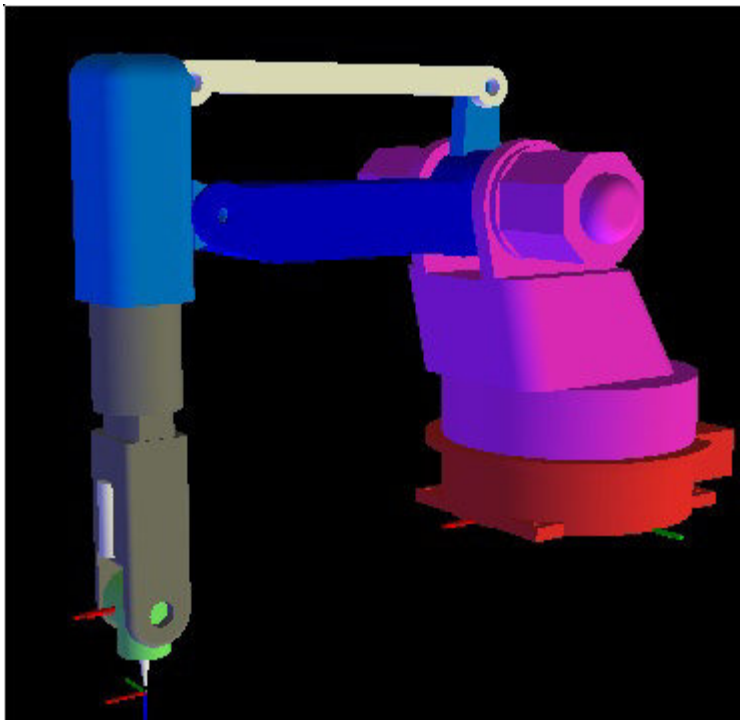

3D 6-DOF SERIAL ARM ROBOT - Kinematics and Implementation in LinuxCNC

Prof. Rudy du Preez
SA-CNC-CLUB

February 5, 2014

1 3D ROBOT ARM

In this section we deal with the typical serial arm robot in 3-dimensional space with six joints or degrees-of-freedom.

With such an arm you can measure positions using *forward kinematics* or you can perform machining operations using *forward and inverse kinematics*.

A typical serial arm robot is shown on the front page, and also schematically in Fig. 7-10. In references [2-4] there are many examples of such robots with various configurations.

In general a 3D robot arm has n joints and $n+1$ links. Numbering of links starts with 0 for the *fixed base* link to n for the *end-effector* link. Numbering of the joints start with 1 for the joint connecting the first movable link to the base link, and increases sequentially up to n. Therefor the link i is connected between the lower link $i-1$ by joint i and the next link $i+1$ by joint $i+1$, as shown in Fig. 1.

Fig. 1 shows the links $i-1$, i and $i+1$ of a serial robot, along with joints $i-1$, i and $i+1$. Every joint has an axis, which may be translational or rotational. To relate the kinematic information of the robot components, we attach a local coordinate frame F_i to each link i at joint $i+1$ based on the *Denavit-Hartenberg* (DH) method (see [2-4]):

1. The z_i-axis is aligned with the $i+1$ joint axis.
2. The x_i-axis is defined along the *common normal* between the z_{i-1} and z_i axes, pointing from the z_{i-1} to the z_i-axis.
3. The y_i-axis is determined by the right-hand rule.

In the DH method the origin o_i of the frame $F_i(o_i, x_i, y_i, z_i)$ attached to link i is placed at the *intersection* of the joint axis $i+1$ with the common normal between the z_{i-1} and z_i axes.

Four parameters $(a_i, \alpha_i, d_i, \theta_i)$, called the DH parameters, are used to define the geometry of a link i:

1. *Link length* a_i: the distance along the x_i-axis between the z_{i-1} and z_i axes.
2. *Link twist* α_i: the rotation angle about the x_i-axis between the z_{i-1} and z_i axes.
3. *Joint distance* d_i: the distance along the z_{i-1}-axis between the x_{i-1} and x_i axes.
4. *Joint angle* θ_i: the rotation angle about the z_{i-1}-axis between the x_{i-1} and x_i axes.

Figs. 7-10 show how the DH parameters are applied to a typical robot.

The DH parameters are usually defined for a specific robot in a DH-table, for example:

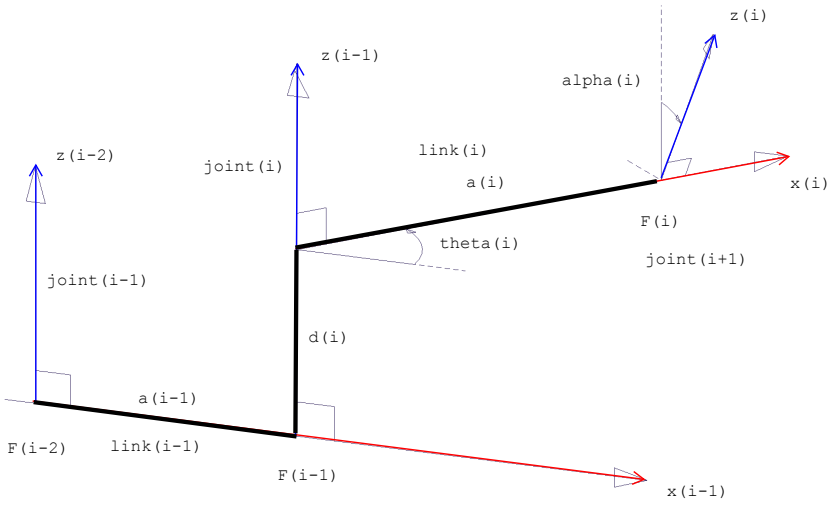

Figure 1: Links and DH parameters

Link	a_i	α_i	d_i	θ_i
1	a_1	α_1	d_1	θ_1
2	a_2	α_2	d_2	θ_2
...	...	...	...	...
j	a_j	α_j	d_j	θ_j
...	...	...	...	...
n	a_n	α_n	d_n	θ_n

For the robot depicted in Fig. 2 the DH parameters can be reduced to:

Link	a_i	α_i	d_i	θ_i
1	0	90	0	θ_1
2	a_2	0	d_2	θ_2
3	a_3	90	0	θ_3
4	0	-90	d_4	θ_4
5	0	90	0	θ_5
6	0	0	d_6	θ_6

1.1 Forward Transformation for a Link

The coordinate frame F_i is fixed to link i and frame F_{i-1} is fixed to link $i-1$. To transform F_i to $F_i - 1$ we can define a transformation matrix ${}^{i-1}T_i$ as follows, using the DH parameters if link i:

$$ {}^{i-1}T_i = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

or in abbreviated form

$$ {}^{i-1}T_i = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

This is a *homogeneous* transformation matrix, and has an *inverse* of the form:

$$ {}^{i}T_{i-1} = ({}^{i-1}T_i)^{-1} = \begin{bmatrix} c\theta_i & s\theta_i & 0 & -a_i \\ -s\theta_i c\alpha_i & c\theta_i c\alpha_i & s\alpha_i & -d_i s\alpha_i \\ s\theta_i s\alpha_i & -c\theta_i s\alpha_i & c\alpha_i & -d_i c\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

If we define the frame $F_{i-1} = (x_{i-1}, y_{i-1}, z_{i-1})$ and frame $F_i = (x_i, y_i, z_i)$ then we may write

$$ \begin{bmatrix} x_{i-1} \\ y_{i-1} \\ z_{i-1} \\ 1 \end{bmatrix} = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{3} $$

or the inverse form

$$ \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \begin{bmatrix} c\theta_i & s\theta_i & 0 & -a_i \\ -s\theta_i c\alpha_i & c\theta_i c\alpha_i & s\alpha_i & -d_i s\alpha_i \\ s\theta_i s\alpha_i & -c\theta_i s\alpha_i & c\alpha_i & -d_i c\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{i-1} \\ y_{i-1} \\ z_{i-1} \\ 1 \end{bmatrix} \tag{4} $$

To transform forward over a number of links, we just have to multiply the link matrices sequentially, ie.

$$ {}^{0}T_1 \cdot {}^{1}T_2 \cdot {}^{2}T_3 \cdots {}^{n-1}T_n = {}^{0}T_n \tag{5} $$

1.2 Forward Transformation of Typical 6-axis Robot

To simplify the equations somewhat we will present the equations for a typical 6-axis robot. The configuration is chosen so that in will include types such as PUMA, ABB, MOTOMAN, KUKA and FANUC. For this robot, depicted in Fig. 2, the DH parameters can be reduced to:

Link	a_i	α_i	d_i	θ_i
1	a_1	90	d_1	θ_1
2	a_2	0	d_2	θ_2
3	a_3	90	0	θ_3
4	0	−90	d_4	θ_4
5	0	90	0	θ_5
6	0	0	d_6	θ_6

