HERMLE FW801: 10/5/11

5-AXIS MACHINE TOOLS:
Kinematics and Vismach Implementation in LinuxCNC

Rudy du Preez
SA-CNC-CLUB

April 7, 2016

1   INTRODUCTION

Coordinated multi-axis CNC machine tools controlled with LINUXCNC, requires a special kinematics component for each type of machine. This document describes some of the most popular 5-axis machine configurations and then develops the forward (from work to joint coordinates) and inverse (from joint to work) transformations in a general mathematical process.

The kinematics components, as required by LINUXCNC, are given as well as VIS-MACH simulation models to demonstrate their behaviour on a computer screen. Examples of HAL file data are also given.

2   5-AXIS MACHINE TOOL CONFIGURATIONS

In this section we deal with the typical 5-axis milling or router machines with five joints or degrees-of-freedom which are controlled in coordinated moves.

3-axis machine tools cannot change the tool orientation, so 5-axis machine tools use two extra axes to set the cutting tool in an appropriate orientation for efficient machining of freeform surfaces.

A typical 5-axis machine tool (home converted for LINUXCNC) is shown on the front page, and more examples in Figs. 4, 6 and 10-12 in section Figures [1,2].

The kinematics of 5-axes machine tools are much simpler than that of 6-axis serial arm robots, since 3 of the axes are normally linear axes and only two are rotational axes.

3   TOOL ORIENTATION AND LOCATION

CAD/CAM systems are typically used to generate the 3D CAD models of the workpiece as well as the CAM data for input to the CNC 5-axis machine. The tool or cutter location (CL) data, is composed of the cutter tip position and the cutter orientation relative to the workpiece coordinate system. Two vectors, as generated by most CAM systems, as shown in Fig. 1, contain this information:

$$K = \begin{bmatrix} K_x \\ K_y \\ K_z \\ 0 \end{bmatrix} \quad \text{orientation vector;} \qquad Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} \quad \text{position vector;} \qquad (1)$$

The K vector is equivalent to the 3rd vector from the pose matrix E_6 that was used in the 6-axis robot kinematics [3] and the Q vector is equivalent to the 4th vector of E_6. In MASTERCAM for example this information is contained in the intermediate output ".nci" file.
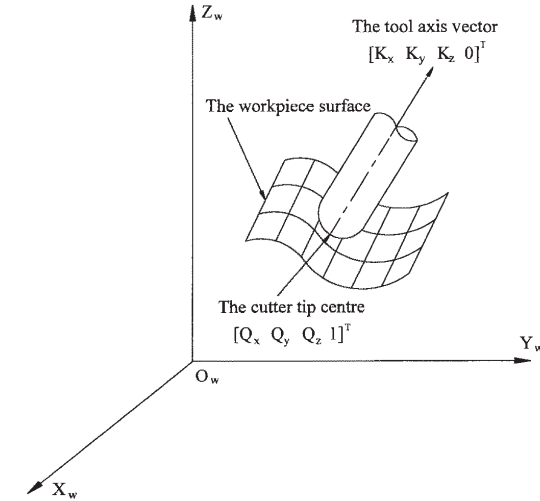
Figure 1: Cutter location data

4 TRANSLATION AND ROTATION MATRICES

There are four fundamental transformation matrices on which 5-axis kinematics can be based:

$$T(a,b,c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R(X,\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\theta & -S\theta & 0 \\ 0 & S\theta & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$R(Y,\theta) = \begin{bmatrix} C\theta & 0 & S\theta & 0 \\ 0 & 1 & 0 & 0 \\ -S\theta & 0 & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R(Z,\theta) = \begin{bmatrix} C\theta & -S\theta & 0 & 0 \\ S\theta & C\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The matrix $T(a,b,c)$ implies a translation in the X, Y, Z coordinate directions by the amounts a, b, c respectively. The R matrices imply rotations of the angle θ about the X, Y and Z coordinate axes respectively. The "C" and "S" letters refer to cosine and sine functions respectively.

5 TABLE ROTARY/TILTING 5-AXIS CONFIGURATIONS: TRT

In these machine tools there two rotational axes mount on the work table of the machine. Two forms are typically used:

- A rotary table which rotates about the vertical Z-axes (C-rotation, secondary) mounted on a tilting table which rotates about the X- or Y-axis (A- or B-rotation, primary). The workpiece is mounted on the rotary table.

- A tilting table which rotates about the X- or Y-axis (A- or B-rotation, secondary) is mounted on a rotary table which rotates about the Z-axis (C-rotation, primary), with the workpiece on the tilting table.
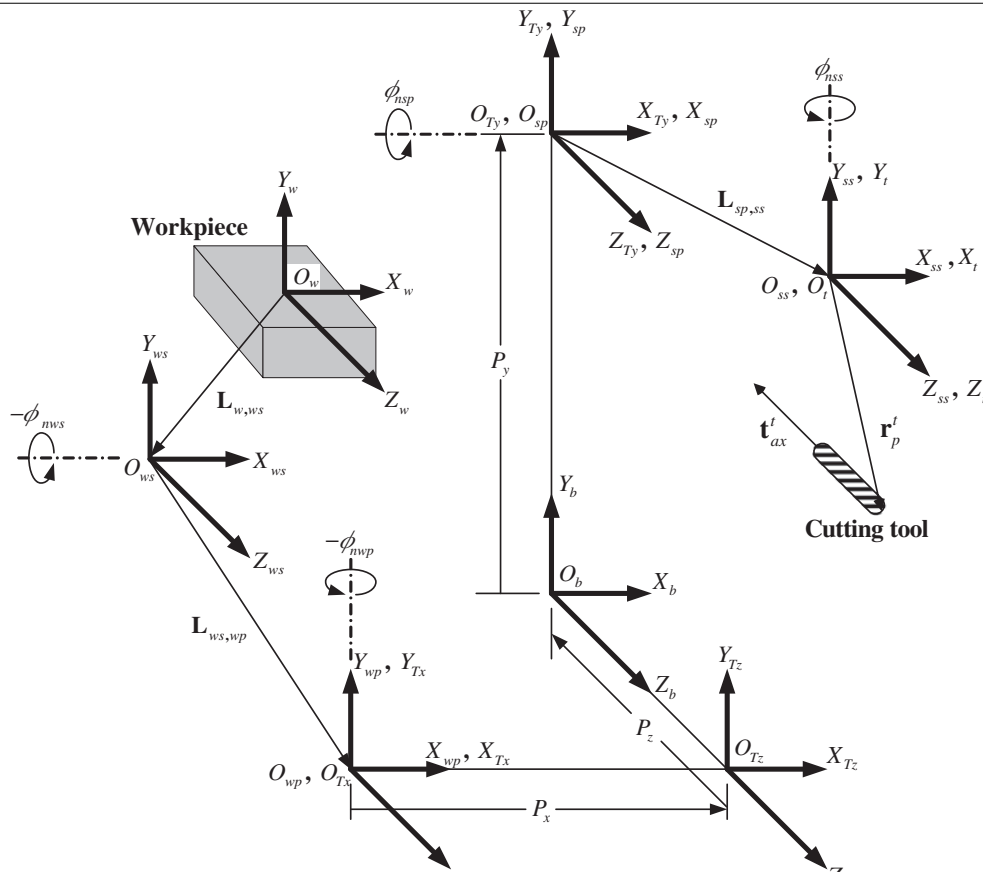
Figure 2: General configuration and coordinate systems

We need to describe a relationship between the workpiece coordinate system and the tool coordinate system. This can be defined by a transformation matrix ^wA_t, which can be found by subsequent transformations between the different structural elements of the machine, each with its own defined coordinate system. In general such a transformation may look as follows:

$$^wA_t = {^wA_1} \cdot {^1A_2} \cdot {^2A_3} \cdots {^nA_t} \tag{4}$$

where each matrix $^{i-1}A_i$ is a translation matrix T or a rotation matrix R of the form (2,3).

In Fig. 2 a generic configuration with coordinate systems is shown [4]. It includes table rotary/tilting axes as well as spindle rotary/tilting axes. Only two of the rotary axes are actually used in a machine tool.

First we will develop the transformations for the first type of configuration mentioned above, ie. a table tilting/rotary (TRT) type with no rotating axis offsets. We may give it the name XYZAC-TRT configuration.

We also develop the transformations for same type (XYZAC-TRT), but with rotating axis offsets.

Then we develop the transformations for a XYZBC-TRT configuration with rotating axis offsets.
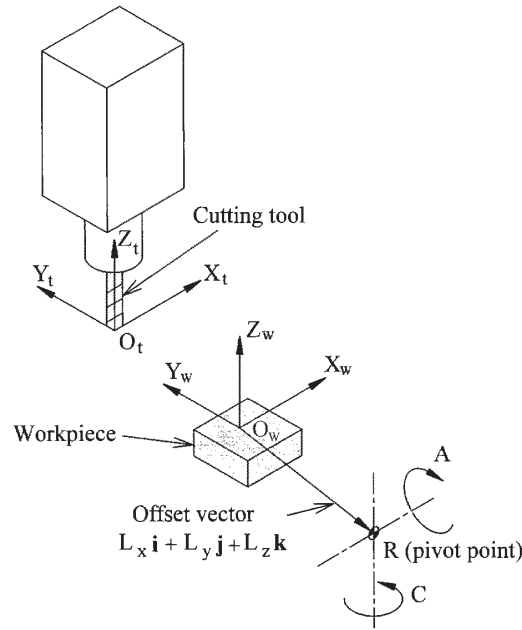
Figure 3: Table tilting/rotary configuration

5.1 Transformations for a XYZAC-TRT machine tool with work offsets

We deal here with a simplified configuration in which the tilting axis and rotary axis intersects at a point called the pivot point as shown in Fig. 3. therefore the two coordinate systems O_{ws} and O_{wp} of Fig. 2 are coincident.

