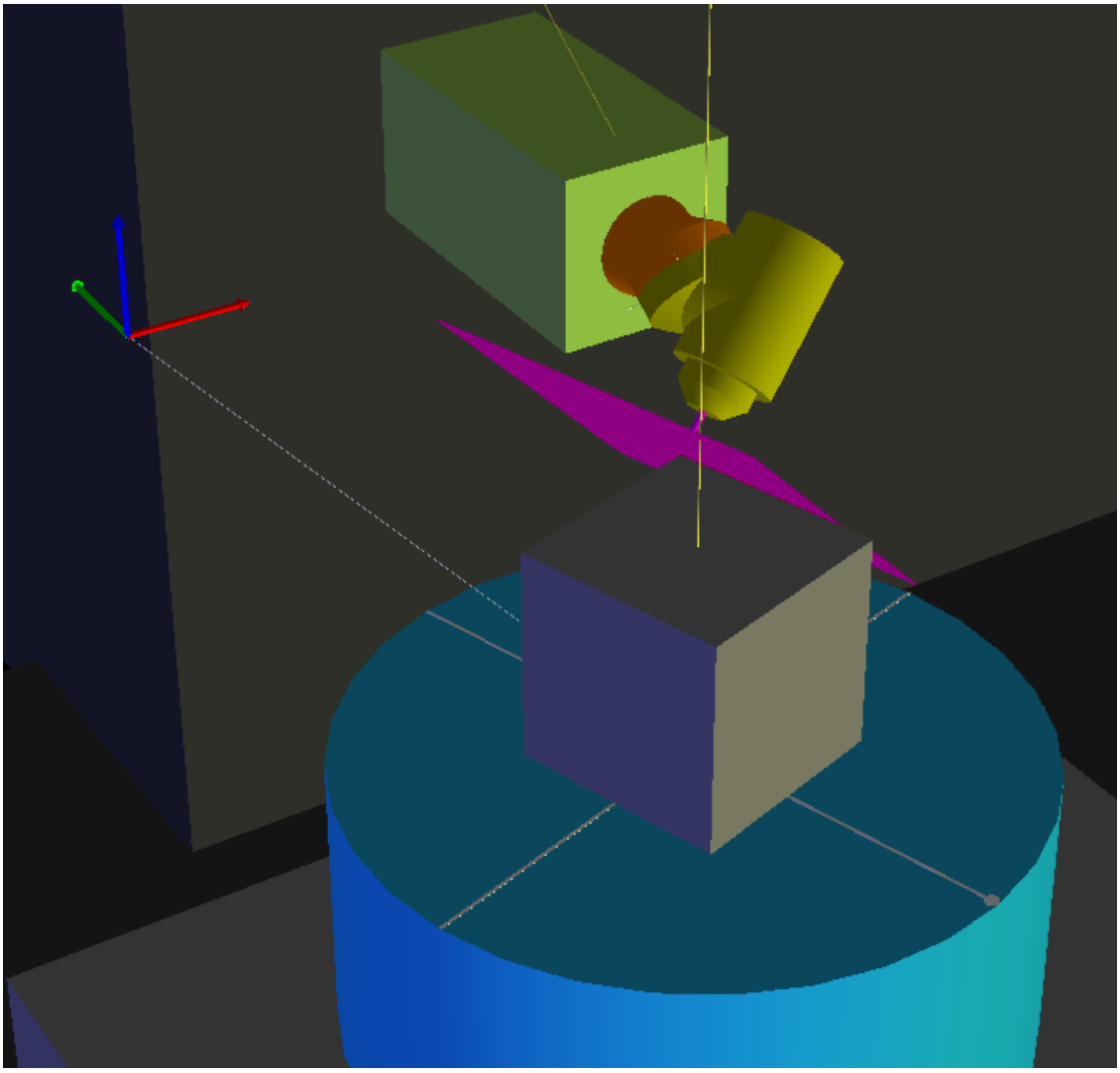
# XYZAB_TDR kinematics for LinuxCNC

David Mueller

July 21, 2026



## 1 Introduction

This paper describes how to derive the kinematic model for a 5-axis machine tool in an XYZBC configuration with Tool side B nutating rotation of the spindle and work side C (table) rotation. The model presented here will be named 'XYBC-SNTR'.

The method used here will be a step by step approach. Starting with a working kinematic model

for a single rotary axis all the required elements will be added to build a complete kinematic model for the machine.

The final model includes tool-length compensation, compensation for spindle pivot length and compensation for setups where the machine reference point is not located on the rotation-axis of the rotary table. The nutation angle is adjustable from 0 to 90° as measured from the vertical.

In this document only basic mathematical functions are used so the kinematic models derived can be used directly in the 'userkins.comp' template file provided with the LinuxCNC installation. All calculations can be done whithout the use of any computer algebra systems, however the use of computer assistance like 'sage' will make the process of matrix multiplication much less error prone.

Note that there are other and potentially more computationally efficient ways to build custom kinematics using built in libraries like 'posemath'. Posemath provides many functions for efficient matrix manipulation and also offers functions for the use of quaternions. However the use of such a library would require a more indepth understanding of the mathematical theory that is beyond the scope of this presentation. Furthermore importing a library like 'posemath' into the 'userkins.comp' template would require substantially more programming skills than using the method applied in this paper.

[1]:
```
from IPython.display import display, Image
# only used to display equations out of code blocks
from IPython.display import Math, display

# joint position vector P
var('Px','Py','Pz')
P_=matrix([[Px],
           [Py],
           [Pz],
           [1]])

# cartesian position vector Q
var('Qx','Qy','Qz')
Q_=matrix([[Qx],
           [Qy],
           [Qz],
           [1]])
```

A custom kinematic model in LinuxCNC is used to calculate cartesian coordinates from given machine joint positions (forward kinematics) and also to calculate the required machine joint positions to reach a given coordinate position (inverse kinematics). In the following description we will use vectors as mathematical representations of the two positions:

$$Q = \begin{pmatrix} Qx \\ Qy \\ Qz \\ 1 \end{pmatrix} \textit{ Cartesian position} \qquad P = \begin{pmatrix} Px \\ Py \\ Pz \\ 1 \end{pmatrix} \textit{ Joint position} \tag{1}$$

Note that the fourth row is added to be able to multiply the vectors with a 4x4 transformation matrix.

# 2 TCP Kinematic model

For the tool to follow a point on the work piece we need a model that calculates where a given position on the work piece moves to when the rotary joints are rotated. In our example configuration the work piece will be mounted on therotary table and it is therefore here where we start to build our forward kinematic model. Note that in matrix multiplication the order is important, that is, $A \cdot B$ is generally not equal to $B \cdot A$.

## 2.1 Table Rotary

### 2.1.1 Forward transformation

We start with the work side rotation. In this case our forward transformation matrix ${}^{Q}A_{P}$ is equal to a rotation around the Z-axis:

$$ {}^{Q}A_{P} = R_{w} \tag{2}$$

[2]:
```
#define rotation matrix for work side rotation (ie the rotary table)
var('Cw','Sw')
Rw=matrix([[ Cw, -Sw, 0, 0],
           [ Sw,  Cw, 0, 0],
           [ 0,   0,  1, 0],
           [ 0,   0,  0, 1]])

display(Math(rf'R_w =~'+latex(Rw)))
```

$$R_{w} = \begin{pmatrix} Cw & -Sw & 0 & 0 \\ Sw & Cw & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

with $Sw = sin(\theta_{w}), \quad Cw = cos(\theta_{w})$ and $\theta_{w}$ being the angle of rotation of the table rotary

To derive the coordinate position $Q(Qx, Qy, Qz)$ we now need to multiply the joint position vector $P(Px, Py, Pz)$ with our forward transformation matrix ${}^{Q}A_{P}$. Note that the input values $P$ to our model need to be on the right hand side of the matrix multiplication.

$$Q = {}^{Q}A_{P} \cdot P$$

[3]:
```
# calculate forward kinematic
Q_out=Rw*P_
display(Math(rf'Q␣
 ↪=~'+latex(Q_)+rf'=~'+latex(Rw)+rf'\cdot'+latex(P_)+rf'=~'+latex(Q_out)))
```

$$Q = \begin{pmatrix} Qx \\ Qy \\ Qz \\ 1 \end{pmatrix} = \begin{pmatrix} Cw & -Sw & 0 & 0 \\ Sw & Cw & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Px \\ Py \\ Pz \\ 1 \end{pmatrix} = \begin{pmatrix} CwPx - PySw \\ CwPy + PxSw \\ Pz \\ 1 \end{pmatrix}$$

### 2.1.2 Inverse transformation

To calculate the joint position $P$ from given coordinate positions $Q$ we need to 'unrotate' the rotary table. Mathematically this means we need to transpose the rotation part in our transformation matrix.

$$ {}^{P}A_{Q} = R_{w}^{T} \tag{3}$$

To derive the joint position $P$ we then multiply the coordinate position vector $Q$ from the right:

$$ P = {}^{P}A_{Q} \cdot Q \tag{4}$$

[4]:
```
# calculate inverse kinematic
P_out=Rw.transpose()*Q_
display(Math(rf'P =~'+latex(P_)+rf'=~'+latex(Rw.
 ↪transpose())+rf'\cdot'+latex(Q_)+rf'=~'+latex(P_out)))
```

$$P = \begin{pmatrix} Px \\ Py \\ Pz \\ 1 \end{pmatrix} = \begin{pmatrix} Cw & Sw & 0 & 0 \\ -Sw & Cw & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Qx \\ Qy \\ Qz \\ 1 \end{pmatrix} = \begin{pmatrix} CwQx + QySw \\ CwQy - QxSw \\ Qz \\ 1 \end{pmatrix}$$

[5]:
```
# define joint position matrix
var('Px','Py','Pz')
Tp=matrix([[ 1, 0, 0, Px ],
           [ 0, 1, 0, Py],
           [ 0, 0, 1, Pz],
           [ 0, 0, 0, 1 ]])
```