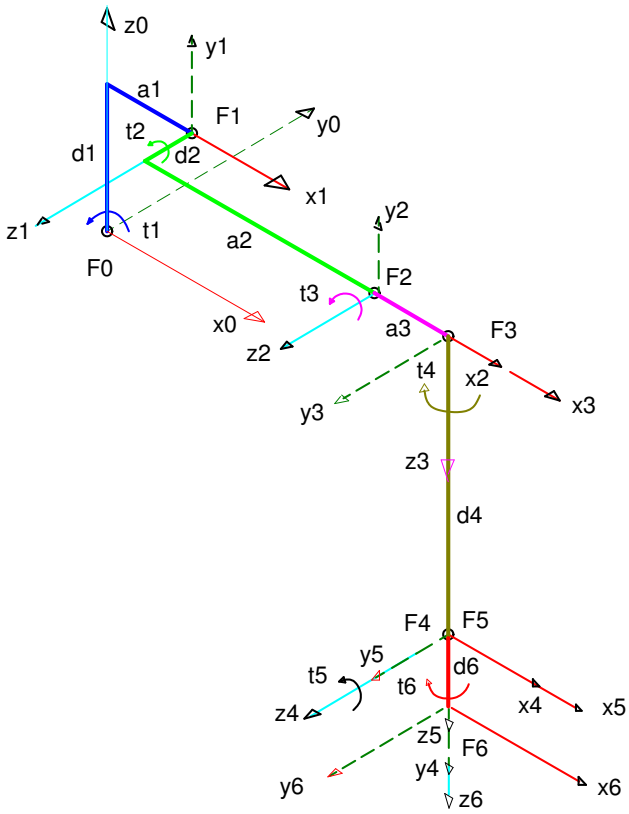

Figure 2: Robot links and DH parameters- zero position

With these parameters the T matrices of the robot are obtained as:

$$ {}^0T_1 = \begin{bmatrix} c_1 & 0 & s_1 & a_1c_1 \\ s_1 & 0 & -c_1 & a_1s_1 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^1T_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^2T_3 = \begin{bmatrix} c_3 & 0 & s_3 & a_3c_3 \\ s_3 & 0 & -c_3 & a_3s_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6} $$

$$ {}^3T_4 = \begin{bmatrix} c_4 & 0 & -s_4 & 0 \\ s_4 & 0 & c_4 & 0 \\ 0 & -1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^4T_5 = \begin{bmatrix} c_5 & 0 & s_5 & 0 \\ s_5 & 0 & -c_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^5T_6 = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7} $$

Note that in ${}^5T_6$ we have set $d_6 = 0$ since we first want to transform only up to the wrist. Following Paul and Zhang [1] we also determine the product ${}^1T_3 = {}^1T_2 \cdot {}^2T_3$:

$$ {}^1T_3 = \begin{bmatrix} c_{23} & 0 & s_{23} & a_2c_2 + a_3c_{23} \\ s_{23} & 0 & -c_{23} & a_2s_2 + a_3s_{23} \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8} $$

The position and orientation of the wrist are given by

$$ W_6 = {}^0T_1 \cdot {}^1T_3 \cdot {}^3T_4 \cdot {}^4T_5 \cdot {}^5T_6 \tag{9} $$

We evaluate this product from right to left and define intermediate matrices U_i which will also be used for the inverse transformation.

$$\begin{aligned} U_6 &= {}^5T_6, \quad U_5 = {}^4T_5 \cdot U_6, \quad U_4 = {}^3T_4 \cdot U_5, \\ U_3 &= {}^2T_3 \cdot U_4, \quad U_2 = {}^1T_3 \cdot U_4, \quad U_1 = {}^0T_1 \cdot U_2 \end{aligned} \tag{10}$$

Thus

$$U_6 = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad U_5 = \begin{bmatrix} c_5c_6 & -c_5s_6 & s_5 & 0 \\ s_5c_6 & -s_5s_6 & -c_5 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

$$U_4 = \begin{bmatrix} c_4U_{511} - s_4s_6 & c_4U_{512} - s_4c_6 & c_4s_5 & 0 \\ s_4U_{511} + c_4s_6 & s_4U_{512} + c_4c_6 & s_4s_5 & 0 \\ -U_{521} & -U_{522} & c_5 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

$$U_3 = \begin{bmatrix} c_3U_{411} - s_3U_{521} & c_3U_{412} - s_3U_{522} & c_3U_{413} + s_3c_5 & d_4s_3 + a_3c_3 \\ s_3U_{411} + c_3U_{521} & s_3U_{412} + c_3U_{522} & s_3U_{413} - c_3c_5 & -d_4c_3 + a_3s_3 \\ U_{421} & U_{422} & U_{423} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

$$U_2 = \begin{bmatrix} c_{23}U_{411} - s_{23}U_{521} & c_{23}U_{412} - s_{23}U_{522} & c_{23}U_{413} + s_{23}c_5 & +d_4s_{23} + a_3c_{23} + a_2c_2 \\ s_{23}U_{411} + c_{23}U_{521} & s_{23}U_{412} + c_{23}U_{522} & s_{23}U_{413} - c_{23}c_5 & -d_4c_{23} + a_3s_{23} + a_2s_2 \\ U_{421} & U_{422} & U_{423} & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$U_1 = \begin{bmatrix} c_1U_{211} + s_1U_{421} & c_1U_{212} + s_1U_{422} & c_1U_{213} + s_1U_{423} & c_1U_{214} + d_2s_1 + a_1c_1 \\ s_1U_{211} - c_1U_{421} & s_1U_{212} - c_1U_{422} & s_1U_{213} - c_1U_{423} & s_1U_{214} - d_2c_1 + a_1s_1 \\ U_{221} & U_{222} & U_{223} & U_{224} + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

If we define E_6, the end-effector matrix as

$$E_6 = \begin{bmatrix} u_x & v_x & w_x & q_x \\ u_y & v_y & w_y & q_y \\ u_z & v_z & w_z & q_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

then the wrist position p_x, p_y, p_z can be found using

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} q_x - d_6w_x \\ q_y - d_6w_y \\ q_z - d_6w_z \\ 1 \end{bmatrix} \tag{17}$$

so that the wrist orientation and position is defined by

$$W_6 = \begin{bmatrix} u_x & v_x & w_x & p_x \\ u_y & v_y & w_y & p_y \\ u_z & v_z & w_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = U_1 \tag{18}$$

The components of W_6 can be obtained directly from (15). Note that we used local variables U_{ijk} where i refers to the ith U matrix and j and k refer to the jth row and kth column of U_i, which in this case is the same position as the wrist of the robot arm.

The variables $u_x, u_y, u_z, v_x, v_y, v_z, w_x, w_y, w_z$ are the direction cosines of the end-effector and wrist axes (defining its orientation), while p_x, p_y, p_z are the cartesian coordinates of the wrist. The position of the end-effector some distance d_6 away from the wrist can be easily determined separately and is dealt with later.

These are the required results of a complete forward transformation, given all the DH parameters of the 6 links of our typical robot (except for the moment d_6).

1.3 Inverse Transformation of the Robot

We now want to solve for the joint angles θ_1 to θ_6 given the wrist position and orientation defined by W_6. Following [1] we first obtain a sequence of V_i transformations by successively pre-multiplying W_6 by each of the $({}^{i-1}T_i)^{-1}$ inverse matrices.

We start with

$$W_6 = V_0 = U_1 \tag{19}$$

and then consecutively

$$\begin{aligned} ({}^0T_1)^{-1} \cdot W_6 &= V_1 = U_2, \\ ({}^1T_2)^{-1} \cdot V_1 &= V_2 = U_3, \\ ({}^1T_3)^{-1} \cdot V_1 &= V_3 = U_4, \\ ({}^3T_4)^{-1} \cdot V_3 &= V_4 = U_5, \\ ({}^4T_5)^{-1} \cdot V_4 &= V_5 = U_6. \end{aligned} \tag{20}$$

We can write V_0 in a generic way

$$V_{0j} = \begin{bmatrix} X \\ Y \\ Z \\ M \end{bmatrix} \tag{21}$$

where V_{0jk} are obtained by substituting the components of the k-th column of E_6 for X, Y, Z, M with $M = 0$ for $k = 1, 2, 3$ and $M = 1$ for $k = 4$.

Pre-multiplying W_6 by $({}^0T_1)^{-1}$ we obtain:

$$V_{1j} = \begin{bmatrix} c_1 & s_1 & 0 & -a_1 \\ 0 & 0 & 1 & -d_1 \\ s_1 & -c_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ M \end{bmatrix} = \begin{bmatrix} c_1X + s_1Y - a_1M \\ Z - d_1M \\ s_1X - c_1Y \\ M \end{bmatrix} \tag{22}$$

Pre-multiplying V_1 by $({}^1T_2)^{-1}$ we obtain:

$$V_{2j} = \begin{bmatrix} c_2 & s_2 & 0 & -a_2 \\ -s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & -d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_{11} \\ V_{12} \\ V_{13} \\ M \end{bmatrix} = \begin{bmatrix} c_2V_{11} + s_2V_{12} - a_2M \\ -s_2V_{11} + c_2V_{12} \\ V_{13} - d_2M \\ M \end{bmatrix} \tag{23}$$

Pre-multiplying V_1 by $({}^1T_3)^{-1}$ we obtain:

$$V_{3j} = \begin{bmatrix} c_{23} & s_{23} & 0 & -(a_2c_3 + a_3) \\ 0 & 0 & -1 & -d_2 \\ s_{23} & -c_{23} & 0 & -a_2s_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_{11} \\ V_{12} \\ V_{13} \\ M \end{bmatrix} = \begin{bmatrix} c_{23}V_{11} + s_{23}V_{12} - (a_2c_3 + a_3)M \\ V_{13} - d_2M \\ s_{23}V_{11} - c_{23}V_{12} - a_2s_3M \\ M \end{bmatrix} \tag{24}$$

Pre-multiplying V_3 by $({}^3T_4)^{-1}$ we obtain:

$$V_{4j} = \begin{bmatrix} c_4 & s_4 & 0 & 0 \\ 0 & 0 & -1 & d_4 \\ -s_4 & +c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_{31} \\ V_{32} \\ V_{33} \\ M \end{bmatrix} = \begin{bmatrix} c_4V_{31} + s_4V_{32} \\ -V_{33} + d_4M \\ -s_4V_{31} + c_4V_{32} \\ M \end{bmatrix} \tag{25}$$

Finally , pre-multiplying V_4 by $({}^4T_5)^{-1}$ we obtain:

$$V_{5j} = \begin{bmatrix} c_5 & s_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ s_5 & -c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_{41} \\ V_{42} \\ V_{43} \\ M \end{bmatrix} = \begin{bmatrix} c_5V_{41} + s_5V_{42} \\ V_{43} \\ s_5V_{41} - c_5V_{42} \\ M \end{bmatrix} \tag{26}$$