Forward transformation

The transformation can be defined by the sequential multiplication of the matrices:

$$^{w}A_t =^{w} A_C \cdot^{C} A_A \cdot^{A} A_P \cdot^{P} A_t \tag{5}$$

with the matrices built up as follows:

$$^{w}A_C = \begin{bmatrix} 1 & 0 & 0 & L_x \\ 0 & 1 & 0 & L_y \\ 0 & 0 & 1 & L_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ^{C}A_A = \begin{bmatrix} C_C & S_C & 0 & 0 \\ -S_C & C_C & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$^{A}A_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C_A & S_A & 0 \\ 0 & -S_A & C_A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ^{P}A_t = \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

In these equations L_x, L_y, L_z defines the offsets of the pivot point of the two rotary axes A and C relative to the workpiece coordinate system origin. Furthermore, P_x, P_y, P_z are the relative distances of the pivot point to the cutter tip position, which can also be called the "joint coordinates" of the pivot point. The pivot point is at the intersection of

the two rotary axes. The signs of the S_A and S_C terms are different to those in (2,3) since there the table rotations are negative relative to the workpiece coordinate axes (note that $sin(-\theta) = -sin(\theta), cos(-\theta) = cos(\theta)$).

When multiplied in accordance with (5), we obtain:

$$
{}^wA_t = \begin{bmatrix}
C_C & S_CC_A & S_CS_A & C_CP_x + S_CC_AP_y + S_CS_AP_z + L_x \\
-S_C & C_CC_A & C_CS_A & -S_CP_x + C_CC_AP_y + C_CS_AP_z + L_y \\
0 & -S_A & C_A & -S_AP_y + C_AP_z + L_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\tag{8}
$$

We can now equate the third column of this matrix with our given tool orientation vector K, ie.:

$$
K = \begin{bmatrix} K_x \\ K_y \\ K_z \\ 0 \end{bmatrix} = \begin{bmatrix} S_CS_A \\ C_CS_A \\ C_A \\ 0 \end{bmatrix}
\tag{9}
$$

From these equations we can solve for the rotation angles θ_A, θ_C. From the third row we find:

$$
\theta_A = cos^{-1}(K_z) \qquad (0 < \theta_A < \pi)
\tag{10}
$$

and by dividing the first row by the second row we find:

$$
\theta_C = tan2^{-1}(K_x, K_y) \qquad (-\pi < \theta_C < \pi)
\tag{11}
$$

These relationships are typically used in the CAM post-processor to convert the tool orientation vectors to rotation angles.

Equating the last column of (8) with the tool position vector Q, we can write:

$$
Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix}
C_CP_x + S_CC_AP_y + S_CS_AP_z + L_x \\
-S_CP_x + C_CC_AP_y + C_CS_AP_z + L_y \\
-S_AP_y + C_AP_z + L_z \\
1
\end{bmatrix}
\tag{12}
$$

The vector on the right hand side can also be written as the product of a matrix and a vector resulting in:

$$
Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix}
C_C & S_CC_A & S_CS_A & L_x \\
-S_C & C_CC_A & C_CS_A & L_y \\
0 & -S_A & C_A & L_z \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = {}^QA_P \cdot P
\tag{13}
$$

This can be expanded to give

$$
\begin{aligned}
Q_x &= C_cP_x + S_CC_AP_y + S_CS_AP_z + L_x \\
Q_y &= -S_CP_x + C_CC_AP_y + C_CS_AP_z + L_y \\
Q_z &= -S_AP_y + C_AP_z + L_z
\end{aligned}
\tag{14}
$$

which is the *forward transformation* of the kinematics.

Inverse Transformation

We can solve for P from equation (13) as $P = ({}^Q A_P)^{-1} \cdot Q$. Noting that the square matrix is a homogenous 4x4 matrix containing a rotation matrix R and translation vector q, for which the inverse can be written as:

$$
{}^q A_p = \begin{bmatrix} R & q \\ 0 & 1 \end{bmatrix} \qquad ({}^Q A_P)^{-1} = \begin{bmatrix} R^T & -R^T q \\ 0 & 1 \end{bmatrix} \tag{15}
$$

where R^T is the transpose of R. We therefore obtain:

$$
\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C & -S_C & 0 & -C_C L_x + S_C L_y \\ S_C C_A & C_C C_A & -S_A & -S_C C_A L_x - C_C C_A L_y + S_A L_z \\ S_C S_A & C_C S_A & C_A & -S_C S_A L_x - C_C S_A L_y - C_A L_z \\ 0 & 0 & & 1 \end{bmatrix} \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} \tag{16}
$$

The desired equations for the *inverse transformation* of the kinematics thus can be written as:

$$
\begin{aligned}
P_x &= C_c(Q_x - L_x) - S_C(Q_y - L_y) \\
P_y &= S_C C_A(Q_x - L_x) + C_C C_A(Q_y - L_y) - S_A(Q_z - L_z) \\
P_z &= S_C S_A(Q_x - L_x) + C_C S_A(Q_y - L_y) + C_A(Q_z - L_z)
\end{aligned} \tag{17}
$$

5.2 Transformations for a XYZAC-TRT machine with rotary axis offsets

We deal here with a extended configuration in which the tilting axis and rotary axis do not intersect at a point but have an offset D_y. Furthermore, there is also an z-offset between the two coordinate systems O_{ws} and O_{wp} of Fig. 2, called D_z. A VISMACH model is shown in Fig. 4 and the offsets are shown in Fig. 5 (positive offsets in this example). To simplify the configuration, the offsets L_x, L_y, L_z of the previous case are not included. They are probably not necessary if one uses the G54 offsets in Linuxcnc by means of the "touch of" facility.

Forward Transformation

The transformation can be defined by the sequential multiplication of the matrices:

$$
{}^w A_t = {}^w A_O \cdot {}^O A_A \cdot {}^A A_P \cdot {}^P A_t \tag{18}
$$

with the matrices built up as follows:

$$
{}^w A_O = \begin{bmatrix} C_C & S_C & 0 & 0 \\ -S_C & C_C & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^O A_A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & D_y \\ 0 & 0 & 1 & D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{19}
$$

$$
{}^A A_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C_A & S_A & 0 \\ 0 & -S_A & C_A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^P A_t = \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y - D_y \\ 0 & 0 & 1 & P_z - D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{20}
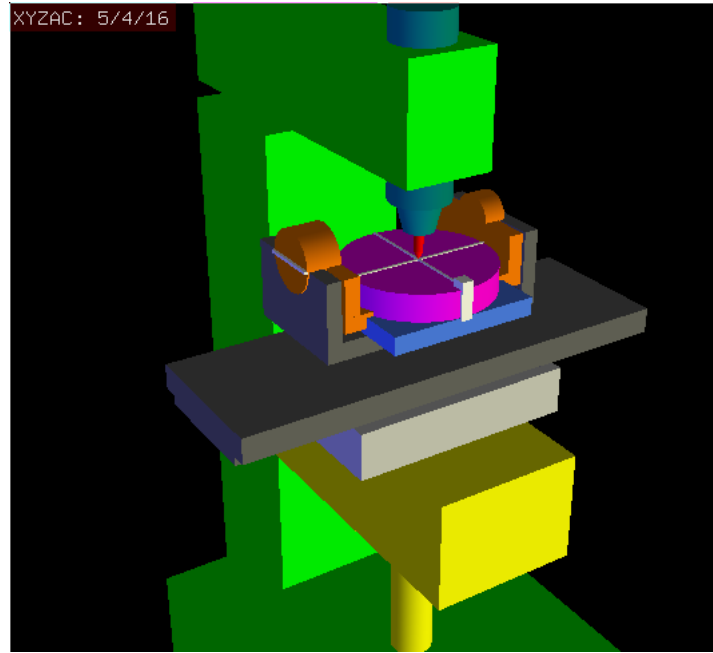$$

Figure 4: VISMACH model of XYZAC-TRT

In these equations D_y, D_z defines the offsets of the pivot point of the rotary axes A relative to the workpiece coordinate system origin. Furthermore, P_x, P_y, P_z are the relative distances of the pivot point to the cutter tip position, which can also be called the "joint coordinates" of the pivot point. The pivot point is on the A rotary axis.