[6]:
```
# calculate the forward transformation matrix
qAp=Rw*Tp
display(Math(rf'^QA_P =~~'+latex(Rw)+rf'\cdot'+latex(Tp)))
display(Math(rf'^QA_P =~~'+latex(qAp)))
```

$${}^{Q}A_{P} = \begin{pmatrix} Cw & -Sw & 0 & 0 \\ Sw & Cw & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & Px \\ 0 & 1 & 0 & Py \\ 0 & 0 & 1 & Pz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$${}^{Q}A_{P} = \begin{pmatrix} Cw & -Sw & 0 & CwPx - PySw \\ Sw & Cw & 0 & CwPy + PxSw \\ 0 & 0 & 1 & Pz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## 2.2 A closer look at the spindle rotary assembly

To get a better understanding of these offsets in the spindle rotary assembly of our example machine we need to have a look at how the primary rotary axis and the secondary rotary axis are arranged

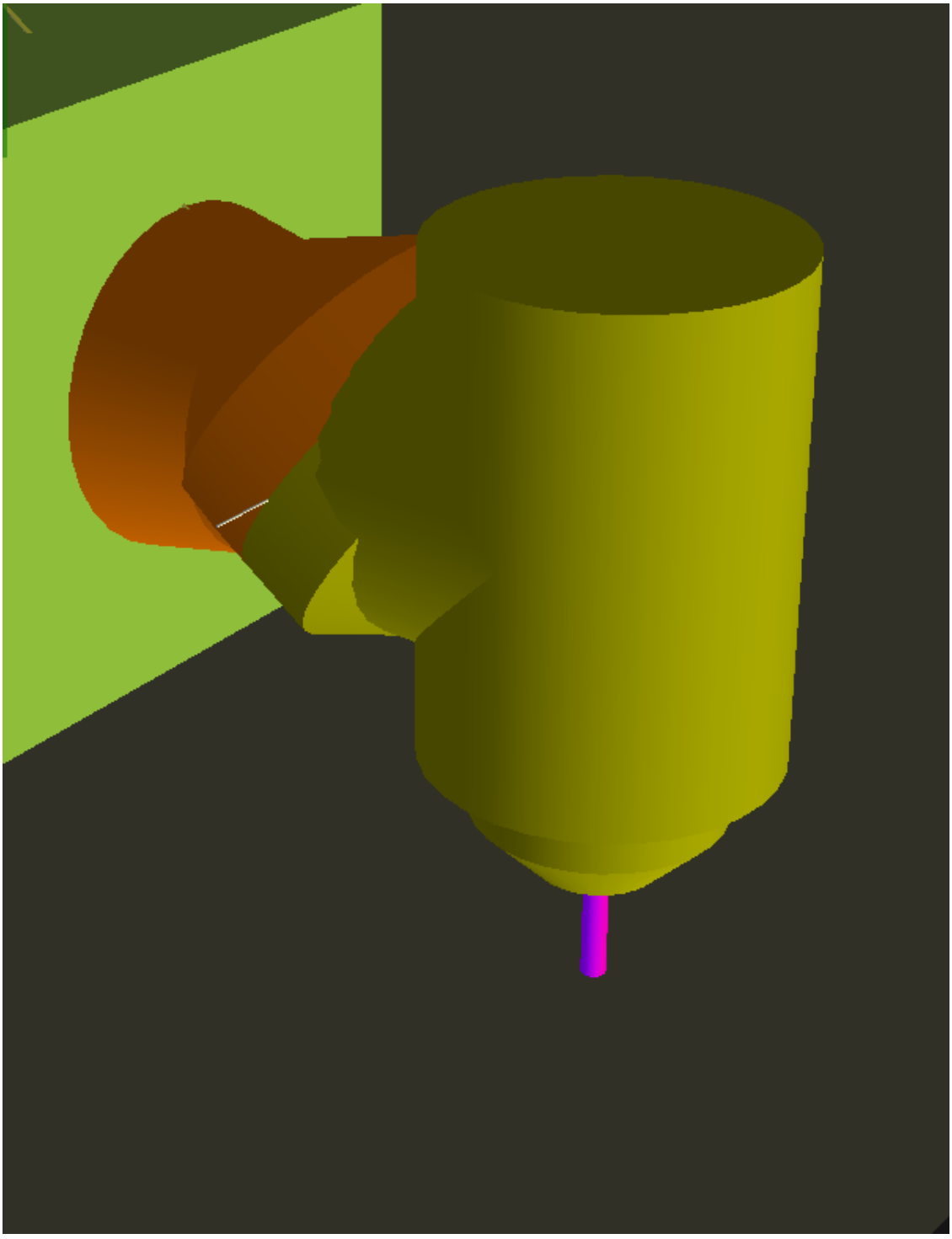
in the spindle body. The image below shows our spindle assembly in its home position ($\theta_p = 0$, $\theta_s = 0$). To get a clearer idea of the situation a tool (purple cylinder) is shown protruding from the spindle nose.

[ ]:

### 2.3 Pivot-point and pivot-length of the spindle head

Because the rotation axes and offsets in question are all hidden inside the spindle body we simplify the model to the relevant kinematic components. The white cross with the arrow pointing up shows the intersection of the rotary axes (yellow sphere). This point will be refered to as the 'pivot-point' of the spindle rotary assembly. The primary rotary axis (yellow line) extends vertically from the pivot-point in the machine-z direction. The secondary rotary axis (yellow line) extends from the pivot-point towards the rotational axis of the tool (purple cylinder) attaches. Note here that this axis will only intersect with the center of the spindle nose it the nutation angle happens to be 45°. The 'y-pivot-length' is the distance in the y-direction between the pivot-point and the spindle nose and shown in green. The 'z-pivot-length' is the distance in the z-direction between the pivot-point and the spindle nose and shown in blue. Note that in this image there is no geometric offset between the two rotary axes and they intersect at the pivot-point. We define the values for the offsets in the example image as *y-pivot = 50* and *z-pivot = 200*. In our kinematic model this represents the situation where, starting from the spindle nose, we need to travel 200 in the positive z-direction and 50 in the positive y-direction to reach the pivot-point. Note that the direction of travel when defining these offsets is arbitrary so in our case the offset situation in the image could also be defined as -50 in y and -200 in z. However once the definition is made we must keep it throughout the entire process of building the kinematic model.

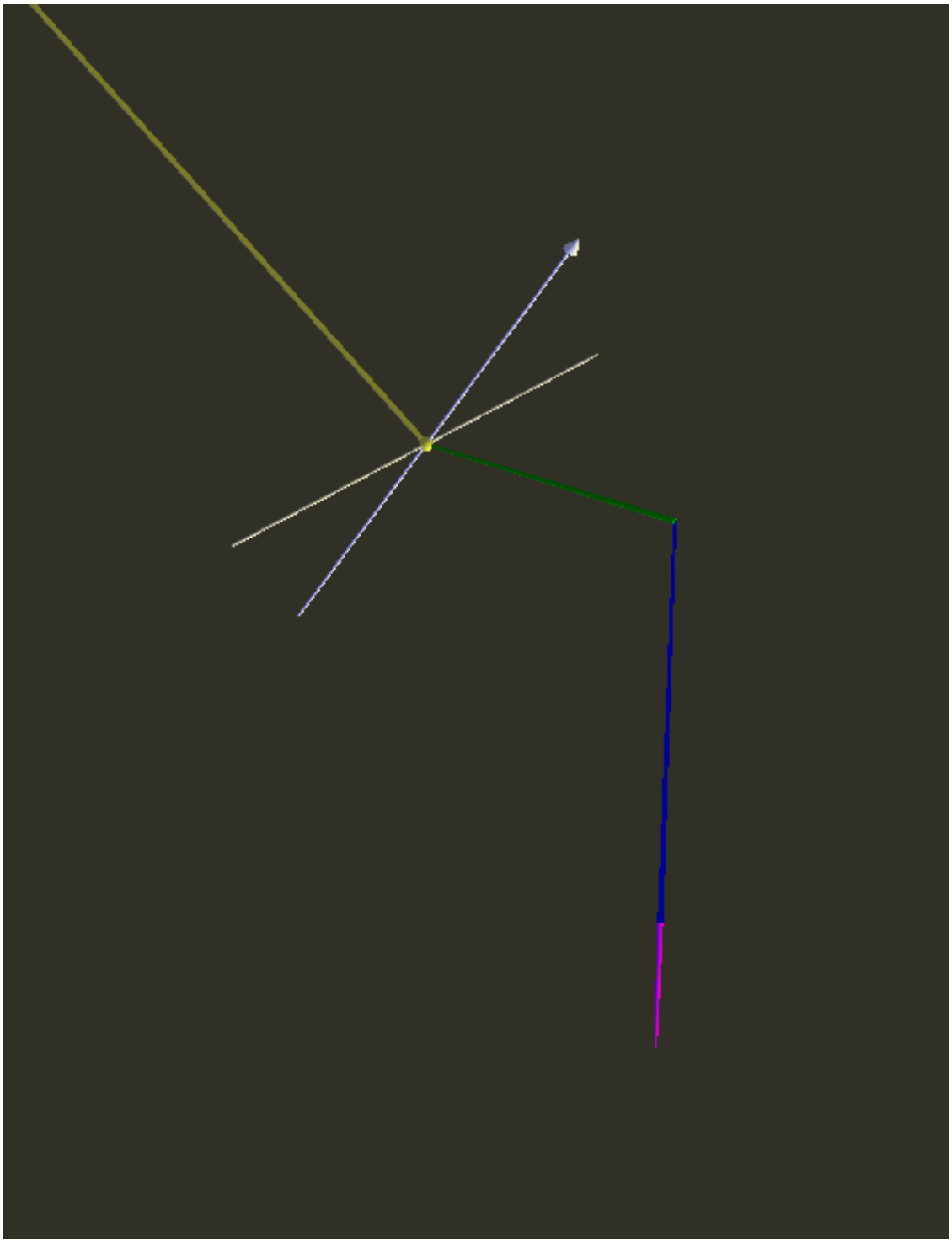
[7]:
```
display(Image(filename="Images/spindle_head_outside.png",
              height=300, width=300))
```



To be able to add these (y,z)-pivot values to our kinematic model we need to create a transformation matrix that has the effect of a translation. Note that the direction chosen for these offsets is important here as our translation is basically a vector and we have chosen that this 'pivot-length' vector points from the spindle nose to the pivot-point. Since we are building the forward kinematic

model we are travelling against this vector and thus its components $(L_y, L_z)$ need to be entered in the negative.