The solution is now found by equating $V_0 = U_1, V_1 = U_2, V_2 = U_3, V_3 = U_4, V_4 = U_5$ and $V_5 = U_6$. Equating V_0 and U_1:

$$\begin{bmatrix} X \\ Y \\ Z \\ M \end{bmatrix} = \begin{bmatrix} c_1U_{211} + s_1U_{421} & c_1U_{212} + s_1U_{422} & c_1U_{213} + s_1U_{423} & c_1U_{214} + d_2s_1 + a_1c_1 \\ s_1U_{211} - c_1U_{421} & s_1U_{212} - c_1U_{422} & s_1U_{213} - c_1U_{423} & s_1U_{214} - d_2c_1 + a_1s_1 \\ U_{221} & U_{222} & U_{223} & U_{224} + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{27}$$

To solve for θ_1 we equate the (1,4) and (2,4) elements:

$$\begin{aligned} p_x &= c_1(U_{214} + a_1) + d_2s_1 \\ p_y &= s_1(U_{214} + a_1) - d_2c_1 \end{aligned} \tag{28}$$

Multiplying the first by s_1 and the second by c_1, and subtracting the second from the first, we find:

$$s_1p_x - c_1p_y = d_2 \tag{29}$$

This equation can be solved by the substitutions:

$$rsin\phi = p_y, \quad rcos\phi = p_x, \quad \text{with} \quad r > 0 \tag{30}$$

Then

$$r = +\sqrt{p_x^2 + p_y^2} \quad \text{and} \quad \phi = tan^{-1}\frac{p_y}{p_x} \tag{31}$$

In a computer program we would use the *atan2* function which returns values in the range $-\pi \leq \phi < \pi$. Making the substitution into (29) we obtain

$$s_1 cos\phi - c_1 sin\phi = sin(\theta_1 - \phi) = \frac{d_2}{r} \tag{32}$$

and

$$\theta_1 = tan^{-1}\frac{p_y}{p_x} + sin^{-1}\frac{d_2}{r} \quad \text{or} \quad \theta_1 = tan^{-1}\frac{p_y}{p_x} - sin^{-1}\frac{d_2}{r} + \pi \tag{33}$$

since the sin^{-1} function is double valued. The two solutions correspond to the shoulder of the robot appearing on the right- and left-hand sides respectively. For a solution to exist $r > d_2$.

Next, to solve for θ_2 we equate V_1 with U_2.

$$\begin{bmatrix} V_{11} \\ V_{12} \\ V_{13} \\ M \end{bmatrix}$$

$$= \begin{bmatrix} c_{23}U_{411} - s_{23}U_{521} & c_{23}U_{412} - s_{23}U_{522} & c_{23}U_{413} + s_{23}c_5 & d_4 s_{23} + a_3 c_{23} + a_2 c_2 \\ s_{23}U_{411} + c_{23}U_{521} & s_{23}U_{412} + c_{23}U_{522} & s_{23}U_{413} - c_{23}c_5 & -d_4 c_{23} + a_3 s_{23} + a_2 s_2 \\ U_{421} & U_{422} & U_{423} & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{34}$$

Again we equate elements (1,4) and (2,4) to obtain:

$$\begin{aligned} V_{114} &= d_4 s_{23} + a_3 c_{23} + a_2 c_2 \\ V_{124} &= -d_4 c_{23} + a_3 s_{23} + a_2 s_2 \end{aligned}$$

or re-arranging:

$$\begin{aligned} V_{114} - a_2 c_2 &= d_4 s_{23} + a_3 c_{23} \\ V_{124} - a_2 s_2 &= -d_4 c_{23} + a_3 s_{23} \end{aligned} \tag{35}$$

Squaring both sides and adding, we obtain

$$c_2 V_{114} + s_2 V_{124} = \frac{a_2^2 - d_4^2 - a_3^2 + V_{114}^2 + V_{124}^2}{2a_2} \tag{36}$$

Substituting again with

$$r sin\phi = V_{124}, \quad r cos\phi = V_{114}, \quad \text{with} \quad r > 0 \tag{37}$$

then

$$r = +\sqrt{V_{114}^2 + V_{124}^2} \quad \text{and} \quad \phi = tan^{-1}\frac{V_{124}}{V_{114}} \tag{38}$$

We substitute (37) into (36) and get

$$cos(\theta_2 - \phi) = \frac{a_2^2 - d_4^2 - a_3^2 + V_{114}^2 + V_{124}^2}{2a_2 r} = \kappa \tag{39}$$

If the value of $\kappa > 1$ then the defined position is beyond the reach of the robot. Defining

$$\psi = cos^{-1}(\kappa) \tag{40}$$

then

$$\theta_2 = tan^{-1}\frac{V_{124}}{V_{114}} \pm \psi \tag{41}$$

If the $+\psi$ solution is used for a right-handed configuration of joint 1, then the elbow of the robot is above the line from the shoulder to the wrist, whereas for the $-\psi$ solution it is below the line. The situation is the opposite for a left-handed configuration of joint 1.

The next joint angle θ_3 is obtained by equating V_2 with U_3:

$$\begin{bmatrix} V_{21} \\ V_{22} \\ V_{23} \\ M \end{bmatrix} = \begin{bmatrix} U_{311} & U_{312} & U_{313} & d_4s_3 + a_3c_3 \\ U_{321} & U_{322} & U_{323} & -d_4c_3 + a_3s_3 \\ U_{331} & U_{332} & U_{333} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{42}$$

Equating elements (1,4) and (2,4) again we obtain

$$\begin{aligned} V_{214} &= d_4s_3 + a_3c_3 \\ V_{224} &= -d_4c_3 + a_3s_3 \end{aligned} \tag{43}$$

As before, we substitute

$$a_3 = rsin\phi, \quad d_4 = rcos\phi \quad \text{with} \quad r > 0 \tag{44}$$

then

$$r = +\sqrt{a_3^2 + d_4^2} \quad \text{and} \quad \phi = tan^{-1}\frac{a_3}{d_4} \tag{45}$$

so that

$$\begin{aligned} s_3cos\phi + c_3sin\phi &= sin(\phi + \theta_3) = V_{214}/r \\ -c_3cos\phi + s_3sin\phi &= -cos(\phi + \theta_3) = V_{224}/r \end{aligned}$$

from which we obtain

$$tan(\phi + \theta_3) = \frac{V_{214}}{-V_{224}}, \qquad \phi + \theta_3 = tan^{-1}\frac{V_{214}}{-V_{224}} \tag{46}$$

and

$$\theta_3 = tan^{-1}\frac{V_{214}}{-V_{224}} - tan^{-1}\frac{a_3}{d_4} \tag{47}$$

We next obtain θ_4 from $V_3 = U_4$ or

$$\begin{bmatrix} V_{31} \\ V_{32} \\ V_{33} \\ M \end{bmatrix} = \begin{bmatrix} U_{411} & U_{412} & c_4s_5 & 0 \\ U_{421} & U_{422} & s_4s_5 & 0 \\ U_{431} & U_{432} & c_5 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{48}$$

so that

$$V_{313} = c_4s_5, \quad V_{323} = s_4s_5 \quad \text{or} \quad tan\theta_4 = \frac{V_{323}}{V_{313}} \tag{49}$$

One again there are two solutions depending on the sign of s_5:

$$\begin{aligned} \theta_4 &= tan^{-1}\frac{V_{323}}{V_{313}} \quad \text{if} \quad s_5 > 0 \\ \theta_4 &= tan^{-1}\frac{V_{323}}{V_{313}} \quad \text{if} \quad s_5 < 0 \end{aligned} \tag{50}$$

and θ_4 is undefined or arbitrary if $s_5 = 0$. To evaluate V_{313} and V_{323} we may use

$$\begin{aligned} V_{113} &= c_1 w_x + s_1 w_y \\ V_{313} &= c_{23} V_{113} + s_{23} V_{123} = c_{23} V_{113} + s_{23} w_z \\ V_{323} &= V_{133} = s_1 w_x - c_1 w_y \end{aligned} \tag{51}$$

For the next angle we use $V_4 = U_5$ or

$$\begin{bmatrix} c_4 V_{31} + s_4 V_{32} \\ -V_{33} + d_4 M \\ -s_4 V_{31} + c_4 V_{32} \\ M \end{bmatrix} = \begin{bmatrix} U_{511} & U_{512} & s_5 & 0 \\ U_{521} & U_{522} & -c_5 & 0 \\ U_{531} & U_{532} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{52}$$

from which

$$\begin{aligned} s_5 &= c_4 V_{313} + s_4 V_{323} \\ c_5 &= s_{23} V_{113} - c_{23} V_{123} = s_{23} V_{113} - c_{23} w_z \end{aligned} \tag{53}$$

so that

$$\theta_5 = tan^{-1}\frac{s_5}{c_5} \tag{54}$$

Finally with $V_5 = U_6$:

$$\begin{bmatrix} c_5 V_{41} + s_5 V_{42} \\ V_{43} \\ s_5 V_{41} - c_5 V_{42} \\ M \end{bmatrix} = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{55}$$

from which

$$\begin{aligned} s_6 &= V_{431} \\ c_6 &= c_5 V_{411} + s_5 V_{421} \end{aligned} \tag{56}$$

so that

$$\theta_6 = tan^{-1}\frac{s_6}{c_6} \tag{57}$$

The components can be evaluated using:

$$\begin{aligned} V_{111} &= c_1 u_x + s_1 u_y \\ V_{131} &= s_1 u_x - c_1 u_y \\ V_{311} &= c_{23} V_{111} + s_{23} u_z \\ V_{331} &= s_{23} V_{111} - c_{23} u_z \\ V_{411} &= c_4 V_{311} + s_4 V_{131} \\ V_{421} &= -V_{331} \\ V_{431} &= -s_4 V_{311} + c_4 V_{131} \end{aligned} \tag{58}$$

1.4 End-effector Position

We have left out the position of the end-effector from the kinematic transformations treated above. Its position q_x, q_y, q_z can be determined from the wrist position p_x, p_y, p_z by a forward transformation:

$$\begin{bmatrix} q_x \\ q_y \\ q_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + d_6 w_x \\ p_y + d_6 w_y \\ p_z + d_6 w_z \\ 1 \end{bmatrix} \tag{59}$$

For the inverse transformation, given q_x, q_y, q_z and E_6 we find

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} q_x - d_6 w_x \\ q_y - d_6 w_y \\ q_z - d_6 w_z \\ 1 \end{bmatrix} \tag{60}$$

1.5 End-effector Orientation

There are a number of ways to describe the orientation of the end-effector: using the homogeneous matrix E_6 defined above, or using rotation about selected axes.