When multiplied in accordance with (18), we obtain:

$$
{}^w A_t = \begin{bmatrix}
C_C & S_C C_A & S_C S_A & C_C P_x + S_C C_A (P_y - D_y) + S_C S_A (P_z - D_z) + S_C D_y \\
-S_C & C_C C_A & C_C S_A & -S_C P_x + C_C C_A (P_y - D_y) + C_C S_A (P_z - D_z) + C_C D_y \\
0 & -S_A & C_A & -S_A (P_y - D_y) + C_A (P_z - D_z) + D_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\tag{21}
$$

We can now equate the third column of this matrix with our given tool orientation vector K, ie.:

$$
K = \begin{bmatrix} K_x \\ K_y \\ K_z \\ 0 \end{bmatrix} = \begin{bmatrix} S_C S_A \\ C_C S_A \\ C_A \\ 0 \end{bmatrix}
\tag{22}
$$

From these equations we can solve for the rotation angles θ_A, θ_C. From the third row we find:

$$
\theta_A = cos^{-1}(K_z) \qquad (0 < \theta_A < \pi)
\tag{23}
$$

and by dividing the second row by the first row we find:

$$
\theta_C = tan2^{-1}(K_x, K_y) \qquad (-\pi < \theta_C < \pi)
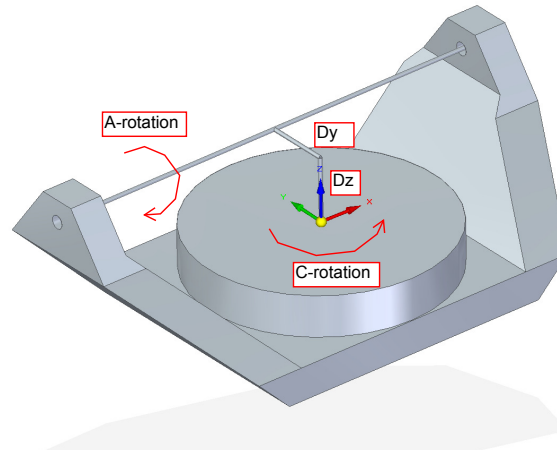\tag{24}
$$

Figure 5: Table tilting/rotary XYZAC-TRT configuration, with axis offsets

These relationships are typically used in the CAM post-processor to convert the tool orientation vectors to rotation angles.

Equating the last column of (21) with the tool position vector Q, we can write:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C P_x + S_C C_A (P_y - D_y) + S_C S_A (P_z - D_z) + S_C D_y \\ -S_C P_x + C_C C_A (P_y - D_y) + C_C S_A (P_z - D_z) + C_C D_y \\ -S_A (P_y - D_y) + C_A (P_z - D_z) + D_z \\ 1 \end{bmatrix} \quad (25)$$

The vector on the right hand side can also be written as the product of a matrix and a vector resulting in:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C & S_C C_A & S_C S_A & -S_c C_A D_y - S_C S_A D_z + S_C D_y \\ -S_C & C_C C_A & C_C S_A & -C_C C_A D_y - C_C S_A D_z + C_C D_y \\ 0 & -S_A & C_A & S_A D_y - C_A D_z + D_z \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = ^Q A_P \cdot P \quad (26)$$

which is the *forward transformation* of the kinematics.

Inverse Transformation

We can solve for P from equation (25) as $P = (^Q A_P)^{-1} \cdot Q$ using (15) as before. We thereby obtain:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C & -S_C & 0 & 0 \\ S_C C_A & C_C C_A & -S_A & -C_A D_y + S_A D_z + D_y \\ S_C S_A & C_S S_A & C_A & -S_A D_y - C_A D_z + D_z \\ 0 & 0 & & 1 \end{bmatrix} \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} \quad (27)$$

The desired equations for the *inverse transformation* of the kinematics thus can be written as:

$$
\begin{aligned}
P_x &= C_c Q_x - S_C Q_y \\
P_y &= S_C C_A Q_x + C_C C_A Q_y - S_A Q_z - C_A D_y + S_A D_z + D_y \\
P_z &= S_C S_A Q_x + C_S S_A Q_y + C_A Q_z - S_A D_y - C_A D_z + D_z
\end{aligned}
\tag{28}
$$

5.3 Transformations for a XYZBC-TRT machine with rotary axis offsets
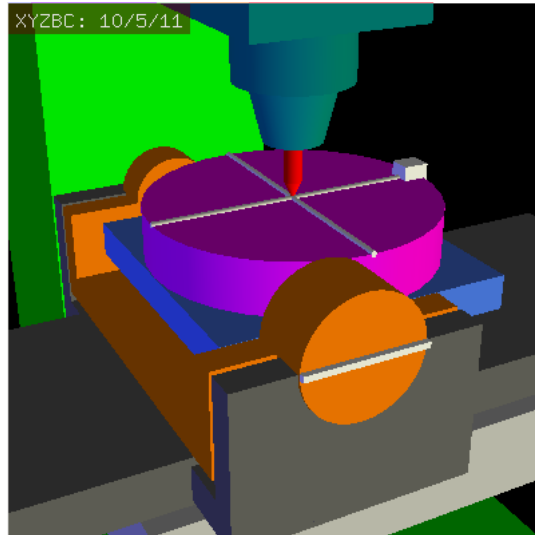

Figure 6: VISMACH model of XYZBC-TRT

We deal here again with a extended configuration in which the tilting axis (about the y-axis) and rotary axis do not intersect at a point but have an offset D_x. Furthermore, there is also an z-offset between the two coordinate systems O_{ws} and O_{wp} of Fig. 2, called D_z. A VISMACH model is shown in Fig. 6 (negative offsets in this example) and the positive offsets are shown in Fig. 7.

Forward Transformation

The transformation can be defined by the sequential multiplication of the matrices:

$$
{}^w A_t = {}^w A_O \cdot {}^O A_B \cdot {}^B A_P \cdot {}^P A_t
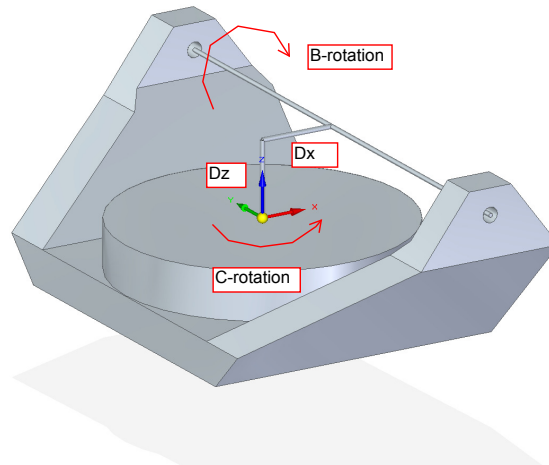\tag{29}
$$

Figure 7: Table tilting/rotary XYZBC-TRT configuration, with axis offsets

with the matrices built up as follows:

$$
{}^{w}A_{O} = \begin{bmatrix} C_C & S_C & 0 & 0 \\ -S_C & C_C & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{O}A_{B} = \begin{bmatrix} 1 & 0 & 0 & D_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{30}
$$

$$
{}^{B}A_{P} = \begin{bmatrix} C_B & 0 & -S_B & 0 \\ 0 & 1 & 0 & 0 \\ S_B & 0 & C_B & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{P}A_{t} = \begin{bmatrix} 1 & 0 & 0 & P_x - D_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z - D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{31}
$$

In these equations D_x, D_z defines the offsets of the pivot point of the rotary axes B relative to the workpiece coordinate system origin. Furthermore, P_x, P_y, P_z are the relative distances of the pivot point to the cutter tip position, which can also be called the "joint coordinates" of the pivot point. The pivot point is on the B rotary axis.

When multiplied in accordance with (29), we obtain:

$$
{}^{w}A_{t} = \begin{bmatrix} C_C C_B & S_C & -C_C S_B & C_C C_B(P_x - D_x) + S_C P_y - C_C S_B(P_z - D_z) + C_C D_x \\ -S_C C_B & C_C & S_C S_B & -S_C C_B(P_x - D_X) + C_C P_y + S_C S_B(P_z - D_z) - S_C D_x \\ S_B & 0 & C_B & S_B(P_x - D_x) + C_B(P_z - D_z) + D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{32}
$$

We can now equate the third column of this matrix with our given tool orientation vector K, ie.:

$$
K = \begin{bmatrix} K_x \\ K_y \\ K_z \\ 0 \end{bmatrix} = \begin{bmatrix} -C_C S_B \\ S_C S_B \\ C_B \\ 0 \end{bmatrix} \tag{33}
$$

From these equations we can solve for the rotation angles θ_B, θ_C. From the third row we find:

$$\theta_B = cos^{-1}(K_z) \qquad (0 < \theta_B < \pi) \tag{34}$$

and by dividing the second row by the first row we find:

$$\theta_C = tan2^{-1}(K_y, K_x) \qquad (-\pi < \theta_C < \pi) \tag{35}$$

These relationships are typically used in the CAM post-processor to convert the tool orientation vectors to rotation angles.

Equating the last column of (32) with the tool position vector Q, we can write:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C C_B(P_x - D_x) + S_C P_y - C_C S_B(P_z - D_z) + C_C D_x \\ -S_C C_B(P_x - D_X) + C_C P_y + S_C S_B(P_z - D_z) - S_C D_x \\ S_B(P_x - D_x) + C_B(P_z - D_z) + D_z \\ 1 \end{bmatrix} \tag{36}$$

The vector on the right hand side can also be written as the product of a matrix and a vector resulting in:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C C_B & S_C & -C_C S_B & -C_c C_B D_x + C_C S_B D_z + C_C D_x \\ -S_C C_B & C_C & S_C S_B & S_C C_B D_x - S_C S_B D_z - S_C D_x \\ S_B & 0 & C_B & -S_B D_x - C_B D_z + D_z \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} =^Q A_P \cdot P \tag{37}$$

which is the *forward transformation* of the kinematics.