[8]:
```
display(Image(filename="Images/spindle_head_inside.png",
              height=300, width=300))
```



[9]:
```
# define translation matrix for the pivot lengths
var('Ly','Lz')
Tl=matrix([[ 1, 0, 0, 0 ],
           [ 0, 1, 0, -Ly],
           [ 0, 0, 1, -Lz],
           [ 0, 0, 0, 1 ]])
display(Math(rf'T_l =~'+latex(Tl)))
```

$$T_l = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -Ly \\ 0 & 0 & 1 & -Lz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

[ ]:
```
```

In this machine configuration we have no further geometrical offsets.

[10]:
```
# define translation matrix for the offsets
# no offsets so we use the identity matrix
Td=matrix([[ 1, 0, 0, 0],
           [ 0, 1, 0, 0],
           [ 0, 0, 1, 0],
           [ 0, 0, 0, 1 ]])
display(Math(rf'T_d =~'+latex(Td)))
```

$$T_d = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## 2.4 Spindle Rotary

### 2.4.1 How to derive a rotation matrix for a nutating joint

The machine in our example has a 'nutating' secondary rotary joint. Nutating meaning here that the rotational axis is not parallel to any of the axes (x, y or z) of our machine coordinate system and hence we cannot use any of the basic rotation matrices to model it in our kinematic. The basic rotation matrices are only three particular solutions of a general formula to describe rotations around any given vector $V(v_x, v_y, v_z)$ in space. In our particular case we will see that a regular (ie orthogonal) rotary assembly corresponds to a nutation angle of 90° and our kinematic model will also cover this particular machine type. To construct the rotation matrix $R_s$ for the slanted sencondary joint of our spindle we will use the 'Rodrigues' form of the general rotation formula. (Note that there is also an 'Euler-Rodrigues' formula that can be used for the same purpose but uses different parameters.)

$$R_s \; = \; I \; + \; sin\theta \cdot V + (2sin^2\frac{\theta}{2}) \cdot V^2 \tag{5}$$

where $\theta$ is the angle of positive rotation around a vector $\vec{v}$ acording to the 'Right-Hand' rule.

With:

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad V = \begin{pmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{pmatrix} \tag{6}$$

For the matrix $V$ we need a unit vector that represents the rotational axis of the nutating joint:

$$\vec{v} = (v_x, v_y, v_z)$$

In our case the nutating axis lies in the YZ-plane and has the components:

$$v_x = 0, \quad v_y = sin\nu, \quad v_z = cos\nu$$

where $\nu$ is the angle of the nutating axis as measured from the vertical.

Using the definitions of $I$ and $V$ we can rewrite \$R_s \$ as:

$$R_s = \begin{pmatrix} cos\theta + v_x^2(1-cos\theta) & v_xv_y(1-cos\theta) - v_zsin\theta & v_xv_z(1-cos\theta) + v_ysin\theta \\ v_xv_y(1-cos\theta) + v_zsin\theta & cos\theta + v_y^2(1-cos\theta) & v_yv_z(1-cos\theta) - v_xsin\theta \\ v_xv_z(1-cos\theta) - v_ysin\theta & v_yv_z(1-cos\theta) + v_xsin\theta & cos\theta + v_z^2(1-cos\theta) \end{pmatrix} \tag{7}$$

For simplicity we substitute

$$sin\nu = Sv, \quad cos\nu = Cv, \quad sin\theta = Ss, \quad cos\theta = Cs$$

using the components of $\vec{v}$ the rotation matrix of our secondary joint now becomes

$$R_s = \begin{pmatrix} Cs & -CvSs & SvSs \\ CvSs & Cs + S^2v(1-Cs) & SvCv(1-Cs) \\ -SvSs & SvCv(1-Cs) & Cs + C^2v(1-Cs) \end{pmatrix} \tag{8}$$

and by expanding it to a 4x4 matrix we have now the required transformation matrix

$$R_s = \begin{pmatrix} Cs & -CvSs & SvSs & 0 \\ CvSs & Cs + S^2v(1-Cs) & SvCv(1-Cs) & 0 \\ -SvSs & SvCv(1-Cs) & Cs + C^2v(1-Cs) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{9}$$

To make the matrix multiplications more manageable we substitute

$$Cs + S^2v(1-Cs) = r \qquad Cs + C^2v(1-Cs) = s \qquad SvCv(1-Cs) = t$$

and get the transformation matrix in it's final form

$$R_s = \begin{pmatrix} Cs & -CvSs & SvSs & 0 \\ CvSs & r & t & 0 \\ -SvSs & t & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{10}$$

[11]:
```
# define rotation matrix for the secondary spindle joint
var('Cs','CvSs','SvSs','r','s','t')
Rs=matrix([[ Cs  , -CvSs,  SvSs, 0],
           [ CvSs,     r,     t, 0],
           [-SvSs,     t,     s, 0],
           [    0,     0,     0, 1]])
```

### 2.5 Tool length offset (TLO)

As we could see above in our inspection of the spindle rotary assembly the tool length offset ($Dt$) has the same direction as the z-component of the pivot-offset ($Lz$) that is, a positive $Dt$ value points in the negative machine-z direction. So as a building block in our forward kinematic transformation the value of $Dt$ needs to be in the negative:

[12]:
```
# define translation matrix for the tool length
var('Dt')
Tt=matrix([[ 1, 0, 0,  0],
           [ 0, 1, 0,  0 ],
           [ 0, 0, 1, -Dt],
           [ 0, 0, 0, 1 ]])
display(Math(rf'T_t =~'+latex(Tt)))
```

$$T_t = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -Dt \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

[13]:
```
Tz=Tl*Tt
display(Math(rf'T_lt =~'+latex(Tl)+rf'\cdot'+latex(Tt)+rf'=~'+latex(Tz)))
```

$$T_l t = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -Ly \\ 0 & 0 & 1 & -Lz \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -Dt \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -Ly \\ 0 & 0 & 1 & -Dt - Lz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

### 2.6 Table and Spindle rotations with offsets

Now that we have all the building blocks we can put everything together:

$$^{Q}A_P = R_w \cdot T_p \cdot T_d \cdot R_s \cdot T_l \cdot T_t$$

[14]:
```
# calculate the forward transformation matrix
# note: the brackets *(Tl*Tt) are only used to get a cosmetically
# prettier result since it seems to keep "(Dt+Lz)"
qAp=Rw*Tp*Td*Rs*(Tl*Tt)
display(Math(rf'^Q A_P =~'+latex(qAp)))
```

$$ {}^{Q}A_{P} = \begin{pmatrix} CsCw - CvSsSw & -CvSsCw - Swr & CwSvSs - Swt & -(CwSvSs - Swt)(Dt + Lz) + (CvSsCw + Swr)Ly \\ CvSsCw + CsSw & -CvSsSw + Cwr & SvSsSw + Cwt & -(SvSsSw + Cwt)(Dt + Lz) + (CvSsSw - Cwr)Ly \\ -SvSs & t & s & -(Dt + \\ 0 & 0 & 0 & \end{pmatrix} $$

[15]:
```
# Extract the tool-position vector Q (FORWARD KINEMATICS) from
# the fourth column of the forward transformation matrix
Qx=qAp[0][3]
Qy=qAp[1][3]
Qz=qAp[2][3]

display(Math(latex(Q_[0][0]) + rf'~=~'  + latex(Qx)))
display(Math(latex(Q_[1][0]) + rf'~=~'  + latex(Qy)))
display(Math(latex(Q_[2][0]) + rf'~=~'  + latex(Qz)))
```

$$ Qx ~=~ -(CwSvSs - Swt)(Dt + Lz) + (CvSsCw + Swr)Ly + CwPx - PySw $$

$$ Qy ~=~ -(SvSsSw + Cwt)(Dt + Lz) + (CvSsSw - Cwr)Ly + CwPy + PxSw $$

$$ Qz ~=~ -(Dt + Lz)s - Lyt + Pz $$

So with this as our core forward transformation matrix we can now check the output of this model for a nutation angle $\nu = 45°$ with the spindle assembly in the home position:

$$ P(Px, Py, Pz) = (0, 0, 0) $$

$$ \theta_w = \theta_p = \theta_s = 0 \quad => \quad (Cw = Cp = Cs = 1, \quad Sw = Sp = Ss = 0) $$

taking our substitution in $R_s$ into account:

$$ Cs + S^2\nu(1 - Cs) ~=~ r \qquad Cs + C^2\nu(1 - Cs) ~=~ s \qquad S\nu C\nu(1 - Cs) ~=~ t $$

for our chosen nutation angle of $\nu = 45°$ this becomes

$$ Cs ~=~ r \qquad Cs ~=~ s \qquad S_\nu ~=~ t $$

~ which for $\theta_a = 0$ reduces to:

$$ r = 1 \qquad s ~=~ 1 \qquad t ~=~ 0 $$

Thus we get the coordinates:

$$ Qx = 0 $$

$$Qy = -Ly)$$
$$Qz = -(Dt + Lz)$$

$$Q = \begin{pmatrix} Qx \\ Qy \\ Qz \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -Ly \\ -Dt - Lz \\ 1 \end{pmatrix} \tag{11}$$