A common method is to use *roll-pitch-yaw* angles denoted as ϕ_r, ϕ_p, ϕ_y for rotations at the end-effector position q_x, q_y, q_z about x, y, z axes which are parallel to the fixed base axes.

Given ϕ_r, ϕ_p, ϕ_y we can determine the E_6 matrix, ie. the matrix which defines the position and orientation of the end-effector at q_x, q_y, q_z as:

$$E_7 = \begin{bmatrix} u_x & v_x & w_x & q_x \\ u_y & v_y & w_y & q_y \\ u_z & v_z & w_z & q_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_p c_y & s_r s_p c_y - c_r s_y & c_r s_p c_y + s_r s_y & q_x \\ c_p s_y & s_r s_p s_y + c_r c_y & c_r s_p s_y - s_r c_y & q_y \\ -s_p & s_r c_p & c_r c_p & q_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{61}$$

where $s_r = sin\phi_r, \quad c_r = cos\phi_r, \quad s_y = sin\phi_y, \quad c_p = cos\phi_p$, etc.

The roll, pitch and yaw angles can also be determined from a given E_6 matrix as follows, for ϕ_p not equal to ± 90 degrees:

$$\begin{aligned} \phi_r &= \text{atan2}(v_z, w_z) \\ \phi_p &= \text{atan2}(-u_z, \sqrt{u_x \cdot u_x + u_y \cdot u_y}) \\ \phi_y &= \text{atan2}(u_y, u_x) \end{aligned} \tag{62}$$

If $\phi_p = \pm 90$ degrees, then for $\phi_p = 90$:

$$\phi_r = \text{atan2}(v_z, w_z), \quad \phi_p = \pi/2, \quad \phi_y = 0 \tag{63}$$

and for $\phi_p = -90$:

$$\phi_r = -\text{atan2}(v_z, w_z), \quad \phi_p = -\pi/2, \quad \phi_y = 0 \tag{64}$$

1.6 Configuration

Given the joint angles and the end-effector matrix E_6 of a PUMA type robot, the configuration can be determined as follows

- Configuration *left* or *right* can be determined by the sign of

$$\theta_1 = tan^{-1}\frac{p_y}{p_x} - \frac{\pi}{2} \tag{65}$$

  A positive sign indicates a *left* and a negative sign a *right* shoulder.

- Configuration *elbow-up* or *elbow-down* can be determined by the sign of

$$\theta_2 = tan^{-1}\frac{V_{124}}{V_{114}} \tag{66}$$

  If the shoulder is *left* a positive sign here corresponds to *elbow-down* and a negative sign to *elbow-up*.

  If the shoulder is *right* a positive sign here corresponds to *elbow-up* and a negative sign to *elbow-down*.

- Configuration *wrist-flip* corresponds to a negative $sin\theta_5$ and *wrist-no-flip* to a positive $sin\theta_5$.

1.7 Tool Movement

If we think of a tool being at the tip of the last link of the robot, ie. a welding or cutting torch, a router cutter or a spray gun, then we can use the robot to perform such operations. We would then need to describe a tool path in some way.

Three methods seems to be generally used:

- The tool path is achieved by movement of the joints, ie. by describing joint angles θ_i as functions of time. This is the so-called ”joint mode”. This is however in applications such as mentioned above not very useful.

- The tool path is achieved by describing the tool position and orientation in a predefined frame or coordinate system which is parallel to the fixed base frame of the robot. The base frame of the robot, ie. F1 is at joint 1 with coordinates X, Y, Z. These are also called ”world coordinates” and we then operate in ”world mode”.

- A third possibility is similar to the previous case but with a variable coordinate system that can be set by positioning first in ”world mode” or ”joint mode” and then setting that state as the current ”local” or ”tool” position and orientation (local coordinates x, y, z). This can be called ”local mode” or ”tool mode” since we typically work in a coordinate frame aligned with the current ”tool” attached to the end link of the robot.

The first case of ”joint mode” is typically only used to position the robot arm before starting an operation and this is done by ”jogging” using buttons or an MPG (not by a GCODE program). Forward kinematics is used in this case to determine the coordinates of the tool position and orientation after a ”joint mode” move.

The other two methods of path control are briefly discussed below. They make use of inverse kinematics.

1.8 World mode path control

In this mode of control, the move from one position and orientation, called ”pose”, to another pose is done by a command to the CNC controller (such as LinuxCNC) is given by jogging or Gcode commands in ”world coordinates”, for example:

G0 X100	move from current position to X=100, at rapid speed parallel to ”world” X-axis.
G1 Y-20 F40	move from current position to Y=-20, at 40 mm/min parallel to ”world” Y-axis.
G1 C20 F180	rotate tool from current position, at 180 deg/min around local z-axis parallel to ”world” Z-axis.
G2 X30 Y0 R15	move from current position in the X-Y plane (G17 default) along an arc with radius 15mm to the end point X30, Y0, clockwise

Note that the third command is for a rotation in the X-Y plane around the axis normal to the plane *at the current position*. The rotation axis is a local axis parallel to the base frame axis or ”world” coordinate axis Z. The local axis is at the current position of the tool point.

It is convenient to work with a local coordinate frame which is still parallel to the fixed base frame but with the origin re-positioned at a convenient point (on a work table or on the object being operated on). This can be done with a G54 offset using the ”touch off” facility in AXIS. Subsequent movements are then in terms of ”relative” or ”offset” coordinates

If the ”offset” pose is indicated as $p_o(X_o, Y_o, Z_o, A_o, B_o, C_o)$ and the new pose after the move $p(X_p, Y_p, Z_p, A_p, B_p, C_p)$ is then:

$$\begin{aligned} X_p &= X_o + \Delta X \\ Y_p &= Y_o + \Delta Y \\ Z_p &= Z_o + \Delta Z \\ A_p &= A_o + \Delta A \\ B_p &= B_o + \Delta B \\ C_p &= C_o + \Delta C \end{aligned} \tag{67}$$

Working in this ”world” mode is what comes standard with most multi-axis implementations in LinuxCNC.

In Fig. 3 such a ”world” mode move is shown in 2 dimensions for clarity.

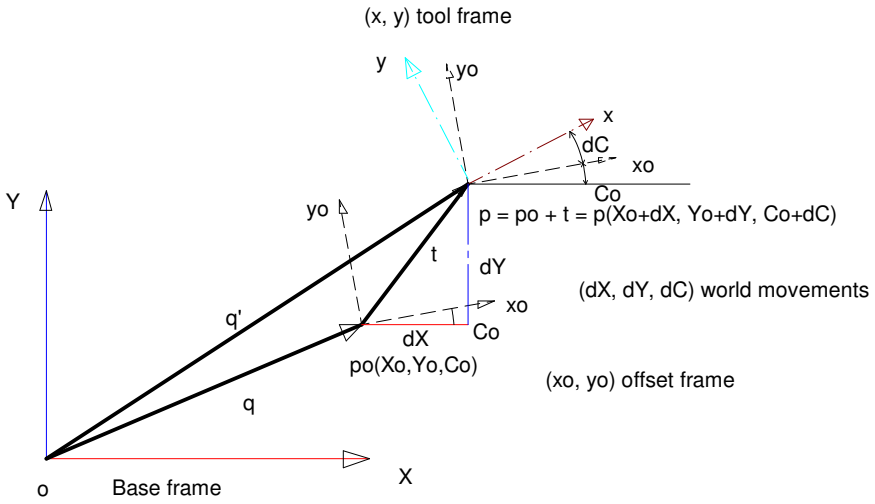

Figure 3: World coordinate move in the X-Y plane

1.9 Tool mode path control

In this mode of control, a change is made from the ”world” coordinate frame to a ”tool” coordinate frame at some convenient point, so that the following X, Y,$\cdots$ C movements (jogging or Gcode) are now in the new coordinate system. The new coordinate frame can be positioned and rotated relative to the world coordinate frame.

Typical situations are where we need to keep the tool normal or parallel to some plane or line which is not parallel to the ”world” frame.

The first step to work in this mode is to move the tool to a selected point and rotate the tool coordinate frame into the new offset working frame. After setting this frame as the offset frame all further movements will be relative to this frame. The offset frame can be defined with the position and orientation relative to the world or base coordinate system as $X_o, Y_o, Z_o, A_o, B_o, C_o$ or as a matrix (refer to (61)):

$$ {}^{1}T_o = \begin{bmatrix} u_{Xo} & v_{Xo} & w_{Xo} & q_{Xo} \\ u_{Yo} & v_{Yo} & w_{Yo} & q_{Yo} \\ u_{Zo} & v_{Zo} & w_{Zo} & q_{Zo} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_B c_C & s_A s_B c_C - c_A s_C & c_A s_B c_C + s_A s_C & X_o \\ c_B s_C & s_A s_B s_C + c_A c_C & c_A s_B s_C - s_A c_C & Y_o \\ -s_B & s_A c_B & c_A c_B & Z_o \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{68} $$

where $s_A = sinA_o, \quad c_A = cosA_o$, etc.

