BOTH~)=0
  POS.GAIN=1
  KVF@(BOTH~)=0
  KP@(A~)=POS.GAIN*LOOP.RATE@*VLTC@(A~)/395
  KP@(B~)=POS.GAIN*LOOP.RATE@*VLTC@(B~)/395
'---- initialize some variables
  PROG.INIT
  ENABLE   'enable the axes
'---- get the current joint angles and calculate
'     the corresponding X and Z positions
  GET.XZ.POSITION
'---- set first position and move to it
'     (S! is the speed in inches/second)
  X2!=20 :Z2!=0   :S!=1 :MOVE.TO
'---- set second position and move to it
  X2!=20 :Z2!=-10 :S!=1 :MOVE.TO
  WAIT UNTIL DSP.DONE@(BOTH~) AND IN.POS@(BOTH~)
  MODE@(BOTH~)=0
END
'
GET.XZ.POSITION:
  A1!=POS.ACT@(A~)*2*PI!/3600       'convert to radians
  B1!=A1!+(POS.ACT@(B~)*2*PI!/3600) 'convert to radians
  X1!=P!*COS(A1!)+Q!*COS(B1!)       'calculate X
  Z1!=P!*SIN(A1!)+Q!*SIN(B1!)       'calculate Y
RETURN
'
SOLVE.MOVE:
  XD!=X2!-X1!                   'X distance
  ZD!=Z2!-Z1!                   'Z distance
  PATH.LEN!=SQR(XD!^2+ZD!^2)    'vector path length
  DELTA!=S!*UPD.TIME.S!         'distance increment along vector
                                'per update
  N&=ABS(INT(PATH.LEN!/DELTA!)) 'number of updates
  DELTA.X!=XD!/N&               'Increment on X per update
  DELTA.Z!=ZD!/N&               'Increment on Z per update
RETURN
'
ENABLE:
'---- clear faults and enable
  OTL.FWD@=0:OTL.REV@=0:AFAULT@=0:FAULT@=0:WAIT 300:MODE@=5
RETURN
```

```
'
MOVE.TO:
  SOLVE.MOVE                      'set up the interpolator
  X!=X1!  :Z!=Z1!                 'set interpolator output to
                                  'present positions
  X1!=X2! :Z1!=Z2! :COUNT&=1      'set start position for next move
                                  'to end position for this one
  WHILE INKEY$="" AND COUNT&<=N&
    IF COUNT&=N& THEN
      X!=X2! :Z!=Z2!              'make sure we hit the end position
                                  'exactly
    ELSE
      X!=X!+DELTA.X!              'Increment X and Z along the vector
      Z!=Z!+DELTA.Z!
    ENDIF
    CALC.AB                       'calculate the A and B targets
'----- move the axes
    MOVE BOTH~ TO AXIS.VAR@ IN UPD.TIME.MS,0,0
    COUNT&=COUNT&+1               'Increment the counter
  WEND
RETURN
'
CALC.AB:
  KB1!=2*Z!*Q!
  KB2A!=2*(Z!^2*Q!^2+X!^2*P!^2-X!^2*Z!^2+X!^2*Q!^2+P!^2*Z!^2+P!^2*Q!^2)
  KB2B!=-X!^4-P!^4-Z!^4-Q!^4)
  KB3!=X!^2-P!^2+Z!^2+Q!^2+2*X!*Q!
  KA1!=2*Z!*P!
  KA3!=X!^2-Q!^2+Z!^2+P!^2+2*X!*P!
  B!=2*ATN((KB1!-SQR(KB2A!+KB@B!))/KB3!)
  A!=2*ATN((KA1!+SQR(KB2A!_KB2B!))/KA3!)
  'convert A B to tenths of a degree
  AXIS.VAR@(A~)=3600*A!/(2*PI!)
  AXIS.VAR@(B~)=3600*(B!+PI!-A!)/(2*PI!)
RETURN
'
PROG.INIT:
  PI!=3.14159
  HALF.PI!=PI!/2
  UPD.TIME.MS=10                '(ms)
  UPD.TIME.S!=UPD.TIME.MS/1000  '(seconds)
  P!=12  'these set the geometry of the robot
  Q!=18
RETURN
```