Since we started out with all joints in the home position we would expect the resulting coordinate position to be $Q(Qx, Qy, Qz) = (0, 0, 0)$. Hence we need to compensate for the offsets in our resulting coordinate position by subtracting them from the result of our core transformation matrix. Note subtracting a vector is the same as adding its inverse vector We define the transformation:

[16]:
```
Tc=matrix([[ 1, 0, 0,   0 ],
           [ 0, 1, 0,   Ly],
           [ 0, 0, 1, (Dt+Lz)],
           [ 0, 0, 0,   1 ]])
display(Math(rf'T_c =~'+latex(Tc)))
```

$$T_c = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & Ly \\ 0 & 0 & 1 & Dt + Lz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note that this is a translation of the result of our forward transformation matrix ${}^{Q}A_P$ it needs to be multiplied from the left:

$${}^{Q}A_P = T_c \cdot R_w \cdot T_p \cdot T_d \cdot R_s \cdot T_l \cdot T_t$$

[17]:
```
# calculate the forward transformation matrix
# note: the brackets *(Tl*Tt) are only used to get a cosmetically
# prettier result since it seems to keep "(Dt+Lz)"
qAp=Tc*Rw*Tp*Td*Rs*(Tl*Tt)
display(Math(rf'^Q A_P =~'+latex(qAp)))
```

$${}^{Q}A_P = \begin{pmatrix} CsCw - CvSsSw & -CvSsCw - Swr & CwSvSs - Swt & -(CwSvSs - Swt)(Dt + Lz) + (CvSsCw + S \\ CvSsCw + CsSw & -CvSsSw + Cwr & SvSsSw + Cwt & -(SvSsSw + Cwt)(Dt + Lz) + (CvSsSw - Cwr)Ly \\ -SvSs & t & s & -(Dt + Lz)s \\ 0 & 0 & 0 & \end{pmatrix}$$

[18]:
```
# Extract the tool-position vector Q (FORWARD KINEMATICS) from
# the fourth column of the forward transformation matrix
Qx=qAp[0][3]
Qy=qAp[1][3]
Qz=qAp[2][3]

display(Math(latex(Q_[0][0]) + rf'~=~'  + latex(Qx)))
```

```
display(Math(latex(Q_[1][0]) + rf'~=~'  + latex(Qy)))
display(Math(latex(Q_[2][0]) + rf'~=~'  + latex(Qz)))
```

$$Qx = -(CwSvSs - Swt)(Dt + Lz) + (CvSsCw + Swr)Ly + CwPx - PySw$$

$$Qy = -(SvSsSw + Cwt)(Dt + Lz) + (CvSsSw - Cwr)Ly + CwPy + PxSw + Ly$$

$$Qz = -(Dt + Lz)s - Lyt + Dt + Lz + Pz$$

---

### 2.7 Position offset of the rotary table to the machine reference point

Up to this point we have assumed that the machine reference point coincides with the rotational axis of our rotary table. However, it is somewhat unlikely that a machines home position is exactly in the rotational axis of the rotary table and this will need to be taken into account in the kinematic model. Let us assume that we have a setup with no TLO ($Dt = 0$), no geometric offset ($Dx = Dy = 0$), no pivot-length ($Ly = Lz = 0$) and no offset of the rotation-axis of the rotary table to the machine reference point. A tool positioned on the rotational axis of the rotary table would have a joint-position of $P(0, 0, P_z)$ and that would give the expected resulting coordinate position of $Q(0, 0, P_z)$. If we now assume that the rotation-axis of our rotary table is offset from the machine reference point by $(ra_y, ra_z)$ then our joint-position would be equal to $P(0, ra_y, ra_z)$ which would give us a result of $Q(0, ra_y, ra_z)$. So we need to subtract the offset $(ra_y, ra_z)$ from the joint-position $P = (P_x, P_y - ra_y, P_z - ra_z)$ before we input it into our transformation matrix. In other words we need to translate the joint-position vector in the opposite direction along the offset vector with the components $(0, ra_y, ra_z)$.

[19]:
```
var('Drax','Dray','Draz')
# define offset translation matrix for the rotary c table
Tr=matrix([[ 1, 0, 0, -Drax],
           [ 0, 1, 0, -Dray],
           [ 0, 0, 1,     0],
           [ 0, 0, 0,     1]])
display(Math(rf'Tr =~'+latex(Tr)))
```

$$Tr = \begin{pmatrix} 1 & 0 & 0 & -Drax \\ 0 & 1 & 0 & -Dray \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note that this is a translation $T_r$ of the input $T_p$ of our forward transformation matrix ${}^{Q}A_{P}$ so it needs to be multiplied to $T_p$ from the right :

[20]:
```
Tpr=Tp*Tr
display(Math(rf'T_p \cdot T_r␣
 ↪=~'+latex(Tp)+rf'\cdot'+latex(Tr)+rf'=~'+latex(Tpr)))
```

$$T_p \cdot T_r = \begin{pmatrix} 1 & 0 & 0 & Px \\ 0 & 1 & 0 & Py \\ 0 & 0 & 1 & Pz \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -Drax \\ 0 & 1 & 0 & -Dray \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -Drax + Px \\ 0 & 1 & 0 & -Dray + Py \\ 0 & 0 & 1 & Pz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Thus we get:

$$^{Q}A_{P} = T_{c} \cdot R_{w} \cdot T_{p} \cdot T_{r} \cdot R_{p} \cdot T_{d} \cdot R_{s} \cdot T_{l} \cdot T_{t}$$

However some more consideration is needed for now a joint position of $P(0, ra_{y}, ra_{z})$ will result in a coordinate position of $Q(0, 0, 0)$ which is of course not the value we can hand back to LinuxCNC because if the rotation axis is offset from machine home position then the coordinate position would need to be $Q(0, ra_{y}, ra_{z})$. So to be consistent we need to add the offset values back to the results of our calculations which we do again by multiplying a vector translation to the left of our forward kinematic:

[21]:
```
Tnr=matrix([[ 1, 0, 0, Drax],
            [ 0, 1, 0, Dray],
            [ 0, 0, 1,    0],
            [ 0, 0, 0,    1]])
display(Math(rf'T._r =~'+latex(Tnr)))
```

$$T._{r} = \begin{pmatrix} 1 & 0 & 0 & Drax \\ 0 & 1 & 0 & Dray \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Thus we get our final forward transformation matrix:

$$^{Q}A_{P} = T_{-r} \cdot T_{c} \cdot R_{w} \cdot T_{p} \cdot T_{r} \cdot R_{p} \cdot T_{d} \cdot R_{s} \cdot T_{l} \cdot T_{t}$$

[23]:
```
# calculate the forward transformation matrix
# note: the brackets *(Tl*Tt) are only used to get a cosmetically
# prettier result since it seems to keep "(Dt+Lz)"
# similar for (Tp*Tr)
qAp=Tnr*Tc*Rw*(Tp*Tr)*Td*Rs*(Tl*Tt)
display(Math(rf'^Q A_P =~'+latex(qAp)))
```

$$^{Q}A_{P} = \begin{pmatrix} CsCw - CvSsSw & -CvSsCw - Swr & CwSvSs - Swt & -Cw(Drax - Px) - (CwSvSs - Swt)(Dt + L \\ CvSsCw + CsSw & -CvSsSw + Cwr & SvSsSw + Cwt & -Cw(Dray - Py) - (SvSsSw + Cwt)(Dt + Lz) + ( \\ -SvSs & t & s & \\ 0 & 0 & 0 & \end{pmatrix}$$

[24]:
```
# Extract the tool-position vector Q (FORWARD KINEMATICS) from
# the fourth column of the forward transformation matrix
Qx=qAp[0][3]
Qy=qAp[1][3]
Qz=qAp[2][3]

display(Math(latex(Q_[0][0]) + rf'~=~'  + latex(Qx)))
display(Math(latex(Q_[1][0]) + rf'~=~'  + latex(Qy)))
display(Math(latex(Q_[2][0]) + rf'~=~'  + latex(Qz)))
```

$$Qx = -Cw(Drax - Px) - (CwSvSs - Swt)(Dt + Lz) + (CvSsCw + Swr)Ly + (Dray - Py)Sw + Drax$$

$$Qy = -Cw(Dray - Py) - (SvSsSw + Cwt)(Dt + Lz) + (CvSsSw - Cwr)Ly - (Drax - Px)Sw + Dray + Ly$$

$$Qz = -(Dt + Lz)s - Lyt + Dt + Lz + Pz$$

[25]:
```
# expressions as used in kinematics component
print('TCP kinematics FORWARD')
print('pos->tran.x = ', Qx,';')
print('pos->tran.y = ', Qy,';')
print('pos->tran.z = ', Qz,';')
```

```
TCP kinematics FORWARD
pos->tran.x =  -Cw*(Drax - Px) - (Cw*SvSs - Sw*t)*(Dt + Lz) + (CvSs*Cw +
Sw*r)*Ly + (Dray - Py)*Sw + Drax ;
pos->tran.y =  -Cw*(Dray - Py) - (SvSs*Sw + Cw*t)*(Dt + Lz) + (CvSs*Sw -
Cw*r)*Ly - (Drax - Px)*Sw + Dray + Ly ;
pos->tran.z =  -(Dt + Lz)*s - Ly*t + Dt + Lz + Pz ;
```

Notes: In this kinematic model the offset of the rotational axis of the rotary table is the vector $(Dra_x, 0, Dra_z)$ going from the pivot-point of the spindle assembly to the rotary-axis of the table rotary, with the machine in the home position. For this to be correct we need to compensate for any geometric offsets in the XZ plane $(L_z)$ of the spindle assembly. In other words if the rotary-axis has the absolute machine coordinates $(0, Rot_y, Rot_z)$ then

$$Dra_y = Rot_y - L_y$$

$$Dra_z = Rot_z - Lz$$

In the kinematic comp this is handled in the variable declaration(see below).