The new tool move relative to this working or offset coordinate system is defined by $x_t, y_t, z_t, a_t, b_t, c_t$ and this can also be written in a transformation matrix:

$$ {}^{o}T_t = \begin{bmatrix} u_x & v_x & w_x & x_t \\ u_y & v_y & w_y & y_t \\ u_z & v_z & w_z & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_b c_c & s_a s_b c_c - c_a s_c & c_a s_b c_c + s_a s_c & x_t \\ c_b s_c & s_a s_b s_c + c_a c_c & c_a s_b s_c - s_a c_c & y_t \\ -s_b & s_a c_b & c_a c_b & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{69} $$

where $s_a = sina_t, \quad c_a = cosa_t$, etc.

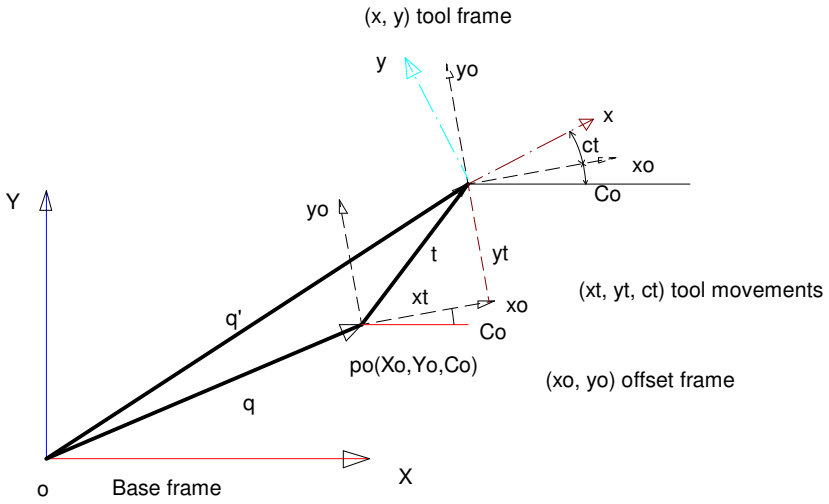

Figure 4: Tool coordinate move in the X-Y plane

Now with these two matrices we can get the effect of the new tool move in terms of the world coordinates by the product of the two:

$$ {}^{1}T_t = {}^{1}T_o \cdot {}^{o}T_t = \begin{bmatrix} u_{Xt} & v_{Xt} & w_{Xt} & X_t \\ u_{Yt} & v_{Yt} & w_{Yt} & Y_t \\ u_{Zt} & v_{Zt} & w_{Zt} & Z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{70} $$

The matrix ${}^{1}T_t$ contains all the information used as input to the inverse kinematics calculation as described in section 1.3.

1.10 Kinematics component

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
```

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 robot. This is done in the .hal file of your config directory. The standard command

```
loadrt trivkins
```

is replaced by

```
loadrt robot6kins
```

where ”robot6kins” is the name of our 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 DH parameters of the robot. In our 6-joint robot the entries could be:

```
# set robot DH parameters
setp robot6kins.DH-a1 200
setp robot6kins.DH-a2 600
setp robot6kins.DH-a3 110
setp robot6kins.DH-d1 450
setp robot6kins.DH-d2   0
setp robot6kins.DH-d4 620
setp robot6kins.DH-d6 150
```

An example of a kinematics component, applicable to our 6-joint serial planer robot, is given below. It has the following structure:

- A set of ”include” declarations which is need to link in other procedures from libraries
- A struct definition to define the ”hal” input links or pins. These are defined in your .hal file.
- Functions or procedures called from the subsequent procedures, if any.
- The ”kinematicsForward” procedure with its fixed parameter list (joints, pos, and flags).
- The ”kinematicsInverse” procedure with its fixed parameter list (pos, joints, and flags).
- The ”kinematicsType” procedure which defines that BOTH forward and inverse kins are used. There are also other possibilities
- The ”rtapi_app_main” main program with its own include declarations and in which the links to the HAL pins are defined.

```
/************************************************************************
* Description: robot6kins.c
*   Kinematics for  6 axis ABB,Fanuc,Kuka type robot.
*   This serial arm robot has the following DH parameters:
*   a1,a2,a3, d1,d2,d4,d6
*
* Author: Rudy du Preez (SA-CNC-CLUB)
* License: GPL Version 2
*
************************************************************************/
```

```
#include "kinematics.h"
#include "hal.h"
#include "rtapi.h"
#include "rtapi_math.h"


struct haldata {
    hal_float_t *DH_a1;
    hal_float_t *DH_a2;
    hal_float_t *DH_a3;
    hal_float_t *DH_d1;
    hal_float_t *DH_d2;
    hal_float_t *DH_d4;
    hal_float_t *DH_d6;
    hal_bit_t *tmode;
    hal_bit_t *soffs;
} *haldata;

double Ao = 0;
double Bo = 0;
double Co = 0;

double aw = 0;
double bw = 0;
double cw = 0;
double To[4][4];
double Tt[4][4];
double Tw[4][4];

int elbup = 1;
int wrflp = 0;
int wsing = 0;

double eps = 0.0001;

//=============================================================================
int fwABC2(double ux, double uy, double uz, double vx, double vy,
           double vz, double wz)
{
// to Craig: p40 RPY(c,b,a) => (Rz,Ry,Rz)

    bw = atan2(-uz, sqrt(ux*ux + uy*uy) );
    if (fabs(fabs(bw) - M_PI/2.0) > eps) {
       aw = atan2(vz, wz);
       cw = atan2(uy, ux);
    } else if (bw > 0.0) {
       aw = atan2(vx, vy);
```

```
       bw = M_PI/2.0;
       cw = 0;
    } else {
       aw = -atan2(vx, vy);
       bw = -M_PI/2.0;
       cw = 0;
    }
    if (aw < 0) {aw = aw + 2*M_PI;}

    return 0;
}
//=========================================================================
static void MatMult(double A[][4], double B[][4], double C[][4])
{
  int i, j, k;

  for (i=0; i<=3; ++i){
    for (j=0; j<=3; ++j){
      C[i][j] = 0;
      for (k=0; k<=3; ++k){
        C[i][j] = C[i][j] + A[i][k]*B[k][j];
      }
    }
  }
}
//=========================================================================
static void PoseMat(double X, double Y, double Z,
                    double A, double B, double C, double T[][4])
{
double cr, sr, cp, sp, cy, sy;

       sr = sin(A); cr = cos(A);
       sp = sin(B); cp = cos(B);
       sy = sin(C); cy = cos(C);

       T[0][0] = cp*cy;
       T[1][0] = cp*sy;
       T[2][0] = -sp;
       T[3][0] = 0;

       T[0][1] = sr*sp*cy - cr*sy;
       T[1][1] = sr*sp*sy + cr*cy;
       T[2][1] = sr*cp;
       T[3][1] = 0;

       T[0][2] = cr*sp*cy + sr*sy;
       T[1][2] = cr*sp*sy - sr*cy;
```

```
        T[2][2] = cr*cp;
        T[3][2] = 0;

        T[0][3] = X;
        T[1][3] = Y;
        T[2][3] = Z;
        T[3][3] = 1;
}
//=============================================================================
int kinematicsForward(const double *joint,
      EmcPose * pos,
      const KINEMATICS_FORWARD_FLAGS * fflags,
      KINEMATICS_INVERSE_FLAGS * iflags)
{
    double a1 = *(haldata->DH_a1);
    double a2 = *(haldata->DH_a2);
    double a3 = *(haldata->DH_a3);
    double d1 = *(haldata->DH_d1);
    double d2 = *(haldata->DH_d2);
    double d4 = *(haldata->DH_d4);
    double d6 = *(haldata->DH_d6);
    int tmode = *(haldata->tmode);
    int soffs = *(haldata->soffs);

    double th1 = joint[0]*M_PI/180;
    double th2 = joint[1]*M_PI/180;
    double th3 = joint[2]*M_PI/180;
    double th4 = joint[3]*M_PI/180;
    double th5 = joint[4]*M_PI/180;
    double th6 = joint[5]*M_PI/180;

    double c1, s1, c2, s2, c3, s3, c4, s4, c5, s5, c6, s6, c23, s23;
    double u511, u512, u521, u522, u411, u412, u413, u421, u422, u423;
    double u311, u312, u313, u314, u321, u322, u323, u324;
    double u211, u212, u213, u214, u221, u222, u223, u224, v114;
    double ux, uy, uz, vx, vy, vz, wx, wy, wz, qx, qy, qz;

    c1 = cos(th1);  s1 = sin(th1);
    c2 = cos(th2);  s2 = sin(th2);
    c3 = cos(th3);  s3 = sin(th3);
    c4 = cos(th4);  s4 = sin(th4);
    c5 = cos(th5);  s5 = sin(th5);
    c6 = cos(th6);  s6 = sin(th6);
    c23 = cos(th2+th3);  s23 = sin(th2+th3);

    u511 = c5*c6; u512 = -c5*s6;
    u521 = s5*c6; u522 = -s5*s6;
```

```
u411 = c4*u511 - s4*s6;
u412 = c4*u512 - s4*c6;
u413 = c4*s5;
u421 = s4*u511 + c4*s6;
u422 = s4*u512 + c4*c6;
u423 = s4*s5;

u311 = c3*u411 - s3*u521;
u312 = c3*u412 - s3*u522;
u313 = c3*u413 + s3*c5;
u314 = s3*d4 + c3*a3;

u321 = s3*u411 + c3*u521;
u322 = s3*u412 + c3*u522;
u323 = s3*u413 - c3*c5;
u324 = -c3*d4 + s3*a3;

u211 = c23*u411 - s23*u521;
u212 = c23*u412 - s23*u522;
u213 = c23*u413 + s23*c5;
u214 = s23*d4 + c23*a3 + c2*a2;

u221 = s23*u411 + c23*u521;
u222 = s23*u412 + c23*u522;
u223 = s23*u413 - c23*c5;
u224 = -c23*d4 + s23*a3 + s2*a2;
v114 = -s23*d4 + c23*a3 + c2*a2;

ux = c1*u211 + s1*u421;
uy = s1*u211 - c1*u421;
uz = u221;
vx = c1*u212 + s1*u422;
vy = s1*u212 - c1*u422;
vz = u222;
wx = c1*u213 + s1*u423;
wy = s1*u213 - c1*u423;
wz = u223;
qx = c1*u214 + s1*d2 + c1*a1 + d6*wx;
qy = s1*u214 - c1*d2 + s1*a1 + d6*wy;
qz = u224 + d1 + d6*wz;

 fwABC2(ux, uy, uz, vx, vy, vz, wz);

if (v114 > 0) {elbup = 1;} else {elbup = 0;}
if (s5 >= 0)  {wrflp = 0;} else {wrflp = 1;}
if (fabs(s5) < eps)   {wsing = 1;} else {wsing = 0;}
```