Inverse Transformation

We can solve for P from equation (37) as $P = (^Q A_P)^{-1} \cdot Q$. With the same approach as before, we obtain:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C C_B & -S_C C_B & S_B & -C_B D_x - S_B D_z + D_x \\ S_C & C_C & 0 & 0 \\ -C_C S_B & S_C S_B & C_B & S_B D_x - C_B D_z + D_z \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} \tag{38}$$

The desired equations for the *inverse transformation* of the kinematics thus can be written as:

$$\begin{aligned} P_x &= C_c C_B Q_x - S_C C_B Q_y + S_B Q_z - C_B D_x - S_B D_z + D_x \\ P_y &= S_C Q_x + C_C Q_y \\ P_z &= -C_C S_B Q_x + S_C S_B Q_y + C_B Q_z + S_B D_x - C_B D_z + D_z \end{aligned} \tag{39}$$

6   SPINDLE ROTARY/TILTING 5-AXIS CONFIGURATIONS: SRT

In these machine tools there two rotational axes mount on the work spindle head of the machine. Two forms are typically used:
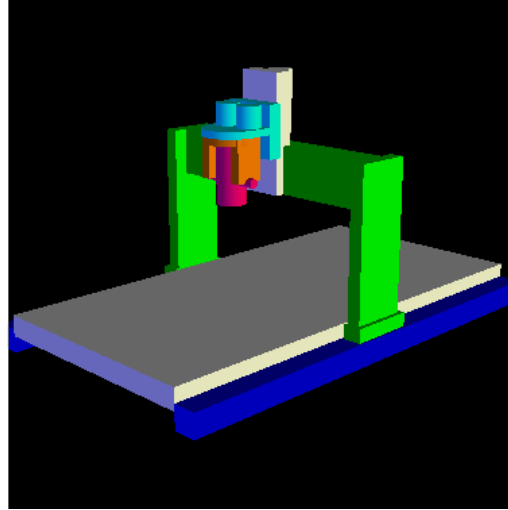
Figure 8: Rotating/tilting spindle router

- A rotary axis which rotates about the vertical Z-axes (C-rotation, primary) on which a tilting spindle is mounted such that the tilting is a secondary rotation about the X-axis (A-rotation) or Y-axis (B-rotation).

- A rotary axis (primary) which rotates about the X-axis or Y-axis on which a tilting spindle is mounted such that the tilting is a secondary rotation about the opposite Y-axis or X-axis.

As an example we will develop the transformations for the first type of configuration mentioned above, in which we will use for the secondary rotation a B-rotation. We give the name XYZBC-SRT to this configuration.

6.1 Transformations for a XYZBC-SRT machine tool

A model of a router with a rotary/tilting spindle head is shown in Fig. 8. The general geometry of a tilting/rotating head with three possible rotations is shown in Fig. 9. Only two rotations are typically implemented.

Forward Transformation

The transformation can be defined by the sequential multiplication of the matrices:

$$^{w}A_t = {}^{w}A_P \cdot {}^{P}A_C \cdot {}^{C}A_B \cdot {}^{B}A_t \tag{40}$$

with the matrices built up as follows:

$$^{w}A_P = \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{P}A_C = \begin{bmatrix} C_C & S_C & 0 & 0 \\ -S_C & C_C & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{41}$$
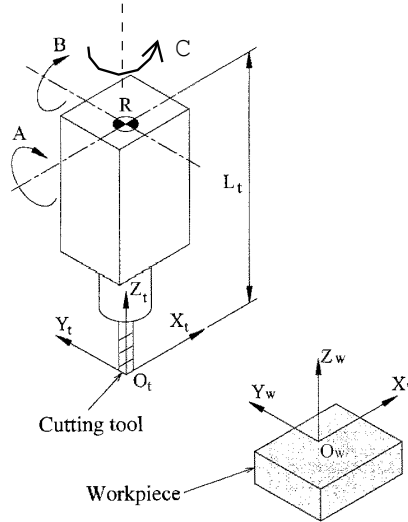
Figure 9: Spindle tilting/rotary configuration

$$
{}^C A_B = \begin{bmatrix} C_B & 0 & S_B & 0 \\ 0 & 1 & 0 & 0 \\ -S_B & 0 & C_B & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^B A_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -L_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{42}
$$

In these equations L_t defines the offset of the pivot point of the two rotary axes C and B relative to the tool tip. This gives the effective tool length, so that if tools are changed this variable must be set to reflect the new effective length.

P_x, P_y, P_z are again the relative distances of the pivot point with respect to the tool position in the workpiece coordinate system, which can also be called the "joint coordinates" of the pivot point.

When multiplied in accordance with (40), we obtain:

$$
{}^w A_t = \begin{bmatrix} C_C C_B & -S_C & C_C S_B & -C_C S_B L_t + P_x \\ S_C C_B & C_C & S_C S_B & -S_C S_B L_t + P_y \\ 0 & 0 & C_B & -C_B L_t + P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{43}
$$

We can now equate the third column of this matrix with our given tool orientation vector K, ie.:

$$
K = \begin{bmatrix} K_x \\ K_y \\ K_z \\ 0 \end{bmatrix} = \begin{bmatrix} C_C S_B \\ S_C S_B \\ C_B \\ 0 \end{bmatrix} \tag{44}
$$

From these equations we can solve for the rotation angles θ_B, θ_C. From the third row we find:

$$
\theta_B = cos^{-1}(K_z) \qquad (0 < \theta_A < \pi) \tag{45}
$$

and by dividing the second row by the first row we find:

$$
\theta_C = tan2^{-1}(K_y, K_x) \qquad (-\pi < \theta_C < \pi) \tag{46}
$$

These relationships can be used in the CAM post-processor to convert the tool orientation vectors to rotation angles.

Equating the last column of (43) with the tool position vector Q, we can write:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} -C_C S_B L_t + P_x \\ -S_C S_B L_t + P_y \\ -C_B L_t + P_z \\ 1 \end{bmatrix} \tag{47}$$

This can be rewritten to give simply

$$\begin{aligned} Q_x &= P_x - C_C S_B L_t \\ Q_y &= P_y - S_C S_B L_t \\ Q_z &= P_z - C_B L_t \end{aligned} \tag{48}$$

which is the *forward transformation* of the kinematics.

The relationship between the X, Y, Z values which we have to supply in our Gcode program can be found from (48) by setting $\theta_A = \theta_C = 0$ and $X = Q_x, Y = Q_y, Z = Q_z$. This results in:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z - L_t \\ 1 \end{bmatrix} \tag{49}$$

Inverse Transformation

We can solve for P from equation (48) directly to get:

$$\begin{aligned} P_x &= Q_x + C_C S_B L_t \\ P_y &= Q_y + S_C S_B L_t \\ P_z &= Q_z + C_B L_t \end{aligned} \tag{50}$$

These are desired equations for the *inverse transformation* of the kinematics.

and using (49):

$$\begin{aligned} X &= Q_x + C_C S_B L_t \\ Y &= Q_y + S_C S_B L_t \\ Z &= Q_z + C_B L_t - L_t \end{aligned} \tag{51}$$

7 TOOL-LENGTH COMPENSATION

In order to use tools from a tool table sequentially with tool-length compensation applied automatically, a further Z-offset is required. For a tool that is longer than the "master" tool, which typically has a tool length of zero, Linuxcnc has a variable called "motion.tooloffset.z". If this variable is passed on to the kinematic component (and VISMACH PYTHON script), then the necessary additional Z-offset for a new tool can be accounted for by adding the component statement, for example:

$D_z = D_z +$ tool-offset

where "tool-offset" is the variable passed on the the component by a .hal entry such as:

```
net tool-offset <= motion.tooloffset.z => XYZACkins.Tool-offset
```

8 KINEMATICS COMPONENTS

The kinematics is provided in LinuxCNC by a specially written component in the C-language. It has a standard procedure structure and is therefore normally copied from some standard example from the library of components, and then modified.

The component is compiled and installed in the correct place in the file system by a command such as:

```
sudo comp --install kinsname.c
```

in older versions of Linuxcnc or

```
sudo halcompile --install kinsname.c
```

in newer versions, where "kinsname" is the name you give to your component. The sudo prefix is required to install it and you will be asked for your root password.

Once it is compiled and installed you can reference it in your config setup of your machine. This is done in the .hal file of your config directory. The standard command

```
loadrt trivkins
```

is replaced by

```
loadrt kinsname
```

where "kinsname" is the name of your kins program. A further modification to the .hal file is required (typically at the end of your initial template file), where we have to set the offset parameters of the configuration, such as $D_x, D_y, D_z, Tool - offset$. In our XYZAC-TRT configuration the entries could be for example:

```
# set offset parameters
net tool-offset <= motion.tooloffset.z => XYZACkins.Tool-offset
setp XYZACkins.Y-offset 0
setp XYZACkins.Z-offset 20
```

Three examples of kinematics components, applicable to our XYZAC-TRT, XYZBC-TRT and XYZBC-SRT configurations, are given in Appendix A: Kinematics components.

