PLANER 3-DOF SERIAL ROBOT - Kinematics Implementation in LinuxCNC

Prof. Rudy du Preez
SA-CNC-CLUB

January 29, 2014

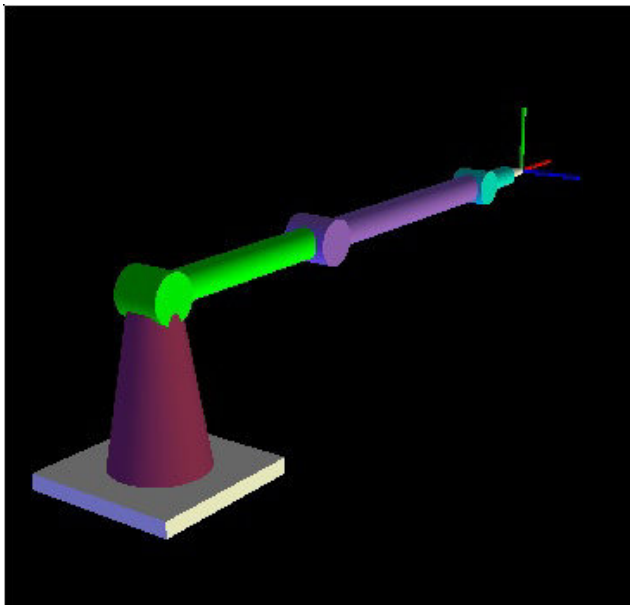

Figure 1: Vismach model of Robot3

1 PLANER 3-DOF ROBOT ARM

In this section we deal with the simplest possible serial robot arm in a 2-dimensional plane with three joints or degrees-of-freedom. With such an arm you can measure positions using *forward kinematics* or you can machine using *forward and inverse kinematics*, all in one plane.

We develop the kinematics transformations for a 3-dof robot and include some examples. We then show how this can be implemented in LinuxCNC.

1.1 Configuration

The configuration of a 3-dof planer robot is shown in Fig. 2. The lengths of the 3 links are given by a_1, a_2 and a_3. The orientation in the x-y plane is defined by the three angles th1,th2 and th3 (or θ_1, θ_2 and θ_3)- these are also called *joint angles*.

The position of each joint j_i is defined by the coordinates x_i, y_i. The end-effector's position and orientation is defined by x_E, y_E and the angle phi (or ϕ).

1.2 Forward Kinematics Transformation

For link1 we may write, with x_B, y_B the location of the robot's fixed base:

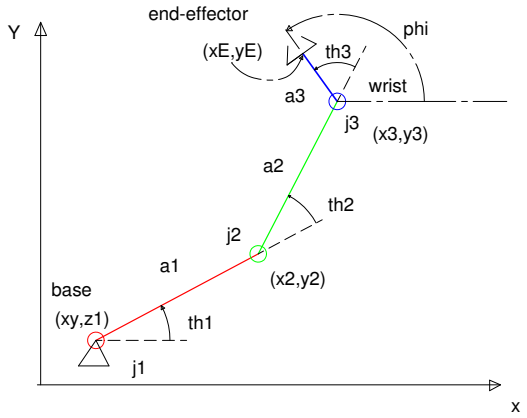

Figure 2: Planer 3-dof serial arm

$$x_1 = x_B, \qquad y_1 = y_B \tag{1}$$

For link2:

$$x_2 = x_1 + a_1 \cdot cos(\theta_1), \qquad y_2 = y_1 + a_1 \cdot sin(\theta_1), \tag{2}$$

For link2:

$$x_3 = x_2 + a_2 \cdot cos(\theta_1 + \theta_2), \qquad y_3 = y_2 + a_2 \cdot sin(\theta_1 + \theta_2), \tag{3}$$

After substitution of (1) and (2) into (3) we get:

$$x_3 = x_B + a_1 \cdot cos(\theta_1) + a_2 \cdot cos(\theta_1 + \theta_2), \qquad y_3 = y_B + a_1 \cdot sin(\theta_1) + a_2 \cdot sin(\theta_1 + \theta_2) \tag{4}$$

so the wrist coordinates are defined.