```
    if (soffs) {
//
      To[0][0] = ux; To[0][1] = vx; To[0][2] = wx; To[0][3] = qx;
      To[1][0] = uy; To[1][1] = vy; To[1][2] = wy; To[1][3] = qy;
      To[2][0] = uz; To[2][1] = vz; To[2][2] = wz; To[2][3] = qz;
      To[3][0] =  0; To[3][1] =  0; To[3][2] =  0; To[3][3] =  1;

      Ao = aw; Bo = bw; Co = cw;
    }

    if (tmode) {
      PoseMat(qx-To[0][3], qy-To[1][3], qz-To[2][3],
               aw-Ao, bw-Bo, cw-Co, Tt);
      MatMult(To, Tt, Tw);
      ux = Tw[0][0]; vx = Tw[0][1]; wx = Tw[0][2]; qx = Tw[0][3];
      uy = Tw[1][0]; vy = Tw[1][1]; wy = Tw[1][2]; qy = Tw[1][3];
      uz = Tw[2][0]; vz = Tw[2][1]; wz = Tw[2][2]; qz = Tw[2][3];
    }

    fwABC2(ux, uy, uz, vx, vy, vz, wz);

    pos->tran.x = qx;
    pos->tran.y = qy;
    pos->tran.z = qz;
    pos->a = aw/M_PI*180;
    pos->b = bw/M_PI*180;
    pos->c = cw/M_PI*180;

    return 0;
}

//=============================================================================
int kinematicsInverse(const EmcPose * pos,
      double *joint,
      const KINEMATICS_INVERSE_FLAGS * iflags,
      KINEMATICS_FORWARD_FLAGS * fflags)
{
    double a1 = *(haldata->DH_a1);
    double a2 = *(haldata->DH_a2);
    double a3 = *(haldata->DH_a3);
    double d1 = *(haldata->DH_d1);
    double d2 = *(haldata->DH_d2);
    double d4 = *(haldata->DH_d4);
    double d6 = *(haldata->DH_d6);
    int tmode = *(haldata->tmode);
```

```
    double Xt = pos->tran.x;
    double Yt = pos->tran.y;
    double Zt = pos->tran.z;
    double At = pos->a/180*M_PI;
    double Bt = pos->b/180*M_PI;
    double Ct = pos->c/180*M_PI;

    double th1, th2, th3, th4, th5, th6;
    double ux, uy, uz, vx, vy, vz, wx, wy, wz, px, py, pz;
    double r, k1, k2, qx, qy, qz;
    double c1, s1, c2, s2, c3, s3, c4, s4, c5, s5, c23, s23;
    double v114, v124, v214, v224, v113, v313, v323;
    double v111, v131, v311, v331, v411, v431;
    static double th4old = 0.0;
//
    int n1 = 1; // shoulder on the right
    int n2 = 1; // elbow up
    int n4 = 1; // wrist not flipped

    joint[6] = To[0][3];  joint[7] = To[1][3]; joint[8] = To[2][3];

    if (tmode) {
//              tool coordinates
       PoseMat(Xt-To[0][3], Yt-To[1][3], Zt-To[2][3],
               At-Ao, Bt-Bo, Ct-Co, Tt);
       MatMult(To, Tt, Tw);
    } else {
//              world coordinates
       PoseMat(Xt, Yt, Zt, At, Bt, Ct, Tw);
    }

      ux = Tw[0][0]; vx = Tw[0][1]; wx = Tw[0][2]; qx = Tw[0][3];
      uy = Tw[1][0]; vy = Tw[1][1]; wy = Tw[1][2]; qy = Tw[1][3];
      uz = Tw[2][0]; vz = Tw[2][1]; wz = Tw[2][2]; qz = Tw[2][3];

/*  wrist position -----------------------------------------------*/
    px = qx - d6*wx;
    py = qy - d6*wy;
    pz = qz - d6*wz;

/*  solve for th1 ------------------------------------------------*/
    r = sqrt(px*px + py*py);
    if (r < d2) {
/*      'ERROR:--------- point not reachable' */
        return 1;
    }
    k1 = atan2(py, px);
```

```
    k2 = asin(d2/r);
    if (n1 == 1) { th1 = k1 + k2;}
    else { th1 = k1 - k2 + M_PI;}
    c1 = cos(th1);  s1 = sin(th1);

/*  solve for th2 ------------------------------------------------*/
    v114 = px*c1 + py*s1 - a1;
    v124 = pz - d1;
    r = sqrt(v114*v114 + v124*v124);
    k1 = (a2*a2 - d4*d4 - a3*a3 + v114*v114 + v124*v124)/(2*a2*r);
    if (abs(k1) > 1) {
/*       'ERROR:--------- point not reachable'; */
         return 2;
    }
    k2 = acos(k1);
    if (elbup == 1) {n2 = 1;}
    else {n2 = -1;}
    th2 = atan2(v124, v114) + n2*k2;
    c2 = cos(th2); s2 = sin(th2);

/*  solve for  th3  ------------------------------------------------*/
    v214 =  c2*v114 + s2*v124 - a2;
    v224 = -s2*v114 + c2*v124;
    th3 = -atan2(a3, d4) + atan2(v214, -v224);
    c3 = cos(th3); s3 = sin(th3);

/*  solve for  th4  ------------------------------------------------*/
    c23 = cos(th2+th3); s23 = sin(th2+th3);
    v113 = c1*wx + s1*wy;
    v313 = c23*v113 + s23*wz;
    v323 = s1*wx - c1*wy;

    if ((fabs(v323) < eps) && (fabs(v313) < eps)){ th4 = 0;}
    else {th4 = atan2(n4*v323, n4*v313);}
//         take care of singularities and map for continuity
    if ((fabs(v323) < eps) && (v313 < eps)) {th4 = th4old;}
    if ((v323 > eps) && (v313 < eps)) {th4 = th4 - 2*M_PI;}
    if ((fabs(v113) < eps) && (fabs(v313) < eps) &&
        (fabs(v323) < eps) ) {th4 = th4old;}
    th4old = th4;
//
    c4 = cos(th4); s4 = sin(th4);

/*  solve for  th5  ------------------------------------------------*/
    k1 = c4*v313 + s4*v323;
    k2 = s23*v113 - c23*wz;
    th5 = atan2(k1, k2);
```

```
//
    c5 = cos(th5); s5 = sin(th5);

/*  solve for th6  ------------------------------------------------*/
    v111 = c1*ux + s1*uy;
    v131 = s1*ux - c1*uy;
    v311 = c23*v111 + s23*uz;
    v331 = s23*v111 - c23*uz;
    v411 =  c4*v311 + s4*v131;
    v431 = -s4*v311 + c4*v131;
//
    k1 = v431;
    k2 = c5*v411 - s5*v331;
    th6 = atan2(k1, k2);
//
/* convert to degrees -------------------------------------------*/
   joint[0] = th1*180/PM_PI;
   joint[1] = th2*180/PM_PI;
   joint[2] = th3*180/PM_PI;
   joint[3] = th4*180/PM_PI;
   joint[4] = th5*180/PM_PI;
   joint[5] = th6*180/PM_PI;

   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("robot6kins");
    if(comp_id < 0) return comp_id;
```

```
    haldata = hal_malloc(sizeof(struct haldata));

    if((res = hal_pin_float_new("robot6kins.DH-a1", HAL_IO,
             &(haldata->DH_a1), comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("robot6kins.DH-a2", HAL_IO,
             &(haldata->DH_a2), comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("robot6kins.DH-a3", HAL_IN,
             &(haldata->DH_a3), comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("robot6kins.DH-d1", HAL_IO,
             &(haldata->DH_d1), comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("robot6kins.DH-d2", HAL_IO,
             &(haldata->DH_d2), comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("robot6kins.DH-d4", HAL_IN,
             &(haldata->DH_d4), comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("robot6kins.DH-d6", HAL_IN,
             &(haldata->DH_d6), comp_id)) < 0) goto error;
    if((res = hal_pin_bit_new("robot6kins.soffs", HAL_IN,
             &(haldata->soffs), comp_id)) < 0) goto error;
    if((res = hal_pin_bit_new("robot6kins.tmode", HAL_IN,
             &(haldata->tmode), 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 two extra HAL pins have been added: robot3kins.tmode and robot3kins.soffs. The ”tmode” pin should be connected to a VCP button (toggle) that can be used to change between ”tool” and ”world” coordinates. The ”soffs” pin is connected to a VCP button to trigger an ”offset” of coordinates.