9   VISMACH Simulation Models

VISMACH is a library of PYTHON routines to display a dynamic simulation of the CNC machine on the PC screen. The PYTHON script for a particular machine is loaded in the .hal file and data to it is passed on by pin connections. Typical entries in the .hal file are (example for XYZBC-TRT):

```
loadusr -W ./XYZBCgui
net j0 axis.0.joint-pos-fb XYZBCgui.table-x
net j1 axis.1.joint-pos-fb XYZBCgui.saddle-y
net j2 axis.2.joint-pos-fb XYZBCgui.spindle-z
net j4 axis.4.joint-pos-fb XYZBCgui.tilt-b
net j5 axis.5.joint-pos-fb XYZBCgui.rotate-c

setp XYZBCgui.X-offset -20
setp XYZBCgui.Z-offset -10
net tool-offs => XYZBCgui.Tool-offset
```

Three examples of VISMACH simulation PYTHON scripts, applicable to our XYZAC-TRT, XYZBC-TRT and XYZBC-SRT configurations, are given in Appendix B: VISMACH simulation PYTHON scripts.

10   HAL FILE EXAMPLES

In addition to the normal HAL file entries for 5 joints, some entries are required to pass data on to the kinematics component and the Vismach script. Here is an example of these additional entries in the HAL file for the XYZBC-TRT configuration (with negative offsets - see Fig. 6):

```
# load RT modules: first one the 5-axis kinematics
loadrt XYZBCkins

....
....
#  extras for XYZBC 5 axis config

setp XYZBCkins.X-offset -20
setp XYZBCkins.Z-offset -10
net tool-offs <= motion.tooloffset.z
net tool-offs => XYZBCkins.Tool-offset

loadusr -W ./XYZBCgui
net j0 axis.0.joint-pos-fb XYZBCgui.table-x
net j1 axis.1.joint-pos-fb XYZBCgui.saddle-y
net j2 axis.2.joint-pos-fb XYZBCgui.spindle-z
```

```
net j4 axis.4.joint-pos-fb XYZBCgui.tilt-b
net j5 axis.5.joint-pos-fb XYZBCgui.rotate-c

setp XYZBCgui.X-offset -20
setp XYZBCgui.Z-offset -10
net tool-offs => XYZBCgui.Tool-offset
```

11 REFERENCES

[1] A Postprocessor Based on the Kinematics Model for General Five-Axis machine Tools: C-H She, R-S Lee, J Manufacturing Processes, V2 N2, 2000.

[2] NC Post-processor for 5-axis milling of table-rotating/tilting type: YH Jung, DW Lee, JS Kim, HS Mok, J Materials Processing Technology,130-131 (2002) 641-646.

[3] 3D 6-DOF Serial Arm Robot Kinematics, RJ du Preez, SA-CNC-CLUB, Dec. 5, 2013.

[4] Design of a generic five-axis postprocessor based on generalized kinematics model of machine tool: C-H She, C-C Chang, Int. J Machine Tools & Manufacture, 47 (2007) 537-545.
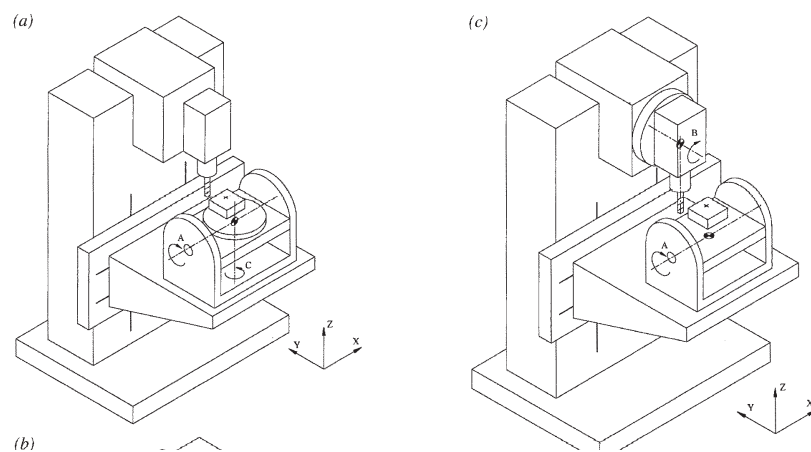
12 FIGURES

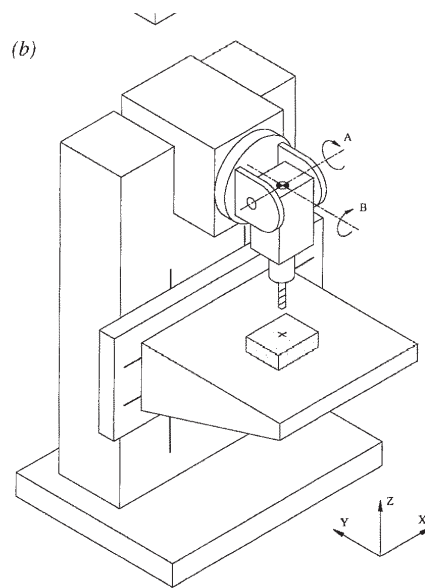Figure 10: Table tilting/rotating and Spindle/table tilting configuration

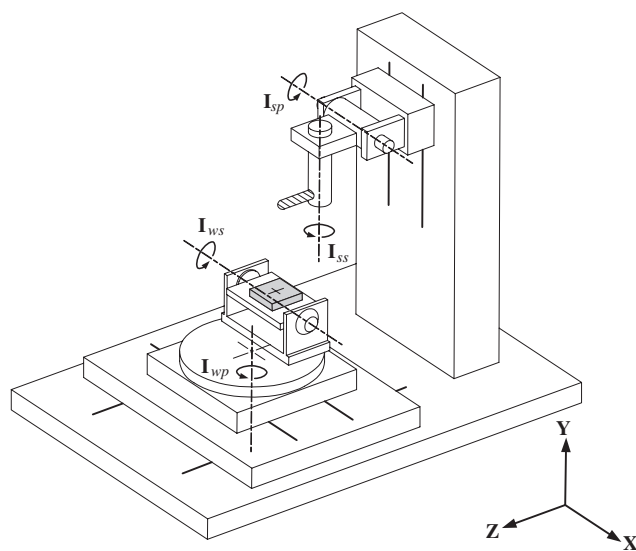Figure 11: Spindle tilting/tilting configuration

Figure 12: Spindle/table tilting/rotary configuration

13   APPENDIX A: Kinematics Components

13.1   Kinematics component for XYZAC-TRT

```c
/********************************************************************
* Description: XYZACkins.c
*   Kinematics for  5 axis mill named 'XYZAC'.
*   This mill has a tilting table (A axis) and horizontal rotary
*   mounted to the table (C axis).
*   with rotary axis offsets
*
* Author: Rudy du Preez
* License: GPL Version 2
*
********************************************************************/

#include "kinematics.h" /* these decls */
#include "posemath.h"
#include "hal.h"
#include "rtapi.h"
#include "rtapi_math.h"

struct haldata {
    hal_float_t *Y_offset;
    hal_float_t *Z_offset;
    hal_float_t *Tool_offset;
} *haldata;

int kinematicsForward(const double *joints,
      EmcPose * pos,
      const KINEMATICS_FORWARD_FLAGS * fflags,
      KINEMATICS_INVERSE_FLAGS * iflags)
{
    double dy = *(haldata->Y_offset);
    double dz = *(haldata->Z_offset);
    double dT = *(haldata->Tool_offset);
    double a_rad = joints[3]*M_PI/180;
    double c_rad = joints[5]*M_PI/180;
    dz = dz + dT;

    pos->tran.x = cos(c_rad)*(joints[0]) +
                sin(c_rad)*cos(a_rad)*(joints[1] - dy) +
                sin(c_rad)*sin(a_rad)*(joints[2] - dz) +
                sin(c_rad)*dy;

    pos->tran.y = -sin(c_rad)*(joints[0]) +
                 cos(c_rad)*cos(a_rad)*(joints[1] - dy) +
                 cos(c_rad)*sin(a_rad)*(joints[2] - dz) +
                 cos(c_rad)*dy;

    pos->tran.z = -sin(a_rad)*(joints[1] - dy) +
                cos(a_rad)*(joints[2] - dz) + dz;

    pos->a = joints[3];
    pos->b = joints[4];
    pos->c = joints[5];

    return 0;
}

int kinematicsInverse(const EmcPose * pos,
      double *joints,
      const KINEMATICS_INVERSE_FLAGS * iflags,
      KINEMATICS_FORWARD_FLAGS * fflags)
{
    double dy = *(haldata->Y_offset);
    double dz = *(haldata->Z_offset);
    double dT = *(haldata->Tool_offset);
    double c_rad = pos->c*M_PI/180;
```

```
    double a_rad = pos->a*M_PI/180;
    dz = dz + dT;

    joints[0] = cos(c_rad)*pos->tran.x -
                sin(c_rad)*pos->tran.y;

    joints[1] = sin(c_rad)*cos(a_rad)*pos->tran.x  +
                cos(c_rad)*cos(a_rad)*pos->tran.y  -
                sin(a_rad)*pos->tran.z -
                cos(a_rad)*dy + sin(a_rad)*dz + dy;

    joints[2] = sin(c_rad)*sin(a_rad)*pos->tran.x  +
                cos(c_rad)*sin(a_rad)*pos->tran.y  +
                cos(a_rad)*pos->tran.z -
                sin(a_rad)*dy - cos(a_rad)*dz + dz;

    joints[3] = pos->a;
    joints[4] = pos->b;
    joints[5] = pos->c;

    return 0;
}

KINEMATICS_TYPE kinematicsType()
{
    return KINEMATICS_BOTH;
}

#include "rtapi.h" /* RTAPI realtime OS API */
#include "rtapi_app.h" /* RTAPI realtime module decls */
#include "hal.h"

EXPORT_SYMBOL(kinematicsType);
EXPORT_SYMBOL(kinematicsInverse);
EXPORT_SYMBOL(kinematicsForward);
MODULE_LICENSE("GPL");

int comp_id;
int rtapi_app_main(void) {
    int res = 0;
    comp_id = hal_init("XYZACkins");
    if(comp_id < 0) return comp_id;

    haldata = hal_malloc(sizeof(struct haldata));

    if((res = hal_pin_float_new("XYZACkins.Tool-offset",  HAL_IN, &(haldata->Tool_offset),
                    comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("XYZACkins.Y-offset",  HAL_IO, &(haldata->Y_offset),
                    comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("XYZACkins.Z-offset",  HAL_IO, &(haldata->Z_offset),
                    comp_id)) < 0) goto error;

    hal_ready(comp_id);
    return 0;

error:
    hal_exit(comp_id);
    return res;

}

void rtapi_app_exit(void) { hal_exit(comp_id); }
```