We can go further and determine the end-effector position and orientation:

$$x_E = x_3 + a_3 \cdot cos(\phi), \qquad y_E = y_3 + a_3 \cdot sin(\phi), \quad \text{with} \qquad \phi = \theta_1 + \theta_2 + \theta_3 \tag{5}$$

Using shorthand notations $c_i = cos(\theta_i)$, $s_i = sin(\theta_i)$, and $c_{ij} = cos(\theta_i + \theta_j)$, etc, we can write the combined result:

$$\begin{aligned} x_E &= x_B + a_1 \cdot c_1 + a_2 \cdot c_{12} + a_3 \cdot c_{123} \\ y_E &= y_B + a_1 \cdot s_1 + a_2 \cdot s_{12} + a_3 \cdot s_{123} \end{aligned} \tag{6}$$

Example 1

Given the joint angles (0, 90, -90), base position (0, 0), and link lengths (200,200,100), we can calculate from (4) and (5)):

$$\begin{aligned} x_3 &= 0.0 + 200 \cdot cos(0) + 200 \cdot cos(0 + 90) = 200 \\ y_3 &= 0.0 + 200 \cdot sin(0) + 200 \cdot sin(0 + 90) = 200 \end{aligned}$$

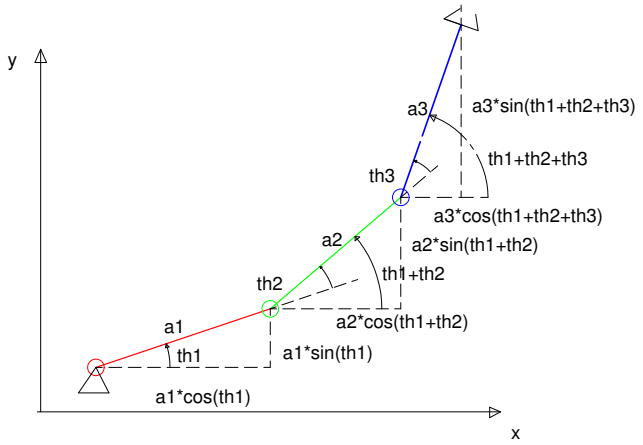

Figure 3: x- and y-components of links

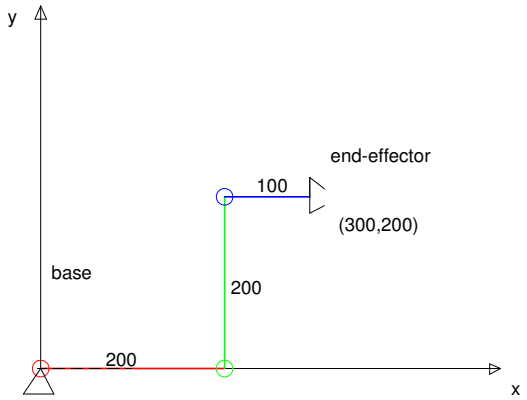

Figure 4: Example 1: Planer 3-dof serial arm

and

$$x_E = 200 + 100 \cdot cos(0) = 300$$
$$y_E = 200 + 100 \cdot sin(0) = 200$$

or from (6):

$$x_E = 0.0 + 200 \cdot cos(0) + 200 \cdot cos(0 + 90) + 100 \cdot cos(0 + 90 - 90) = 300$$
$$y_E = 0.0 + 200 \cdot sin(0) + 200 \cdot sin(0 + 90) + 100 \cdot sin(0 + 90 - 90) = 200$$
$$\phi = 0 + 90 - 90 = 0$$

1.3 Matrix formulation of Forward Kinematics

We can write the forward transformation very compact using matrices. We define a general forward transformation matrix for link i as follows:

$$ {}^{i-1}T_i(\theta_i, a_i) = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & a_i \cdot cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i) & a_i \cdot sin(\theta_i) \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_i & -s_i & a_i c_i \\ s_i & c_i & a_i s_i \\ 0 & 0 & 1 \end{bmatrix} \tag{7} $$

then the combined transformation from the robot base to the wrist is two matrices multiplied:

$$ {}^1T_2 \cdot {}^2T_3 = {}^1T_3 \tag{8} $$

or

$$ {}^1T_3 = \begin{bmatrix} c_1 & -s_1 & a_1 c_1 \\ s_1 & c_1 & a_1 s_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_2 & -s_2 & a_2 c_2 \\ s_2 & c_2 & a_2 s_2 \\ 0 & 0 & 1 \end{bmatrix} = $$

$$ \begin{bmatrix} c_1c_2 - s_1s_2 & -c_1s_2 - s_1c_2 & a_1c_1 + a_2c_1c_2 - a_2s_1s_2 \\ s_1c_2 + c_1s_2 & -s_1s_2 + c_1c_2 & a_1s_1 + a_2s_1c_2 + a_2c_1s_2 \\ 0 & 0 & 1 \end{bmatrix} $$

which can be simplified to:

$$ {}^1T_3 = \begin{bmatrix} c_{12} & -s_{12} & a_1c_1 + a_2c_{12} \\ s_{12} & c_{12} & a_1s_1 + a_2s_{12} \\ 0 & 0 & 1 \end{bmatrix} \tag{9} $$

1.4 End-effector Pose

To get the end-effector position and orientation (pose) we can do:

$$ {}^1T_3 \cdot {}^3T_4 = {}^1T_4 \tag{10} $$

which will give

$$ {}^1T_4 = \begin{bmatrix} c_{123} & -s_{123} & a_1c_1 + a_2c_{12} + a_3c_{123} \\ s_{123} & c_{123} & a_1s_1 + a_2s_{12} + a_3s_{123} \\ 0 & 0 & 1 \end{bmatrix} \tag{11} $$

where $s_{12} = sin(\theta_1 + \theta_2)$ and $s_{123} = sin(\theta_1 + \theta_2 + \theta_3)$, etc.

Equation (11) gives the end-effector position and orientation relative to the base frame. We can partition it into sub-matrices as follows:

$$ \left[\begin{array}{cc|c} c_{123} & -s_{123} & a_1c_1 + a_2c_{12} + a_3c_{123} \\ s_{123} & c_{123} & a_1s_1 + a_2s_{12} + a_3s_{123} \\ \hline 0 & 0 & 1 \end{array}\right] = \begin{bmatrix} R & q \\ 0 & 1 \end{bmatrix} \tag{12} $$

then R is a rotation matrix that defines the orientation of the end-effector and q is a position vector that defines its position:

$$ R = \begin{bmatrix} c_{123} & -s_{123} \\ s_{123} & c_{123} \end{bmatrix}, \quad q = \begin{bmatrix} q_x \\ q_y \end{bmatrix} = \begin{bmatrix} a_1c_1 + a_2c_{12} + a_3c_{123} \\ a_1s_1 + a_2s_{12} + a_3s_{123} \end{bmatrix} \tag{13} $$

1.5 General Form of Forward Kinematics

The general form of the forward kinematics transformation can be written as:

$$ {}^1T_n = {}^1T_2 \cdot {}^2T_3 \cdots {}^{n-1}T_n = \begin{bmatrix} {}^1R_n & {}^1q_n \\ 0 & 1 \end{bmatrix} \tag{14} $$

where ${}^1R_n$ is a rotation transformation matrix

$$ {}^1R_n = \begin{bmatrix} cos(\sum_{i=1}^n \theta_i) & -sin(\sum_{i=1}^n \theta_i) \\ sin(\sum_{i=1}^n \theta_i) & cos(\sum_{i=1}^n \theta_i) \end{bmatrix} = \begin{bmatrix} u_x & w_x \\ u_y & w_y \end{bmatrix} = \begin{bmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix} \tag{15} $$

with u_x, u_y the direction cosines of the end-effector x_E-axis and w_x, w_y the direction cosines of the y_E-axis. Also we have ${}^1q_n$, a position vector:

$$ {}^1q_n = \begin{bmatrix} \sum_{i=1}^n (a_i \cdot cos(\sum_{j=1}^i \theta_j) \\ \sum_{i=1}^n (a_i \cdot sin(\sum_{j=1}^i \theta_j) \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \end{bmatrix} \tag{16} $$

where q_x, q_y are the cartesian coordinates of the end-effector. ϕ is the orientation of the end-effector with respect to the x-axis (see Fig. 2).