To work in ”tool” coordinates with this component the following steps is normally required (using the AXIS gui with an attached VCP):

1. Move in ”world” mode to a offset pose $X_o, Y_o, Z_o, A_o, B_o, C_o$.
2. Do a ”touch off” for all 6 axes X, Y, Z, A, B, C that are non-zero.
3. Activate the ”set offset” button.
4. Toggle the ”tool” mode button to change into tool mode.
5. Perform operations (jogging or Gcode) in tool coordinates until finished.
6. Move to zero tool pose.

7. Toggle ”tool” mode button to go back to ”world” mode.

Warning: the kinematics script given above is still experimental. Although much has been done to avoid this, sudden moves of the wrist joint (joint 3) by 180 degrees may still occur in certain (rare) situations.

1.11 VISMACH simulation model

VISMACH is a tool to show a 3D simulation of a physical machine in operation.

Vismach.py is a python library to draw objects in a simulation window. It is located in /usr/lib/pymodules/python2.6. The simulation program itself is a script based on vismach.py. The scripts are located in /usr/bin. The .hal file of your machine in your /home/user/linuxcnc/config/yourmachine loads this script with ”loadusr” as a HAL component and connects the joints. The following items are described in the script:

- geometry is defined or loaded from a .stl or .obj file
- placement of geometry (translation and or rotation)
- joints are defined and connected to HAL PINs
- colour is defined

Some of the geometric solids that can be directly defined are:

part = CylinderX(x1,r1,x2,r2) : a cylinder along X-axis from x1 to x2 with radiuses r1 and r2.

part = CylinderY(y1,r1,y2,r2) : a cylinder along Y-axis from y1 to y2 with radiuses r1 and r2.

part = CylinderZ(z1,r1,z2,r2) : a cylinder along Z-axis from z1 to z2 with radiuses r1 and r2.

part = Box(x1,y1,z1, x2,y2,z2) : a rectangular box between opposite corners x1,y1,z1 to x2,y2,z2.

Parts can be imported as STL files:

part = AsciiSTL(”filename”) : the part modelled as an STL file is loaded; example: spindle = AsciiSTL(”spindle.stl”).

Some transformations can also be done:

part = Translate(parts,x,y,z) : translate parts to x,y,z.

part = Rotate(parts,th,x,y,z) : Rotate parts th degrees around axis defined by x,y,z.

part = HalTranslate(parts,comp,var,x,y,z) : Translate parts along x,y,z vector. Distance is defined by a Hal pin ”var” of component ”comp”.

part = HalRotate(parts,comp,var,th,x,y,z) : Rotate parts by th degrees around axis defined by x,y,z. Th is scaled by a Hal pin "var" of component "comp".

Parts can be grouped together to form subassemblies:

part = Collection(parts) : the listed parts are collected to move together.

The tool position is "captured" and the parts or collections can be given colours as follows:

part = Capture() : the position (tool or work) is captured.

part = Color(red,green,blue,alpha,parts) : RGB color code is used plus transparency or blending - all values between 0 and 1.

The last entry in the script is the "main" entry:

main(model, tool, work, size) : define parts and window size.

If "parts" is a list then it must be included in square brackets, ie. [table,handle,screw].

For our 6-joint serial robot the following is an example of a VISMACH script. The link lengths a1=200, a2=600, a3=110, d1=450, d2=0, d4=620, and d6=150 mm. The actual robot modelled is the Yaskawa MOTOMAN YFRL NNA 10GB.

```
#!/usr/bin/python
#-----------------------------------------------------------------
#  This program generates a VISMACH model of a 6-joint serial robot.
# Two extra "hal" pins are used "lnkdx, lnkdz", which are linked to a
# component "croffsets.comp". They are the offsets of the bar link at
# the back of the robot. It should be possible to include this
# component into the VISMACH script - so far not achieved!
#
# Author: Rudy du Preez (SA-CNC-CLUB)
# License: GPL Version 2
#-----------------------------------------------------------------
from vismach import *
import hal

c = hal.component("motomangui")
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("joint6", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("lnkdx",  hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("lnkdz",  hal.HAL_FLOAT, hal.HAL_IN)
```

```
c.newpin("grip",   hal.HAL_FLOAT, hal.HAL_IN)
c.ready()

#-----------------------------------------------------------

# finger
finger1 = CylinderZ(-70, 10, 0, 0.2)
#finger2 = CylinderZ(20, 10, 90, 7)
finger1 = HalRotate([finger1],c,"grip", 40,0,1,0)
#finger2 = HalRotate([finger2],c,"grip",-40,0,1,0)
#finger1 = Translate([finger1], 20,0.0,0.1)
#finger2 = Translate([finger2],-20,0.0,0.1)
# axes
xaxis = Color([1,0,0,1],[CylinderX(0,3,70,3)])
yaxis = Color([0,1,0,1],[CylinderY(0,3,70,3)])
zaxis = Color([0,0,1,1],[CylinderZ(0,3,70,3)])
finger1 = Collection([finger1,xaxis,yaxis,zaxis])
tooltip = Capture()
link7 = AsciiSTL(filename="link7.stl")
link7 = Collection([tooltip])

link7 = Color([0.9,0.1,0.0,1],[link7])
# assemble fingers, and make it rotate
link7 = HalRotate([finger1,link7],c,"joint6",1,0,0,1)

# link 6
link6 = AsciiSTL(filename="link6.stl")
# rotate and translate it so that the joint 6 is in origin
link6 = Color([0.5,1,0.5,1],[link6])
link6 = Rotate([link6],90,0,1,0)
link6 = Translate([link6],0,0,-150)
# mount link7 on it
link6 = Collection([link7, link6])
#translate it back so joint 5 rotation in origin
link6 = Translate([link6],0,0,150)
xaxis5 = Color([1,0,0,1],[CylinderX(0,5,100,5)])
yaxis5 = Color([0,0,1,1],[CylinderY(0,5,120,5)])
link6 = Collection([link6, xaxis5, yaxis5])
#apply HAL DOF
link6 = HalRotate([link6],c,"joint5",1,0,1,0) # wrist

# link 5, wrist
link5 = AsciiSTL(filename="link5.stl")
link5 = Color([0.5,0.5,0.5,1],[link5])
#translate it so joint 5 rotation in origin
link5 = Translate([link5],0,0,-400)
xaxis4 = Color([1,0,0,1],[CylinderX(0,3,120,3)])
```

```
yaxis4 = Color([0,1,0,1],[CylinderY(0,3,150,3)])
marker = CylinderZ(-60,15,60,15)
marker = Translate([marker],56,0,-150)
# assemble
link5 = Collection([link6, link5, xaxis4, yaxis4, marker])
# translate back to joint4 in origin
link5 = Translate([link5],0,0,490)
#apply HAL DOF
link5 = HalRotate([link5],c,"joint4",1,0,0,1) # arm twist

# link4, arm, origin is in the joint3 location
link4 = AsciiSTL(filename="link4.stl")
link4 = Color([0,0.5,1,1],[link4])
link4 = Translate([link4],0,0,90)
link4 = Collection([link5, link4])
# move back to joint3 in origin
link4 = Translate([link4],110,0,130)
link4 = Rotate([link4],-90,0,1,0)
link4 = HalRotate([link4],c,"joint3",1,0,1,0) # elbow

#crank
crank = AsciiSTL(filename="crank.stl")
crank = Color([0,0.5,1,1],[crank])
crank = HalRotate([crank],c,"joint3",1,0,1,0)
crank = Translate([crank],0,25,-600)

#link
link = AsciiSTL(filename="link4a.stl")
link = Rotate([link],-90,0,1,0)
link = Translate([link],200,0,-600)
link = HalTranslate([link],c,"lnkdx",-1,0,0)
link = HalTranslate([link],c,"lnkdz",0,0,-1)
crank = Collection([crank, link])

# link 3, upper arm
link3 = AsciiSTL(filename="link3.stl")
link3 = Color([0,0,1,1],[link3])
link3 = Translate([link3],0,15,-600)
#assemble
link3 = Collection([link4, link3, crank])
link3 = Rotate([link3],-90,0,1,0)
#move back to j2 in Origin
link3 = Translate([link3],-600,0,0)
#and rotate according to kinematics
link3 = HalRotate([link3],c,"joint2",1,0,1,0) # shoulder

# link 2 vertical axis
```

```
link2 = AsciiSTL(filename="link2.stl")
link2 = Color([1,0.19,1,1],[link2])
link2 = Rotate([link2], 180,0,0,1)
link2 = Translate([link2], 200,0,-450)
link2 = Collection([link3, link2])
link2 = Translate([link2],-200,0,450)
#
link2 = HalRotate([link2],c,"joint1",1,0,0,1) # vert axis

#move link2 up
link2 = Translate([link2], 0,0,140)

link1 = AsciiSTL(filename="link1.stl");
link1 = Color([1,0.19,0.2,1],[link1])
link1 = Rotate([link1], 180,0,0,1)

xaxis0 = Color([1,0,0,1],[CylinderX(0,5,-300,5)])
yaxis0 = Color([0,1,0,1],[CylinderY(0,5,-300,5)])
# add a floor
floor = Box(-1.5,-1.5,-0.02,1.5,1.5,0.0)
floor = Collection([floor, xaxis0, yaxis0])
work = Capture()

model = Collection([link1, link2, floor, work])

main(model, tooltip, work, 1000)
```

The component ”croffsets” is included below:

```
component croffsets "motoman link motion";
pin in float angle;
pin in float crlen;
pin out float ldx;
pin out float ldz;
function _;
license "GPL";
;;
#include <rtapi_math.h>

FUNCTION(_) {
    ldx = crlen*(1 - cos(angle/180*3.14159));
    ldz = crlen*sin(angle/180*3.14159);
}
```

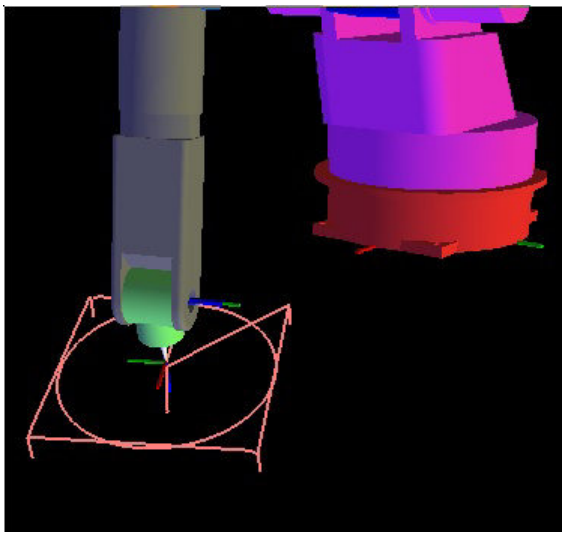

Figure 5: Gcode example 1

1.12 GCODE examples

Two basic examples of Gcode are listed below. They could be loaded and executed after the robot arm has been position at the origin of the tool coordinate system somewhere on the work surface, which can have can have orientation angles Ao, Bo and Co relative to the ”world” axes X,Y,Z.