Note that three extra HAL pins have been added: XYZACkins.Tool-offset, XYZACkins.Y-offset and XYZACkins.Z-offset.

13.2 Kinematics component for XYZBC-TRT with offsets

```
/*******************************************************************
 * Description: XYZBCkins.c
 *   Kinematics for  5 axis mill named 'XYZBC'.
 *   This mill has a tilting table (B axis) and horizontal rotary
 *   mounted to the table (C axis).
 *
 * Author: Rudy du Preez
 * License: GPL Version 2
 *
 *******************************************************************/

#include "kinematics.h" /* these decls */
#include "posemath.h"
#include "hal.h"
#include "rtapi.h"
#include "rtapi_math.h"

struct haldata {
    hal_float_t *X_offset;
    hal_float_t *Z_offset;
    hal_float_t *Tool_offset;
} *haldata;

int kinematicsForward(const double *joints,
      EmcPose * pos,
      const KINEMATICS_FORWARD_FLAGS * fflags,
      KINEMATICS_INVERSE_FLAGS * iflags)
{
    double dx = *(haldata->X_offset);
    double dz = *(haldata->Z_offset);
    double dT = *(haldata->Tool_offset);
    double b_rad = joints[4]*M_PI/180;
    double c_rad = joints[5]*M_PI/180;
    dz = dz + dT;

    pos->tran.x = cos(c_rad)*cos(b_rad)*(joints[0] - dx) +
                  sin(c_rad)*(joints[1]) -
                  cos(c_rad)*sin(b_rad)*(joints[2] - dz) +
                  cos(c_rad)*dx;

    pos->tran.y = -sin(c_rad)*cos(b_rad)*(joints[0] - dx) +
                  cos(c_rad)*(joints[1]) +
                  sin(c_rad)*sin(b_rad)*(joints[2] - dz) -
                  sin(c_rad)*dx;

    pos->tran.z =  sin(b_rad)*(joints[0] - dx) +
                   cos(b_rad)*(joints[2] - dz) +
                   dz;

    pos->a = joints[3];
    pos->b = joints[4];
    pos->c = joints[5];
    return 0;
}

int kinematicsInverse(const EmcPose * pos,
      double *joints,
      const KINEMATICS_INVERSE_FLAGS * iflags,
      KINEMATICS_FORWARD_FLAGS * fflags)
{
    double dx = *(haldata->X_offset);
    double dz = *(haldata->Z_offset);
    double dT = *(haldata->Tool_offset);
    dz = dz + dT;
    double b_rad = pos->b*M_PI/180;
    double c_rad = pos->c*M_PI/180;
    double dpx =  -cos(b_rad)*dx - sin(b_rad)*dz + dx;
    double dpz =   sin(b_rad)*dx - cos(b_rad)*dz + dz;
```

```
    joints[0] = cos(c_rad)*cos(b_rad)*(pos->tran.x) -
                sin(c_rad)*cos(b_rad)*(pos->tran.y) +
                sin(b_rad)*(pos->tran.z) + dpx;

    joints[1] = sin(c_rad)*(pos->tran.x) +
                cos(c_rad)*(pos->tran.y);

    joints[2] = -cos(c_rad)*sin(b_rad)*(pos->tran.x) +
                 sin(c_rad)*sin(b_rad)*(pos->tran.y) +
                 cos(b_rad)*(pos->tran.z) + dpz;

    joints[3] = pos->a;
    joints[4] = pos->b;
    joints[5] = pos->c ;

    return 0;
}

KINEMATICS_TYPE kinematicsType()
{
    return KINEMATICS_BOTH;
}

#include "rtapi.h" /* RTAPI realtime OS API */
#include "rtapi_app.h" /* RTAPI realtime module decls */
#include "hal.h"

EXPORT_SYMBOL(kinematicsType);
EXPORT_SYMBOL(kinematicsInverse);
EXPORT_SYMBOL(kinematicsForward);
MODULE_LICENSE("GPL");

int comp_id;
int rtapi_app_main(void) {
    int res = 0;
    comp_id = hal_init("XYZBCkins");
    if(comp_id < 0) return comp_id;

    haldata = hal_malloc(sizeof(struct haldata));

    if((res = hal_pin_float_new("XYZBCkins.X-offset",  HAL_IO, &(haldata->X_offset),
        comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("XYZBCkins.Z-offset",  HAL_IO, &(haldata->Z_offset),
        comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("XYZBCkins.Tool-offset",  HAL_IN, &(haldata->Tool_offset),
        comp_id)) < 0) goto error;

    hal_ready(comp_id);
    return 0;

error:
    hal_exit(comp_id);
    return res;

}

void rtapi_app_exit(void) { hal_exit(comp_id); }
```

13.3 Kinematics component for XYZBC-SRT

```
/*******************************************************************
* Description: sptiltkins.c   XYZBCW
*   Kinematics for 5 axis tilting spindle router
*
*
* Author: Rudy Preez SA-CNC-CLUB
* License: GPL Version 2
* System: Linux
*
*******************************************************************/

#include "kinematics.h" /* these decls */
#include "posemath.h"
#include "hal.h"
#include "rtapi_math.h"

#define d2r(d) ((d)*PM_PI/180.0)
#define r2d(r) ((r)*180.0/PM_PI)

struct haldata {
    hal_float_t *tool_length;
    hal_float_t *pivot_length;
} *haldata;

int kinematicsForward(const double *joints,
      EmcPose * pos,
      const KINEMATICS_FORWARD_FLAGS * fflags,
      KINEMATICS_INVERSE_FLAGS * iflags)
{
 double thB;
 double thC;
 double Lt;

   thB = d2r(joints[4]);
   thC = d2r(joints[5]);
   Lt  = *(haldata->pivot_length) + *(haldata->tool_length);

    pos->tran.x = joints[0] - cos(thC)*sin(thB)*Lt;
    pos->tran.y = joints[1] - sin(thC)*sin(thB)*Lt;
    pos->tran.z = joints[2] - cos(thB)*Lt + Lt;
    pos->b = joints[4];
    pos->c = joints[5];
    pos->w = joints[8];

    return 0;
}

int kinematicsInverse(const EmcPose * pos,
      double *joints,
      const KINEMATICS_INVERSE_FLAGS * iflags,
      KINEMATICS_FORWARD_FLAGS * fflags)
{

 double thB;
 double thC;
 double Lt;

   thB = d2r(joints[4]);
   thC = d2r(joints[5]);
   Lt  = *(haldata->pivot_length) + *(haldata->tool_length);


    joints[0] = pos->tran.x + cos(thC)*sin(thB)*Lt;
    joints[1] = pos->tran.y + sin(thC)*sin(thB)*Lt;
    joints[2] = pos->tran.z + cos(thB)*Lt - Lt;
    joints[4] = pos->b;
    joints[5] = pos->c;
    joints[8] = pos->w;
```

```
    return 0;
}

/* implemented for these kinematics as giving joints preference */
int kinematicsHome(EmcPose * world,
    double *joint,
    KINEMATICS_FORWARD_FLAGS * fflags,
    KINEMATICS_INVERSE_FLAGS * iflags)
{
    *fflags = 0;
    *iflags = 0;

    return kinematicsForward(joint, world, fflags, iflags);
}

KINEMATICS_TYPE kinematicsType()
{
    return KINEMATICS_BOTH;
}

#include "rtapi.h" /* RTAPI realtime OS API */
#include "rtapi_app.h" /* RTAPI realtime module decls */
#include "hal.h"

EXPORT_SYMBOL(kinematicsType);
EXPORT_SYMBOL(kinematicsForward);
EXPORT_SYMBOL(kinematicsInverse);
MODULE_LICENSE("GPL");

int comp_id;
int rtapi_app_main(void) {
    int result;
    comp_id = hal_init("sptiltkins");
    if(comp_id < 0) return comp_id;

    haldata = hal_malloc(sizeof(struct haldata));

    result = hal_pin_float_new("sptiltkins.tool-length", HAL_IN,
        &(haldata->tool_length), comp_id);
    if(result < 0) goto error;
    result = hal_pin_float_new("sptiltkins.pivot-length", HAL_IO,
        &(haldata->pivot_length), comp_id);
    if(result < 0) goto error;

    *(haldata->pivot_length) = 250.0;

    hal_ready(comp_id);
    return 0;

error:
    hal_exit(comp_id);
    return result;
}

void rtapi_app_exit(void) { hal_exit(comp_id); }
```