The values for 'Cv', 'Sv', 'Ly', 'Lz', 'Dray', 'Draz' are constants for a given machine (unless the rotary table is removable).

The values for 'Cw', 'Sw', 'Cp', 'Sp', 'Cs', 'Ss' are recalculated from the rotary joint positions on each cycle and also stored in the respective variables in the component file. Likewise the values for the substitutions 'r', 's', 't'.

---

### 2.7.1 Inverse transformation

To calculate the joint position $P$ from a given coordinate position $Q$ we need to follow the kinematic chain in the opposite direction from the spindle to the work piece. We can invert our forward kinematic transformation and build it backwards, using the inverted rotations and the inverted translations. Note that the input matrix $T_p$ with its translation $\{T_p \cdot T_r\}$ for the offset of the rotation axis and the output translation $[T_{-r} \cdot T_c]$ need to be handled with some consideration:

$$^{Q}A_P = [T_{-r} \cdot T_c] \cdot R_w \cdot \{T_p \cdot T_r\} \cdot (T_d \cdot R_s \cdot T_l \cdot T_t)$$

$$^{Q}A_P = [offset_{fo}] \cdot R_w \cdot \{input + offset_{fi}\} \cdot (T_d \cdot R_s \cdot T_l \cdot T_t)$$

In the inverse transformation the input is $T_q$ and it's 'offset' is the inverse of the 'output offset' of the forward transformation. Basically what we added to the result of the forward transformation needs to be removed from the input to the inverse transformation and vice versa. On the right side we additionally need to rotate the inverted spindle offsets by the rotation of the rotary table:

$$^{P}A_{Q} = [-offset_{fi}] \cdot R_w^T \cdot \{input - offset_{fo}\} \cdot (R_w \cdot T_{-d} \cdot R_s \cdot T_{-l} \cdot T_{-t})$$

$$^{P}A_{Q} = \{T_{-r}\} \cdot R_w^T \cdot T_q \cdot [T_r \cdot T_{-c}] \cdot (R_w \cdot T_{-d} \cdot R_s \cdot T_{-l} \cdot T_{-t})$$

Note that the brackets in the above expressions are only used to highlight the different parts of the kinematic model and are not mathematically required.

This being the inverse transformation our input matrix has changed to $T_q$:

[26]:
```
# define coordinate position matrix
var('Qx','Qy','Qz')
Tq=matrix([[ 1, 0, 0, Qx ],
           [ 0, 1, 0, Qy],
           [ 0, 0, 1, Qz],
           [ 0, 0, 0, 1 ]])
display(Math(rf'T_q =~'+latex(Tq)))
```

$$T_q = \begin{pmatrix} 1 & 0 & 0 & Qx \\ 0 & 1 & 0 & Qy \\ 0 & 0 & 1 & Qz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

[27]:
```
#define inverse translation matrices
Tnc=matrix([[ 1, 0, 0,   0],
            [ 0, 1, 0, -Ly],
            [ 0, 0, 1, -(Dt+Lz)],
            [ 0, 0, 0,   1]])
display(Math(rf'T._c =~'+latex(Tnc)))

Tnt=matrix([[ 1, 0, 0, 0],
            [ 0, 1, 0, 0 ],
            [ 0, 0, 1, Dt],
            [ 0, 0, 0, 1 ]])
display(Math(rf'T._t =~'+latex(Tnt)))
Tnd=matrix([[ 1, 0, 0, 0],
            [ 0, 1, 0, 0],
            [ 0, 0, 1, 0],
            [ 0, 0, 0, 1]])
display(Math(rf'T._d =~'+latex(Tnd)))
Tnl=matrix([[ 1, 0, 0, 0 ],
            [ 0, 1, 0, Ly],
            [ 0, 0, 1, Lz],
```

```
            [ 0, 0, 0, 1 ]])
display(Math(rf'T._l =~'+latex(Tnl)))
```

$$T._c = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -Ly \\ 0 & 0 & 1 & -Dt - Lz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T._t = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & Dt \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T._d = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T._l = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & Ly \\ 0 & 0 & 1 & Lz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

[29]:
```
#qAp=Tnr*Tc*Rw*(Tp*Tr)*Rp*Td*Rs*(Tl*Tt)
pAq=Tnr*Rw.transpose()*(Tq*Tr*Tnc)*Rw*Tnd*Rs*(Tnl*Tnt)
```

[31]:
```
display(Math(latex(Tq)+rf'\cdot'+latex(Rw.transpose())+rf'\cdot'+latex(Rs.
↪transpose())+rf'\cdot'+latex(Tnl)))
display(Math(rf'^PA_Q=~'+latex(pAq)))
```

$$\begin{pmatrix} 1 & 0 & 0 & Qx \\ 0 & 1 & 0 & Qy \\ 0 & 0 & 1 & Qz \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Cw & Sw & 0 & 0 \\ -Sw & Cw & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Cs & CvSs & -SvSs & 0 \\ -CvSs & r & t & 0 \\ SvSs & t & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & Ly \\ 0 & 0 & 1 & Lz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$${}^{P}A_{Q} = \begin{pmatrix} \left(Cw^2 + Sw^2\right)Cs & -\left(Cw^2 + Sw^2\right)CvSs & \left(Cw^2 + Sw^2\right)SvSs & -\left(Cw^2 + Sw^2\right)CvSsLy + \left(Cw^2 + Sw^2\right)( \\ \left(Cw^2 + Sw^2\right)CvSs & \left(Cw^2 + Sw^2\right)r & \left(Cw^2 + Sw^2\right)t & \left(Cw^2 + Sw^2\right)Lyr + \left(Cw^2 + Sw \\ -SvSs & t & s & \\ 0 & 0 & 0 & \end{pmatrix}$$

[32]:
```
# Extract the joint-position vector P (INVERSE KINEMATICS) from
# the fourth column of the inverse transformation matrix
Px=pAq[0][3]
Py=pAq[1][3]
Pz=pAq[2][3]

display(Math(latex(P_[0][0]) + rf'~=~'  + latex(Px)))
display(Math(latex(P_[1][0]) + rf'~=~'  + latex(Py)))
display(Math(latex(P_[2][0]) + rf'~=~'  + latex(Pz)))
```

$$Px = -\left(Cw^2 + Sw^2\right) CvSsLy + \left(Cw^2 + Sw^2\right)(Dt + Lz)SvSs - Cw(Drax - Qx) - (Dray + Ly - Qy)Sw + Drax$$

$$Py = \left(Cw^2 + Sw^2\right) Lyr + \left(Cw^2 + Sw^2\right)(Dt + Lz)t - Cw(Dray + Ly - Qy) + (Drax - Qx)Sw + Dray$$

$$Pz = (Dt + Lz)s + Lyt - Dt - Lz + Qz$$

Note that

$$cos^2\theta + sin^2\theta = 1$$

[33]:
```
Px=Px.subs({(Cw^2 + Sw^2):1})
Py=Py.subs({(Cw^2 + Sw^2):1})
Pz=Pz.subs({(Cw^2 + Sw^2):1})
display(Math(latex(P_[0][0]) + rf'~=~'  + latex(Px)))
display(Math(latex(P_[1][0]) + rf'~=~'  + latex(Py)))
display(Math(latex(P_[2][0]) + rf'~=~'  + latex(Pz)))
```

$$Px = -Cw(Drax - Qx) - CvSsLy + (Dt + Lz)SvSs - (Dray + Ly - Qy)Sw + Drax$$

$$Py = -Cw(Dray + Ly - Qy) + (Drax - Qx)Sw + Lyr + (Dt + Lz)t + Dray$$

$$Pz = (Dt + Lz)s + Lyt - Dt - Lz + Qz$$

[34]:
```
# expressions as used in gen_xyzbca_trsrn.comp
print('TCP kinematics INVERSE')
print('j[0] = ', Px,';')
print('j[1] = ', Py,';')
print('j[2] = ', Pz,';')
```

```
TCP kinematics INVERSE
j[0] =  -Cw*(Drax - Qx) - CvSs*Ly + (Dt + Lz)*SvSs - (Dray + Ly - Qy)*Sw + Drax
;
j[1] =  -Cw*(Dray + Ly - Qy) + (Drax - Qx)*Sw + Ly*r + (Dt + Lz)*t + Dray ;
j[2] =  (Dt + Lz)*s + Ly*t - Dt - Lz + Qz ;
```

For notes on the calculation of the various variables in the kinematics component see note above.

This concludes the TCP kinematic

---

# 3 Tool Kinematic model

Many, if not most, applications for 5-axis milling do not require TCP kinematics where all five axes are moving simultaneously but only need the work piece to be oriented at certain angles to the tool in between 'conventional' three axis (x,y,z) milling operations. In these operations the tool is not reoriented while cutting the material. This is what is called '3+2' mode. In 3+2 mode the machine operator can use the familiar built in cycles the machine controller offers for 3d (x,y,z) use and does not necessarily require CAM/CAD software to machine a part. The ability to move the tool in a plane perpendicular to its rotational axis also allows the use of probes for job setup. Machines with work side rotation (eg the A rotary in our example, or the 'table-rotary-tilting' examples included in LinuxCNC simulation configs) can orient the work piece to the tool by simply rotating the work

to the required orientation as the tool always remains oriented along the machine z-axis. A drilling operation on such a machine will thus still only require the machine z-axis to be moved no matter how the part is oriented. In this case no special kinematic is required. Machines with tool side rotation like the one presented here with B and C spindle rotations however retain the directions of the unrotated machine coordinate system when moving in IDENTITY and TCP kinematics while the tool orientation changes in respect to the machine coordinate system. A drilling operation on such a machine will require the machine to move in a complex manner that may include all three (x,y,z) axes depending on how tool is oriented. Commercial 5-axis machine controllers offer built in functionality that allow the operator to define work plane orientation using Gcode commands like G68.2 or similar; automatically adjusting the kinematic model to the type of kinematic of the machine. To implement such a feature for our example machine using LinuxCNC we need to create another kind of kinematic model which we will call the TOOL kinematic.