The first one does some ”drilling” in spce at the four corners of a square and cuts a cisicle in the ”tool” x-y plane.

```
( robot3 test 1- "drilling and cutting")
G21 G90 G64 G40 G17
G0 Z10
M3 S1000
G0 X-200 Y-200
G1 F1000.0 Z60
G0 Z10
G0 X200 Y-200
G1 Z60
G0 Z10
G0 X200 Y200
G1 Z60
G0 Z10
G0 X-200 Y200
G1 Z60
G0 Z10
G0 X-200 Y0
```

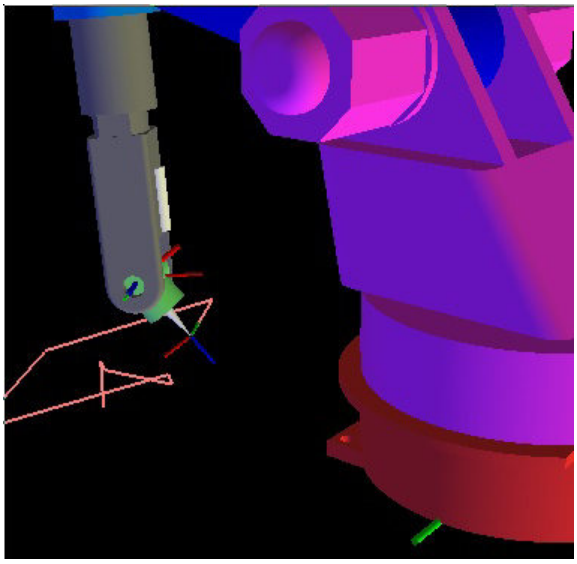

Figure 6: Gcode example 2

```
G1 Z20 F100
G3 I200 J0 F2000
G0 Z0
G0 X0 Y0
M5
M2
```

The second simulates a ”welding” process along the square with the welding torch at an angle to the ”tool” x-y plane.

```
( robot3 test 2 - "welding")
G21 G90 G64 G40 G17
G0 Z20
G0 X-200 Y-200
G0 A20 Z0
G1 X200 Y-200 F1000
G0 A0 B20
G1 X200 Y200
G0 A-20 B0
G1 X-200 Y200
G0 A0 B-20
G1 X-200 Y-200
G0 A0 B0 Z20
G0 X0 Y0
G0 Z0
M5
M2
```

2 REFERENCES

[1] Computationally Efficient Kinematics for Manipulators with Spherical Wrists Based on the Homogeneous Transformation Representation: RP Paul and H Zhang, The Int. Jour. of Robotics Research, V5 N2 1986.

[2] Robot Analysis: Lung-wen Tsai, John Wiley, 1999

[3] Robot Manipulators: RP Paul, MIT Press 1979

[4] Theory of Applied Robotics, RN Jazar, Springer 2006

3 FIGURES

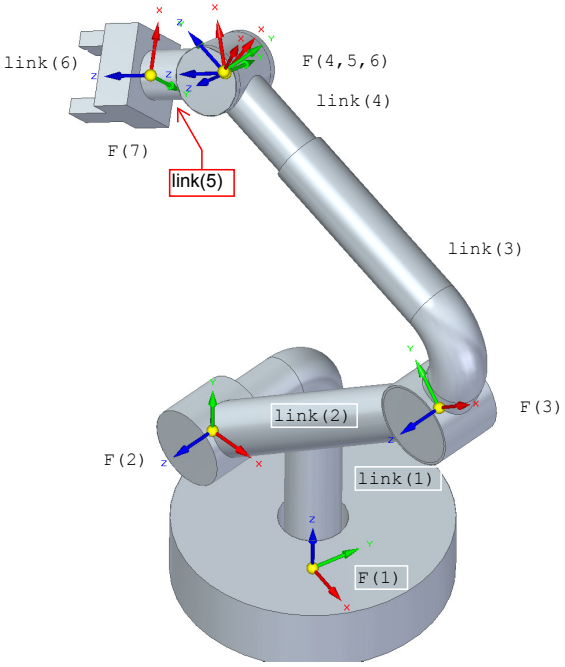

Figure 7: Links and joint frames F_1 to F_7

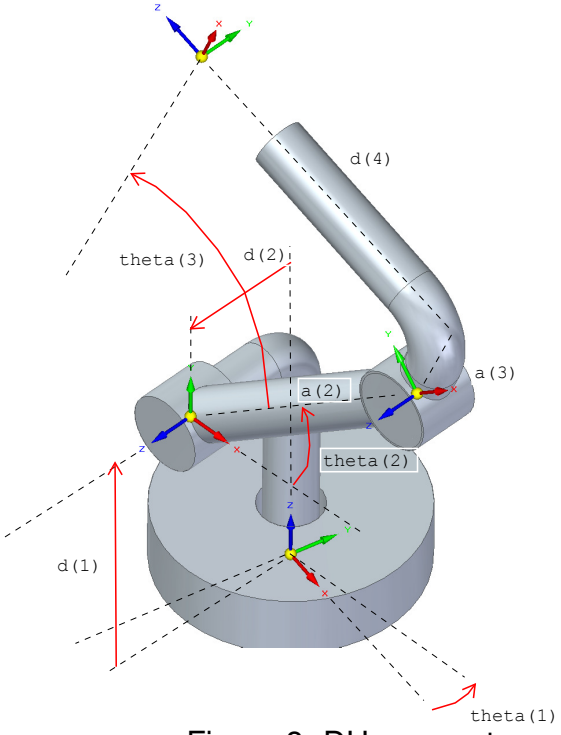

Figure 8: DH parameters

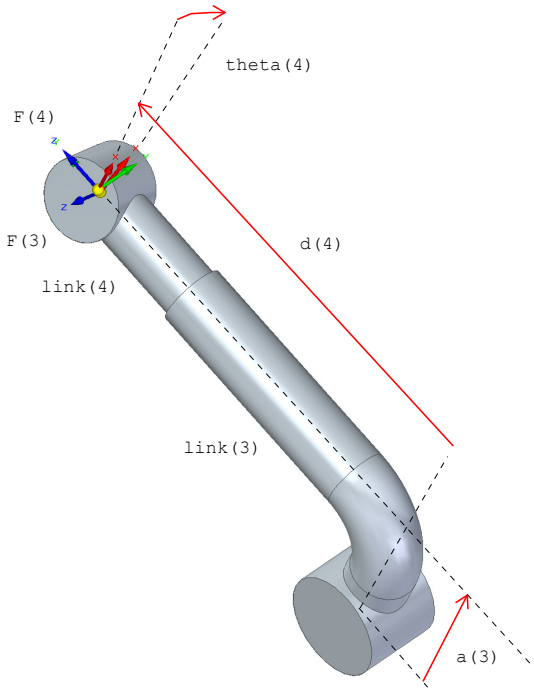

Figure 9: DH parameters

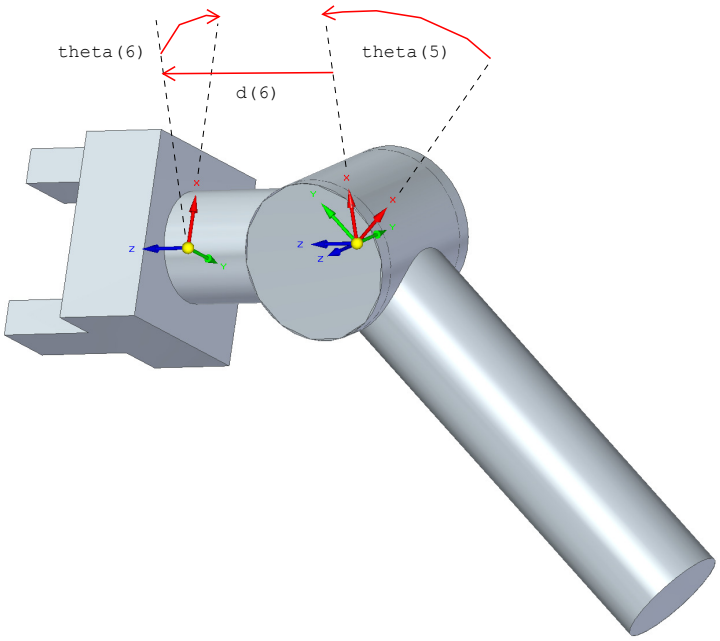

Figure 10: DH parameters