14   APPENDIX B: VISMACH Simulation PYTHON Scripts

14.1   XYZAC-TRT model

For our XYZAC-TRT machine the following is an example of a VISMACH script.

```python
#!/usr/bin/python
#---------------------------------------------------------------------
#    Visualization model of the Hermle  mill, as modified to 5-axis
#    with rotary axes A and C added, with moving knee and rotary axis offsets
#     Rudy du Preez, SA-CNC-CLUB, 3/2016
#---------------------------------------------------------------------

from vismach import *
import hal
import math
import sys

c = hal.component("XYZACgui")
# table-x
c.newpin("table-x", hal.HAL_FLOAT, hal.HAL_IN)
# saddle-y
c.newpin("saddle-y", hal.HAL_FLOAT, hal.HAL_IN)
# head vertical slide
c.newpin("spindle-z", hal.HAL_FLOAT, hal.HAL_IN)
# table-x tilt-b
c.newpin("tilt-a", hal.HAL_FLOAT, hal.HAL_IN)
# rotary table-x
c.newpin("rotate-c", hal.HAL_FLOAT, hal.HAL_IN)
# offsets
c.newpin("Y-offset", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("Z-offset", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("Tool-offset", hal.HAL_FLOAT, hal.HAL_IN)
c.ready()

for setting in sys.argv[1:]: exec setting

tooltip = Capture()

tool = Collection([
      tooltip,
      CylinderZ(0,0.2,6,3),
      CylinderZ(6,3,70,3)
    ])
tool = Translate([tool],0,0,-20)
tool = Color([1,0,0,0], [tool] )
tool = HalTranslate([tool],c,"Tool-offset",0,0,-1)

spindle = Collection([
# spindle nose and/or toolholder
Color([0,0.5,0.5,0], [CylinderZ( 0, 10, 20, 15)]),
# spindle housing
CylinderZ( 20, 20, 135, 20),
])

spindle = Color([0,0.5,0.5,0], [spindle])

spindle = Collection([
       tool,
       spindle
       ])
spindle = Translate([spindle],0,0,20)

#      spindle motor
motor = Collection([
      Color([0,0.5,0.5,0],
      [CylinderZ(135,30,200,30)])
      ])
```

```
motor = Translate([motor],0,200,0)

head = Collection([
spindle,
# head, holds spindle
Color([0,1,0,0], [Box( -30, -30, 60, 30, 240, 135 )]),
        motor
        ])
head = Translate([head],0,0,200)

work = Capture()

ctable = Collection([
work,
CylinderZ(-18, 50, 0, 50),
#        cross
        Color([1,1,1,0], [CylinderX(-50,1,50,1)]),
        Color([1,1,1,0], [CylinderY(-50,1,50,1)]),
# lump on one side
Color([1,1,1,0], [Box( -4, -42, -20, 4, -51, 5)])
        ])

ctable = HalRotate([ctable],c,"rotate-c",1,0,0,1)
ctable = Color([1,0,1,0], [ctable] )

crotary = Collection([
      ctable,
# # rotary table base - part under table
Color([0.3,0.5,1,0], [Box(-50,-50, -30, 50, 50, -18)])
      ])
crotary = HalTranslate([crotary],c,"Y-offset",0,-1,0)
crotary = HalTranslate([crotary],c,"Z-offset",0,0,-1)

yoke = Collection([
#        trunnion plate
        Color([1,0.5,0,0], [Box(-65,-40,-35,65,40,-25)]),
#        side plate left
        Color([1,0.5,0,0], [Box(-65,-40,-35,-55,40,0)]),
#        side plate right
        Color([1,0.5,0,0], [Box(55,-40,-35,65,40,0)])
        ])

trunnion = Collection([
        Color([1,0.5,0,0],[CylinderX(-78,20,-55,20)]),
        Color([1,0.5,0,0],[CylinderX(55,15,70,15)]),
#        mark on drive side
        Color([1,1,1,0], [Box(-80,-20,-1,-78,20,1)])
        ])

arotary = Collection([
        crotary, yoke,
    trunnion
        ])
arotary = HalRotate([arotary],c,"tilt-a",1,1,0,0)

arotary = HalTranslate([arotary],c,"Y-offset",0,1,0)
arotary = HalTranslate([arotary],c,"Z-offset",0,0,1)

brackets = Collection([
        # a bracket left side
        Box(-77,-40,-50,-67,40,0),
        # a bracket right side
        Box(77,-40,-50,67,40,0),
        # mounting plate
        Box(77,40,-52,-77,-40,-40)
   ])
brackets = HalTranslate([brackets],c,"Y-offset",0,1,0)
brackets = HalTranslate([brackets],c,"Z-offset",0,0,1)

# main table - for three axis, put work here instead of rotary
```

```
table = Collection([
arotary,
    brackets,
# body of table
Box(-150,-50, -69, 150, 50, -52),
# ways
Box(-150,-40, -75, 150, 40, -69)
])

table = HalTranslate([table],c,"table-x",-1,0,0)
table = Color([0.4,0.4,0.4,0], [table] )

saddle = Collection([
table,
#
Box(-75,-53, -105, 75, 53, -73),
])

saddle = HalTranslate([saddle],c,"saddle-y",0,-1,0)
saddle = Color([0.8,0.8,0.8,0], [saddle] )

zslide = Collection([
        saddle,
Box(-50, -100, -180, 50, 120, -105),
])

# default Z position is with the workpoint lined up with the toolpoint

zslide = Translate([zslide], 0, 0, 200)
zslide = HalTranslate([zslide],c,"spindle-z",0,0,-1)
zslide = Color([1,1,0,0], [zslide] )

base = Collection([
        head,
# base
Box(-120, -100, -250, 120, 160, -100),
# column
Box(-50, 100, -250, 50, 200, 260),
        # Z motor
        Color([1,1,0,0], [Box(-25,-100,-195,25,-110,-145)]),
        # Z lead screw
        Color([1,1,0,0], [CylinderZ(-100,15,50,15)])
])

base = Color([0,1,0,0], [base] )

model = Collection([zslide, base])

myhud = Hud()
myhud.show("XYZAC: 5/4/16")

main(model, tooltip, work, 500, hud=myhud)
```