### 3.0.1 Forward transformation

For the TOOL kinematic we start at the tool tip and follow the kinematic chain towards the work piece. Since the table rotation is not invovled in the tool orientation it will be omitted in the TOOL kinematic model.

Thus our forward transformation matrix ${}^{Q}A_{P}$ is :

$$ {}^{Q}A_{P} = R_{tc}^{T} \cdot T_{l} \cdot R_{s}^{T} \cdot T_{d} \cdot R_{w}^{T} \ \cdot (T_{p} \cdot T_{-d} \cdot T_{-l}) $$

Note that the brackets in the above expressions are only used to highlight the different parts of the kinematic model and are not mathematically required. $R_{tc}$ is a virtual rotation around the rotational axis of the tool. This is required because we need a way to define the orientation of the tool-x (and -y). Without this the tool-coordinate system would be 'fixed' to the spindle head orientation which we do certainly want for the tool-z orientation but not for the x,y-directions. This will be important when developing the 'Tilted Work Plane' features later.

Tool-length offset is applied automatically in LinuxCNC by subtracting the tool length value stored in the tool table from the z-axis coordinate position while the joint position remains unchanged. This built in compensation also works in our custom tool kinematic because the tool coordinate system is always aligned with the rotational axis of the tool.

[35]:
```
# add c rotation of tool coordinate system
var('Ctc','Stc')
Rtc=matrix([[ Ctc, -Stc,   0, 0],
            [ Stc,  Ctc,   0, 0],
            [ 0 ,  0 ,   1, 0],
            [ 0,   0 ,   0, 1]])
```

[36]:
```
# calculate the forward kinematic
# brackets are only used for cosmetically adjust the resulting formula
qAp=Rtc.transpose()*(Tl)*Rs.transpose()*Td*Rw.transpose()*(Tp*Tnd*Tnl)
display(Math(
             latex(Rtc.transpose())+rf'\cdot'
            +latex(Tl)             +rf'\cdot'
            +latex(Rs.transpose()) +rf'\cdot'
```

```
            +latex(Td)              +rf'\cdot'
            +latex(Rw.transpose()) +rf'\cdot'
            +latex(Tp)              +rf'\cdot'
            +latex(Tnd)             +rf'\cdot'
            +latex(Tnl)             +rf'\cdot'
        ))
display(Math(rf'^Q A_P =~'+latex(qAp)))
```

$$\begin{pmatrix} Ctc & Stc & 0 & 0 \\ -Stc & Ctc & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -Ly \\ 0 & 0 & 1 & -Lz \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Cs & CvSs & -SvSs & 0 \\ -CvSs & r & t & 0 \\ SvSs & t & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot$$
$$\begin{pmatrix} Cw & Sw & 0 & 0 \\ -Sw & Cw & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & Px \\ 0 & 1 & 0 & Py \\ 0 & 0 & 1 & Pz \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & Ly \\ 0 & 0 & 1 & Lz \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot$$

$${}^{Q}A_P = \begin{pmatrix} (CsCtc - CvSsStc)Cw - (CtcCvSs + Stcr)Sw & (CtcCvSs + Stcr)Cw + (CsCtc - CvSsStc)Sw & -Ctc \\ -(CtcCvSs + CsStc)Cw + (CvSsStc - Ctcr)Sw & -(CvSsStc - Ctcr)Cw - (CtcCvSs + CsStc)Sw & Stc \\ CwSvSs - Swt & SvSsSw + Cwt & \\ 0 & 0 & \end{pmatrix}$$

Now it is important to remember that we have a machine configuration with both tool and work side rotations. While the tool orientation is a transformation from the TOOL to the WORK coordinate system and thus needs to include both rotations the tool position is a transformation from the TOOL to the MACHINE coordinate system and thus does not include the work side (ie table) rotation. Since our inverse tool transformation matrix has been calculated with both rotations we need to extract the tranlation vector (ie rows 1,2,3 of the 4. column) at $\theta_W = 0$. Hence we substitute

$$C_w = 1; \quad S_w = 0$$

Note that this substitution we effectively change $R_w$ to the identity matrix (ie we neutralize the table rotation)

[37]:
```
# Extract the tool-position vector Q (FORWARD KINEMATICS) from
# the fourth column of the forward transformation matrix
Qx=qAp[0][3]
Qy=qAp[1][3]
Qz=qAp[2][3]

display(Math(latex(Q_[0][0]) + rf'~=~'  + latex(Qx)))
display(Math(latex(Q_[1][0]) + rf'~=~'  + latex(Qy)))
display(Math(latex(Q_[2][0]) + rf'~=~'  + latex(Qz)))
```

$$Qx = ((CtcCvSs + Stcr)Cw + (CsCtc - CvSsStc)Sw)(Ly + Py) - (CtcSvSs - Stct)(Lz + Pz) + ((CsCtc - CvSsStc)Cw - (CtcCvSs + Stcr)Sw)Px - LyStc$$

$$Qy = -((CvSsStc - Ctcr)Cw + (CtcCvSs + CsStc)Sw)(Ly + Py) - CtcLy + (StcSvSs + Ctct)(Lz + Pz) - ((CtcCvSs + CsStc)Cw - (CvSsStc - Ctcr)Sw)Px$$

$$Qz = (SvSsSw + Cwt)(Ly + Py) + (CwSvSs - Swt)Px + (Lz + Pz)s - Lz$$

[38]:
```
Qx=Qx.subs({Cw:1, Sw:0})
Qy=Qy.subs({Cw:1, Sw:0})
Qz=Qz.subs({Cw:1, Sw:0})
display(Math(latex(Q_[0][0]) + rf'~=~'  + latex(Qx)))
display(Math(latex(Q_[1][0]) + rf'~=~'  + latex(Qy)))
display(Math(latex(Q_[2][0]) + rf'~=~'  + latex(Qz)))
```

$$Qx = (CtcCvSs + Stcr)(Ly + Py) - (CtcSvSs - Stct)(Lz + Pz) + (CsCtc - CvSsStc)Px - LyStc$$

$$Qy = -(CvSsStc - Ctcr)(Ly + Py) - CtcLy + (StcSvSs + Ctct)(Lz + Pz) - (CtcCvSs + CsStc)Px$$

$$Qz = PxSvSs + (Lz + Pz)s + (Ly + Py)t - Lz$$

[39]:
```
# expressions as used in kinematics component
print('TOOL kinematics FORWARD')
print('pos->tran.x = ', Qx,';')
print('pos->tran.y = ', Qy,';')
print('pos->tran.z = ', Qz,';')
```

```
TOOL kinematics FORWARD
pos->tran.x =  (Ctc*CvSs + Stc*r)*(Ly + Py) - (Ctc*SvSs - Stc*t)*(Lz + Pz) +
(Cs*Ctc - CvSs*Stc)*Px - Ly*Stc ;
pos->tran.y =  -(CvSs*Stc - Ctc*r)*(Ly + Py) - Ctc*Ly + (Stc*SvSs + Ctc*t)*(Lz +
Pz) - (Ctc*CvSs + Cs*Stc)*Px ;
pos->tran.z =  Px*SvSs + (Lz + Pz)*s + (Ly + Py)*t - Lz ;
```

---

### 3.0.2 Inverse transformation

For the inverse kinematic we need to move from the machine coordinates to the tool coordinates starting with the inverted offset translations $(T_d \cdot T_l)$ for the offsets we used on the input of the forward kinematic $T_{-d} \cdot T_{-l}$:

$$^{Q}A_P = R_{tc}^T \cdot T_l \cdot R_s^T \cdot T_d \cdot R_w^T \cdot (T_p \cdot T_{-d} \cdot T_{-l})$$

$$^{P}A_Q = (T_d \cdot T_l) \cdot R_w \cdot T_{-d} \cdot R_s \cdot T_{-l} \cdot R_{tc} \cdot T_q$$

Note that the brackets in the above expressions are only used to highlight the different parts of the kinematic model and are not mathematically required.

[40]:
```
# calculate the inverse TOOL kinematic
# brackets are only used for cosmetically adjust the resulting formula
#qAp=Rtc.transpose()*(Tnl)*Rs.transpose()*Tnd*Rp.transpose()*(Tp*Td*Tl)
pAq=(Td*Tl)*Rw*Tnd*Rs*Tnl*Rtc*Tq

display(Math(
            latex(Td)             +rf'\cdot'
```

```
            +latex(Tl)              +rf'\cdot'
            +latex(Rw)              +rf'\cdot'
            +latex(Tnd)             +rf'\cdot'
            +latex(Rs)              +rf'\cdot'
            +latex(Tnl)             +rf'\cdot'
            +latex(Rtc)             +rf'\cdot'
            +latex(Tq)
        ))

display(Math(rf'^PA_Q =~'+latex(pAq)))
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -Ly \\ 0 & 0 & 1 & -Lz \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Cw & -Sw & 0 & 0 \\ Sw & Cw & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot$$
$$\begin{pmatrix} Cs & -CvSs & SvSs & 0 \\ CvSs & r & t & 0 \\ -SvSs & t & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & Ly \\ 0 & 0 & 1 & Lz \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Ctc & -Stc & 0 & 0 \\ Stc & Ctc & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & Qx \\ 0 & 1 & 0 & Qy \\ 0 & 0 & 1 & Qz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$^{P}A_{Q} = \begin{pmatrix} (CsCw - CvSsSw)Ctc - (CvSsCw + Swr)Stc & -(CvSsCw + Swr)Ctc - (CsCw - CvSsSw)Stc & CwSvSs \\ (CvSsCw + CsSw)Ctc - (CvSsSw - Cwr)Stc & -(CvSsSw - Cwr)Ctc - (CvSsCw + CsSw)Stc & SvSsSw \\ -CtcSvSs + Stct & StcSvSs + Ctct & \\ 0 & 0 & \end{pmatrix}$$

[41]:
```
# Extract the joint-position vector P (INVERSE KINEMATICS) from
# the fourth column of the inverse transformation matrix
Px=pAq[0][3]
Py=pAq[1][3]
Pz=pAq[2][3]

display(Math(latex(P_[0][0]) + rf'~=~'  + latex(Px)))
display(Math(latex(P_[1][0]) + rf'~=~'  + latex(Py)))
display(Math(latex(P_[2][0]) + rf'~=~'  + latex(Pz)))
```

$$Px = -(CvSsCw + Swr)Ly + (CwSvSs - Swt)Lz + ((CsCw - CvSsSw)Ctc - (CvSsCw + Swr)Stc)Qx - ((CvSsCw + Swr)Ctc + (CsCw - CvSsSw)Stc)Qy + (CwSvSs - Swt)Qz$$

$$Py = -(CvSsSw - Cwr)Ly + (SvSsSw + Cwt)Lz + ((CvSsCw + CsSw)Ctc - (CvSsSw - Cwr)Stc)Qx - ((CvSsSw - Cwr)Ctc + (CvSsCw + CsSw)Stc)Qy + (SvSsSw + Cwt)Qz - Ly$$

$$Pz = -(CtcSvSs - Stct)Qx + (StcSvSs + Ctct)Qy + Lzs + Qzs + Lyt - Lz$$

Again, it is important to remember that we have a machine configuration with both tool and work side rotations. While the tool orientation is a transformation from the TOOL to the WORK coordinate system and thus needs to include both rotations the tool position is a transformation from the TOOL to the MACHINE coordinate system and thus does not include the work side (ie table) rotation. Since our inverse tool transformation matrix has been calculated with both rotations we need to extract the tranlation vector (ie rows 1,2,3 of the 4. column) at $\theta_W = 0$.

Hence we substitute

$$C_w = 1; \quad S_w = 0$$

Note that this substitution we effectively change $R_w$ to the identity matrix (ie we neutralize the table rotation)

[42]:
```
Px=Px.subs({Cw:1, Sw:0})
Py=Py.subs({Cw:1, Sw:0})
Pz=Pz.subs({Cw:1, Sw:0})
display(Math(latex(P_[0][0]) + rf'~=~'  + latex(Px)))
display(Math(latex(P_[1][0]) + rf'~=~'  + latex(Py)))
display(Math(latex(P_[2][0]) + rf'~=~'  + latex(Pz)))
```

$$Px = -CvSsLy + (CsCtc - CvSsStc)Qx - (CtcCvSs + CsStc)Qy + LzSvSs + QzSvSs$$

$$Py = (CtcCvSs + Stcr)Qx - (CvSsStc - Ctcr)Qy + Lyr + Lzt + Qzt - Ly$$

$$Pz = -(CtcSvSs - Stct)Qx + (StcSvSs + Ctct)Qy + Lzs + Qzs + Lyt - Lz$$

[43]:
```
# expressions as used in the kinematics component
print('TOOL kinematics INVERSE')
print('j[0] = ', Px,';')
print('j[1] = ', Py,';')
print('j[2] = ', Pz,';')
```

```
TOOL kinematics INVERSE
j[0] =  -CvSs*Ly + (Cs*Ctc - CvSs*Stc)*Qx - (Ctc*CvSs + Cs*Stc)*Qy + Lz*SvSs +
Qz*SvSs ;
j[1] =  (Ctc*CvSs + Stc*r)*Qx - (CvSs*Stc - Ctc*r)*Qy + Ly*r + Lz*t + Qz*t - Ly
;
j[2] =  -(Ctc*SvSs - Stc*t)*Qx + (Stc*SvSs + Ctc*t)*Qy + Lz*s + Qz*s + Ly*t - Lz
;
```

This completes the TOOL kinematic

### 3.0.3 Caveats of TOOL kinematics (TOOL kinematics is NOT TCP!)

Any machine operator using TOOL kinematics should be well aware of its behavior before issuing motion commands! It must be stressed that, while TOOL kinematics are active, any movement of a rotary joint will cause the machine to rotate the TCS around the machine reference point. Because the magnitude of the resulting movement increases with the distance between the tool position and the machine reference point the effect may seem unpredictable to the inexperienced operator. Note here that it is possible to set the rotation point in TOOL kinematics in much the same way as we offset the rotation-point from machine zero in TCP kinematics. This however would further complicate work offset transformation and there seems to be no immediate benefit to it.

---

## 4 'Tilted Work Plane' (TWP)

Commercial 5-axis machine controllers offer built in functionality that allow the operator to define work plane orientation using Gcode commands like G68.2 or similar. With the new custom TOOL

kinematic we can now explore a possible implementation of TWP in LinuxCNC. For the purpose of this example we choose the already mentioned 'G68.2' and the associated Gcodes as used by Fanuc:

- G68.2 - Set custom coordinate system using five different modes
- G68.3 - Set custom coordinate system using current tool orientation
- G68.4 - Same as 68.2 but as an incremental reorientation of an existing TWP
- G53.1 - Orient the tool to the TWP using non-TCP joint rotation
- G53.6 - Orient the tool to the TWP using TCP joint rotation
- G69 - Cancel the TWP setting

The TWP feature always requires a TWP-definition (G68.x) before an orientation command (G53.x) can be issued. G68.x commands do not cause any machine movement while G53 command will cause immediate machine movement. At the end of a TWP operation a G69 command is used to return to machine coordinates.

In this presentation we use a 'Pure Python Remap' to make the above Gcode commands available to the machine operator. As a consequence our depth of integration is quite limited in the sense that the LinuxCNC Gcode interpreter will be totally ignorant of our 'TWP'-mode.

### 4.0.1 Implementation outline

This should give a rough overview of how TWP functionality is to be achieved with a python remap:

The idea of TWP is to give the machine operator a mechanism to define a virtual work plane that is rotated (and optionally offset) in respect to the current machine coordinate system. As mentioned above this is fairly trivial in a machine with work side rotation as its tool spindle is fixed to be aligned with the machine z-axis. In such a machine the tool coordinate system (TCS) remains in the orientation of the machine coordinate system (MCS) In a machine with tool side rotation however the TCS rotates in respect to the MCS as the rotary joints move. The orientation of the TCS for a given joint rotation can be calculated using the rotation matrix embedded in the transformation matrix as derived above. Furthermore we can also solve for the inverse case of finding the joint positions that will orient the tool to a given orientation. Note here that this is very much dependent on the specific kinematic of the machine at hand and will require careful analysis of the kinematic model to find any ambiguities in the solution as some specific tool orientations might be achieved in multiple joint positions.

So to define the TWP means to define a target TCS in respect to the MCS and for this the use of a transformation matrix is well suited as it contains the rotation as well as the (optional) translation.

### 4.0.2 Step 1: Definition of the TWP

The TWP is defined by a rotation (I,J,K) and optional offset (X,Y,Z) of the TCS in respect to the MCS. The optional offset is calulated from the current work offset position at the time when the G68.2 or G68.3 command is issued.

The operator can choose from different methods to define the rotation of the custom work plane:

- G68.2 … P0 … - 'True Euler'-angles (this is the default if the P word is omitted)

- G68.2 … P1 … - 'Pitch-Roll-Yaw'-angles
- G68.2 … P2 … - 3 points in the plane (3 points define two vectors)
- G68.2 … P3 … - 2 vectors (tool-x and tool-z)
- G68.2 … P4 … - projected angles (A around machine-x, B around machine-y)

For the above methods we create the transformation matrix either by using euler-rotations of the 4x4 identity matrix or by the vector information passed by the command, using the cross product to calculate the y-vector. For the projected angles we use basic trigonometry to calculate the vector components. These calculations are independent from the specific machine kinematics.

- G68.3 …… - define TWP perpendicular to the current tool orientation

Here we need to use the specific machine kinematic model to calculate the transformation matrix of the current TCS.

- G68.4 …… - same as G68.2 but relative to the current TWP

Here we need to multiply the current TWP transformation matrix with the one built from the command words.

- G69 - cancel TWP

Here we reset the TWP transformation matrix to the 4x4 identity matrix (ie no rotation, no translation).

Note that, contrary to some other controllers, moves commanded between G68.x and G53.x will be in the MCS.

#### 4.0.3 Step 2: Orientation of the tool to the TWP

Once the TWP has been defined the machine spindle can be oriented by moving the rotary joints to the appropriate positions.

The operator can choose whether the rotation is done in IDENTITY kinematics (only the rotary joints move) or in TCP kinematics where all joints may move to keep the tool center point in position:

- G53.1 - orient the tool to the TWP without using TCP
- G53.6 - orient the tool to the TWP with using TCP

#### 4.0.4 Handling work offsets when switching to TWP

The TWP is defined relative to the work offset (eg G54) active when issuing a G62.x command. With the execution of the following G53.x command and the reorientation of the spindle the kinematic is switched to the TOOL kinematics and the coordinate position in the DRO will then reflect the coordinate position in the rotated TCS. If the work offset values were to be used without adjusting for the kinematic switch then the physical position of the work offset would be rotated out of place and the reference to the work coordinate system (WCS) would be lost. So in order to find the reference on the work piece in the TCS we need to use the TWP transformation matrix to transform the work offset values set in WCS to the new TCS. Note that the original work offset will have to be restored when the TWP is canceled with a G69 command.

#### 4.0.5 Handling the optional TWP offsets when switching to TWP

Since the optional offset of the origin of the TWP are to be applied before rotating the TWP we will need to either add these offsets to the current WCS before transforming the WCS or transform the WCS and the TWP offsets separatly and add them after.

---

### 4.1 Calculating the spindle and table rotary joint position for a given tool-orientation vector

The transformation matrix we derived for the inverse tool kinematic contains a rotation matrix that describes the rotation of the unit vectors of the machine coordinate system (MCS)

$$U = \begin{pmatrix} 1,0,0 \\ 0,1,0 \\ 0,0,1 \end{pmatrix} \; Unitvectors\ of\ the\ MCS \tag{12}$$

to the tool coordinate system (TCS).

$$K = \begin{pmatrix} Kx_x, Ky_x, Kz_x \\ Kx_y, Ky_y, Kz_y \\ Kx_z, Ky_z, Kz_z \end{pmatrix} \; Toolvector \tag{13}$$

To orient the tool perpendicular to the TWP requested by the user we need to find the angles of the spindle rotary joints so that the resulting tool-z vector matches the twp-z vector.

First we extract the tool-z vector from the third column of the transformation matrix in the inverse tool kinematic

[44]:
```
# Tool-z orientation vector K
var('Kz_x','Kz_y','Kz_z')
Kz_=matrix([[Kz_x],
            [Kz_y],
            [Kz_z],
            [1]])
# Tool-x orientation vector J
var('Kx_x','Kx_y','Kx_z')
Kx_=matrix([[Kx_x],
            [Kx_y],
            [Kx_z],
            [1]])
```

[45]:
```
# Extract the tool-z orientation vector from
# the third column of the inverse transformation matrix
Kzx=pAq[0][2]
Kzy=pAq[1][2]
Kzz=pAq[2][2]
```