14.2   XYZBC-TRT model

For our XYZBC-TRT machine the following is an example of a VISMACH script.

```
#!/usr/bin/python
#-------------------------------------------------------------------------------
#    Visualization model of the Hermle  mill, as modified to 5-axis
#    with rotary axes B and C added, with moving spindle head and rotary axis offsets
#
#    Rudy du Preez, SA-CNC-CLUB, 3/2016
#-------------------------------------------------------------------------------

from vismach import *
import hal
import math
```

```
import sys

c = hal.component("XYZBCgui")
# table-x
c.newpin("table-x", hal.HAL_FLOAT, hal.HAL_IN)
# saddle-y
c.newpin("saddle-y", hal.HAL_FLOAT, hal.HAL_IN)
# head vertical slide
c.newpin("spindle-z", hal.HAL_FLOAT, hal.HAL_IN)
# table-x tilt-b
c.newpin("tilt-b", hal.HAL_FLOAT, hal.HAL_IN)
# rotary table-x
c.newpin("rotate-c", hal.HAL_FLOAT, hal.HAL_IN)

c.newpin("Z-offset", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("X-offset", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("Tool-offset", hal.HAL_FLOAT, hal.HAL_IN)
c.ready()

for setting in sys.argv[1:]: exec setting

tooltip = Capture()

tool = Collection([
        tooltip,
        CylinderZ(0,0.2,6,3),
        CylinderZ(6,3,70,3)
    ])
tool = Translate([tool],0,0,-20)
tool = Color([1,0,0,0], [tool] )
tool = HalTranslate([tool],c,"Tool-offset",0,0,-1)

spindle = Collection([
# spindle nose and/or toolholder
Color([0,0.5,0.5,0], [CylinderZ( 0, 10, 20, 15)]),
# spindle housing
CylinderZ( 20, 20, 135, 20),
])

spindle = Color([0,0.5,0.5,0], [spindle])

spindle = Collection([
        tool,
        spindle
        ])
spindle = Translate([spindle],0,0,20)

#       spindle motor
motor = Collection([
        Color([0,0.5,0.5,0],
        [CylinderZ(135,30,200,30)])
        ])

motor = Translate([motor],0,60,0)

head = Collection([
spindle,
# head, holds spindle
Color([0,0.5,0.5,0], [Box( -30, -30, 60, 30, 130, 135 )]),
        motor
        ])
head = Translate([head],0,0,150)
head= HalTranslate([head],c,"spindle-z",0,0,1)

work = Capture()

ctable = Collection([
work,
CylinderZ(-18, 50, 0, 50),
#         cross
        Color([1,1,1,0], [CylinderX(-50,1,50,1)]),
```

```
        Color([1,1,1,0], [CylinderY(-50,1,50,1)]),
# lump on one side
Color([1,1,1,0], [Box( -4, -42, -20, 4, -51, 5)])
        ])

ctable = HalRotate([ctable],c,"rotate-c",1,0,0,1)
ctable = Color([1,0,1,0], [ctable] )

crotary = Collection([
        ctable,
# # rotary table base - part under table
Color([0.3,0.5,1,0], [Box(-50,-50, -30, 50, 50, -18)])
        ])
crotary = HalTranslate([crotary],c,"X-offset",0,1,0)
crotary = HalTranslate([crotary],c,"Z-offset",0,0,-1)

yoke = Collection([
#        trunnion plate
        Color([1,0.5,0,0], [Box(-65,-40,-35,65,40,-25)]),
#        side plate left
        Color([1,0.5,0,0], [Box(-65,-40,-35,-55,40,0)]),
#        side plate right
        Color([1,0.5,0,0], [Box(55,-40,-35,65,40,0)])
        ])

trunnion = Collection([
        Color([1,0.5,0,0],[CylinderX(-78,20,-55,20)]),
        Color([1,0.5,0,0],[CylinderX(55,15,70,15)]),
#        mark on drive side
        Color([1,1,1,0], [Box(-80,-20,-1,-78,20,1)])
        ])

arotary = Collection([
        crotary, yoke,
    trunnion
        ])
arotary = Rotate([arotary],90,0,0,1)
arotary = HalRotate([arotary],c,"tilt-b",1,0,1,0)

arotary = HalTranslate([arotary],c,"X-offset",1,0,0)
arotary = HalTranslate([arotary],c,"Z-offset",0,0,1)

brackets = Collection([
        # a bracket left side
        Box(-77,-40,-50,-67,40,0),
        # a bracket right side
        Box(77,-40,-50,67,40,0),
        # mounting plate
        Box(77,40,-52,-77,-40,-40)
   ])
brackets = Rotate([brackets],90,0,0,1)
brackets = HalTranslate([brackets],c,"X-offset",1,0,0)
brackets = HalTranslate([brackets],c,"Z-offset",0,0,1)

# main table - for three axis, put work here instead of rotary
table = Collection([
arotary,
    brackets,
# body of table
Box(-150,-50, -69, 150, 50, -52),
# ways
Box(-150,-40, -75, 150, 40, -69)
])

table = HalTranslate([table],c,"table-x",-1,0,0)
table = Color([0.4,0.4,0.4,0], [table] )

saddle = Collection([
table,
#
Box(-75,-53, -105, 75, 53, -73),
```

```
])

saddle = HalTranslate([saddle],c,"saddle-y",0,-1,0)
saddle = Color([0.8,0.8,0.8,0], [saddle] )

yslide = Collection([
        saddle,
Box(-50, -100, -180, 50, 120, -105),
])

# default Z position is with the workpoint lined up with the toolpoint

yslide = Translate([yslide], 0, 0, 150)
yslide = Color([0,1,0,0], [yslide] )

base = Collection([
        head,
# base
Box(-120, -100, -200, 120, 160, -30),
# column
Box(-50, 120, -200, 50, 220, 360)
])

base = Color([0,1,0,0], [base] )

model = Collection([yslide, base])

myhud = Hud()
myhud.show("XYZBC: 3/4/16")

main(model, tooltip, work, 500, hud=myhud)
```

14.3   XYZBC-STR model

For our XYZBC-SRT machine the following is an example of a VISMACH script.

```
#!/usr/bin/PYTHON
#----------------------------------------------------------------------------
#     VISMACH 5 axis rotating/tilting spindle router simulation
#     Rudy du Preez, SA-CNC-CLUB, 3/2014
#----------------------------------------------------------------------------
from vismach import *
import hal
import math
import sys

# give endpoint Z values and radii
# resulting cylinder is on the Z axis
class HalToolCylinder(CylinderZ):
    def __init__(self, comp, *args):
CylinderZ.__init__(self, *args)
self.comp = comp

    def coords(self):
        return self.comp.tool_length, 3, 0, 3

c = hal.component("sptiltgui")
c.newpin("joint0", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("joint1", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("joint2", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("joint3", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("joint4", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("joint5", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("tool_length", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("pivot_length", hal.HAL_FLOAT, hal.HAL_IN)
c.ready()

pivot_len = 50
```

```
tool_len  = 40
Lt = pivot_len + tool_len

for setting in sys.argv[1:]: exec setting

tooltip = Capture()
tool = Collection([tooltip,
                   HalToolCylinder(c,"tool_length"),
                   CylinderZ(tool_len, 30, Lt+70, 30)
                   ])

baxis = CylinderY(-65,10,65,10)
baxis = Translate([baxis],0,0,Lt)

baxis = Collection([tool,
                    baxis,
                    ])
baxis = Color([1,0,0.5,1],[baxis])
baxis = Translate([baxis],0,0,-Lt)
baxis = HalRotate([baxis],c,"joint4",1,0,1,0)
baxis = Translate([baxis],-130,0,Lt)

yoke = Collection([Box(-30,-32,0, 30,-52,-90),
                   Box(-30, 32,0, 30, 52,-90),
                   CylinderZ(0, 56,-15,56)
                   ])
yoke = Translate([yoke],-130,0,Lt+90)
yoke = Color([1,0.5,0,1],[yoke])

caxis = Collection([baxis,
                     yoke,
                     ])
caxis = Translate([caxis],130,0,0)
caxis = HalRotate([caxis],c,"joint5",1,0,0,1)
caxis = Translate([caxis],-130,0,0)

bmotor = CylinderZ(0,25,60,25)
bmotor = Translate([bmotor],20,30,0)
cmotor = CylinderZ(0,25,60,25)
cmotor = Translate([cmotor],20,-30,0)

zaxis = Collection([cmotor,bmotor,
                    CylinderZ(15,60,0,60),
                    Box(50,-60,-50, 80,60,60),
                    Box(0,-60,15, 50,60,0)
                    ])

zaxis = Translate([zaxis],-130,0,Lt+90)
zaxis = Color([0,1,1,1],[zaxis])
zaxis = Collection([caxis,zaxis,
                    ])

zaxis = Translate([zaxis], 0,0,-150)

zaxis = HalTranslate([zaxis],c,"joint2",0,0,1)

zmotor = CylinderZ(150,25,90,25)
zmotor = Translate([zmotor],10,0,0)
saddle = Collection([zaxis,zmotor,
                     Box(-10,-50,-100, -50,50,150)
                     ])

saddle = HalTranslate([saddle],c,"joint1",0,1,0)

ybridge = Collection([
                      Box(-60,255,-350, 60,285,0),
                      Box(-60,-255,-350, 60,-285,0),
                      Box(-30,-280,-100, 30,280,0),
                      Box(-70,255,-350, 70,295,-320),
                      Box(-70,-255,-350, 70,-295,-320),
                      ])
```

```
ybridge = Color([0,1,0,1],[ybridge])
ybridge = Collection([saddle,ybridge])

ybridge = HalTranslate([ybridge],c,"joint0",1,0,0)

xaxes = Collection([
                    Box(-500,-300,-400, 500, -250,-350),
                    Box(-500, 300,-400, 500,  250,-350)
                    ])
xaxes = Color([0,0,1,1],[xaxes])
router = Collection([xaxes,ybridge])

work = Capture()
table = Collection([
work,
Box(-500,-250,-320, 500,250,-365)
        ])

model = Collection([router, table])

main(model, tooltip, work, 1500)
```

14.4   Test NC Program

Here is a simple test nc program to show the toolpath on an inclined and rotated plane and the effect of a tool change in the XYZBC configuration. Change axis labels B to A to run it with the XYZAC configuration. The tool change position can be set before running the gcode using a G30.1 command.

```
%
(test-XYZBCkins with offsets and tool change)
( 3/11/2016 21:42:22 )
( 5 AXIS XYZBC MILLING )

N40 G00 G17 G40 G90 G94 Z10
(first tool - no z-offset in tool table)
N45 T1 M6
N50 S500 M3
N52 G00 B20 C30
N55 G43 H1 Z0
N60 G01 X20 F200
N65 G01 Y20
N70 G01 X0
N75 G03 X-20 Y0 I0 J-20
N76 G00 Z10
N77 G00 X0
N78 M5
(next tool- with z-offset in tool table)
N80 T2 M6
N81 S600 M3
N82 G00 B20 C30
N85 G43 H2 Z0
N90 G01 X-20 F200
N95 G01 Y-20
N100 G01 X0
N109 G03 X20 Y0 I0 J20
N110 G0 Z10
N112 G0 X0 Y0
N115 M5
N120 B0 C0
N125 M30
%
```