```
display(Math(latex(Kz_[0][0]) + rf'~=~'  + latex(Kzx)))
display(Math(latex(Kz_[1][0]) + rf'~=~'  + latex(Kzy)))
display(Math(latex(Kz_[2][0]) + rf'~=~'  + latex(Kzz)))
```

$$Kz_x = CwSvSs - Swt$$

$$Kz_y = SvSsSw + Cwt$$

$$Kz_z = s$$

### 4.2 Calculating the spindle angle $\theta_s$

Using the z-component of the tool-z-vector in the 3x3 rotation matrix of the inverse TOOL transformation matrix:

$$Kz_z = s$$

With our substitutions:

$$Cs + S^2v(1 - Cs) = r \qquad Cs + C^2v(1 - Cs) = s \qquad SvCv(1 - Cs) = t$$

we get

$$Kz_z = C_s + C_\nu^2(1 - C_s)$$
$$Kz_z = C_\nu^2 + C_s - C_\nu^2 C_s$$
$$Kz_z = C_\nu^2 + C_s(1 - C_\nu^2)$$

$$C_s = \frac{Kz_z - C_\nu^2}{1 - C_\nu^2} \quad ; \theta_\nu \neq 0$$

Note that for $\theta_\nu = 0$ the primary rotary axis is parallel to the secondary rotary axis which is of no interest in our case.

$$\theta_S = \pm acos(\frac{Kz_z - C_\nu^2}{1 - C_\nu^2})$$

Note that the formula above is undefined for

$$|\frac{Kz_z - C_\nu^2}{1 - C_\nu^2}| > 1$$

First we check the positive case

$$\frac{Kz_z - C_\nu^2}{1 - C_\nu^2} > 1$$
$$Kz_z - C_\nu^2 > 1 - C_\nu^2$$

We get for the upper bound

$$Kz_z > 1$$

This case will never occur since we use a normalized vector.

Then we check the negative case

$$\frac{Kz_z - C_\nu^2}{1 - C_\nu^2} < -1$$

$$Kz_z - C_\nu^2 < C_\nu^2 - 1$$

We get for the lower bound

$$Kz_z < 2C_\nu^2 - 1$$

Which tells us that the lower bound for our $Kz_z$ value depends on the nutation angle $\theta_\nu$. For $\theta_\nu = 90°$ the above collapses to

$$Kz_z < -1$$

Which will also never occur for a normalized vector.

For $\theta_\nu = 0°$ the above collapses to

$$Kz_z < 1$$

Which is always true. Note that this is the case when the primary rotary axis is parallel to the secondary and it is intuitive that we cannot rotate the $Kz$ vector out of the vertical at all. In other words we can rotate our tool to a maximum of $2\theta_\nu$ out of the vertical position.

### 4.3 Calculating the table angle $\theta_w$

Solving

$$Kz_y = S_\nu S_s S_w + C_w t$$

for $C_w$ we get

$$C_w = \frac{Kz_y - S_w S_\nu S_s}{t} \quad ; t = SvCv(1 - Cs) \neq 0 \rightarrow \theta_\nu \neq 0°, 90°,\ \theta_S \neq 0°$$

The cases for $\theta_\nu \neq 0°, 90°$ have been discussed above. For $\theta_S \neq 0°$ the tool is vertical and thus we have infinite rotary table angles. In practice we will set it to 0°.

Using this result in

$$Kz_x = C_w S_\nu S_s - S_p t$$

we get

$$Kz_x = \frac{Kz_y - S_w S_\nu S_s}{t} S_\nu S_s - S_w t$$

Here we substitute

$$p = S_\nu S_s$$

and thus

$$Kz_x = \frac{Kz_y - S_w p}{t} p - S_w t$$

which is

$$Kz_x = \frac{pKz_y - p^2 S_w - t^2 S_w}{t}$$

and thus

$$tKz_x - pKz_y = S_w(-(t^2 + p^2))$$

for the angle of primary joint:

$$\theta_W = asin(\frac{tKz_x - pKz_y}{-(t^2 + p^2)})$$

and finally

$$\theta_W = asin(\frac{pKz_y - tKz_x}{t^2 + p^2})$$

Note that since we use the $asin()$ function we really have two relevant angles $~[ _W, ~~ - _W] $.

---

### 4.4 Calculating the angle $\theta_{tc}$

To set the direction of the tool-x and thus tool-y orientation we added a virtual rotation around the tool-z vector to our TOOL kinematic model. Without this we would not be able to define the orientation of, say, a rectangluar pocket in the TWP plane requested by the user, instead the orientation would simply follow the rotation of the x-unit-vector of the machine coordinate system as the spindle rotary assembly orients the tool to the TWP.

Our virtual rotation $R_{tc}$ is at the start of our kinematic chain and comes as an additional rotation to the rotations of the spindle rotary assembly (ie $R_s$ and $R_p$). In order to rotate the tool-x vector to a given orientation in the TWP plane we will first need to calculate the orientation of the tool-x vector that results from orienting the spindle to the requested TWP plane and then calculate the value for $\theta_{tc}$ required to rotate the tool-x vector to the requested orientation.

As shown above, the rotation of the x-tool vector is precisely described in the first column of the transformation matrix in the inverse TOOL kinematic. Setting our correctional rotation $\theta_{tc} = 0$ and using the joint angles $\theta_S, \theta_P$ calculated above to orient the tool to the requestd TWP the orientation of the tool-x vector can thus be calculated:

[46]:
```
# Extract the the tool-x orientation vector from
# the first column of the inverse transformation matrix

Kx_x=pAq[0][0]
Kx_y=pAq[1][0]
Kx_z=pAq[2][0]

display(Math(latex(Kx_[0][0]) + rf'~=~'  + latex(Kx_x)))
display(Math(latex(Kx_[1][0]) + rf'~=~'  + latex(Kx_y)))
display(Math(latex(Kx_[2][0]) + rf'~=~'  + latex(Kx_z)))
```

$$Kx_x = (CsCw - CvSsSw)Ctc - (CvSsCw + Swr)Stc$$

$$Kx_y = (CvSsCw + CsSw)Ctc - (CvSsSw - Cwr)Stc$$

$$Kx_z \;=\; -CtcSvSs + Stct$$

### 4.4.1 Orient the tool-x vector to lie in a plane parallel to the XY-plane of the machine coordinate system.

Since the tool-x vector, should lie in the xy-plane of the of the MCS the z-component of the tool-x vector will need to be equal to zero. Hence we extract the z-component from the tool-x vector and equal that to zero. Then we can solve this equation for $\theta_{tc}$ which is the only undefined variable left.

Whis is

$$Kx_z \;=\; -C_{tc}S_\nu S_s + tS_{tc} \;=\; 0$$

or

$$C_{tc}S_\nu S_s \;=\; tS_{tc}$$

We solve for the angle $\theta_{tc}$

$$\frac{S_\nu S_s}{t} \;=\; \frac{S_{tc}}{C_{tc}} \;=\; tan(\theta_{tc})$$

Resubstitution

$$S_\nu C_\nu (1 - C_s) \;=\; t$$

is:

$$\theta_{tc} = atan\frac{S_\nu S_s}{S_\nu C_\nu (1 - C_s)}$$

### 4.4.2 Orient the tool-x vector to a given vector

Using the G62.2 commands the operator can request the tool-x orientation in form of a vector given as a command argument:

$$\vec{L} \;=\; \begin{pmatrix} Lx_x \\ Lx_y \\ Lx_z \end{pmatrix} \; Tool - X - vector\ requested \tag{14}$$

To find the required value for $\theta_{tc}$ we first calculate the tool-x orientation using the formulas extracted above. Setting $\theta_{tc} = 0$ and using the joint angles $\theta_A, \theta_B$ calculated above to orient the tool to the requestd TWP we get the, as yet uncorrected, tool-x vector:

$$\vec{K} \;=\; \begin{pmatrix} Kx_x \\ Kx_y \\ Kx_z \end{pmatrix} \; Tool - X - vector\ uncorrected \tag{15}$$

The angle between the two vectors can be calculated using the dot product

$$\theta_{tc} \;=\; acos\frac{\vec{L} \;\cdot\; \vec{K}}{|\vec{L}||\vec{K}|}$$

since both our vectors are unit vectors this reduces to

$$\theta_{tc} \;=\; acos(\vec{L} \;\cdot\; \vec{K})$$