Example 2

Given the joint angles (30, 30, 20), base position (0, 0), and link lengths (200,200,100), we can calculate from (6):

$$ x_3 = 0.0 + 200 \cdot cos(30) + 200 \cdot cos(30 + 30) = 200 \cdot 0.866 + 200 \cdot 0.5 = 273.21 $$
$$ y_3 = 0.0 + 200 \cdot sin(30) + 200 \cdot sin(30 + 30) = 200 \cdot 0.5 + 200 \cdot 0.866 = 273.21 $$

and from (5)

$$ x_E = 273.2 + 100 \cdot cos(80) = 273.2 + 100 \cdot 0.174 = 290.57 $$
$$ y_E = 273.2 + 100 \cdot sin(80) = 273.2 + 100 \cdot 0.985 = 371.69 $$
$$ \phi = 30 + 30 + 20 = 80 $$

However, using (13) we can also calculate directly:

$$ \begin{bmatrix} q_x \\ q_y \end{bmatrix} = \begin{bmatrix} 200 \cdot cos(30) + 200 \cdot cos(60) + 100 \cdot cos(80) \\ 200 \cdot sin(30) + 200 \cdot sin(60) + 100 \cdot sin(80) \end{bmatrix} = \begin{bmatrix} 290.57 \\ 371.69 \end{bmatrix} $$

1.6 Inverse Kinematics Transformation

We often want to calculate the joint angles for a given end-effector pose. Then we need to perform an inverse kinematics transformation. We could do it numerically using inverse matrices, however in CNC trajectory control we want the operations as fast as possible, therefore it is better to solve for the joint angles trigonometrically.

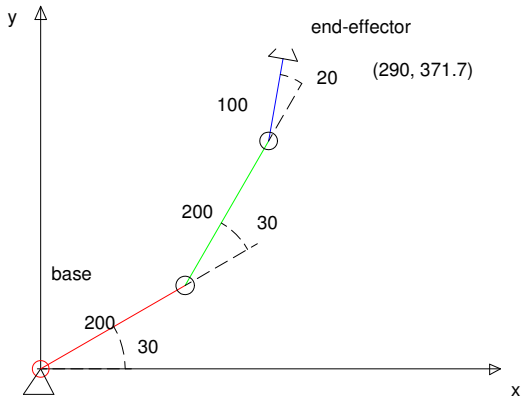

Figure 5: Example 2: Planer 3-dof serial arm

Given the pose as q_x, q_y, ϕ, we can write the overall forward transformation for our 3-dof serial robot in two different forms as follows, using (14-16):

$$ {}^1T_4 = \begin{bmatrix} cos(\phi) & -sin(\phi) & q_x \\ sin(\phi) & cos(\phi) & q_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{123} & -s_{123} & a_1 \cdot c_1 + a_2 \cdot c_{12} + a_3 \cdot c_{123} \\ s_{123} & c_{123} & a_1 \cdot s_1 + a_2 \cdot s_{12} + a_3 \cdot s_{123} \\ 0 & 0 & 1 \end{bmatrix} \tag{17} $$

We now find the coordinates of the wrist $p_x = x_3$, $p_y = y_3$ as follows, since $\phi = \theta_1 + \theta_2 + \theta_3$:

$$ p_x = q_x - a_3 \cdot cos(\phi) = q_x - a_3 \cdot c_{123} \tag{18} $$
$$ p_y = q_y - a_3 \cdot sin(\phi) = q_y - a_3 \cdot s_{123} \tag{19} $$

With the wrist coordinates our above equation becomes:

$$ {}^1T_4 = \begin{bmatrix} cos(\phi) & -sin(\phi) & p_x \\ sin(\phi) & cos(\phi) & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{123} & -s_{123} & a_1 \cdot c_1 + a_2 \cdot c_{12} \\ s_{123} & c_{123} & a_1 \cdot s_1 + a_2 \cdot s_{12} \\ 0 & 0 & 1 \end{bmatrix} $$

Equating the (1,3) and (2,3) elements of the the two matrices, we find:

$$ p_x = a_1 \cdot c_1 + a_2 \cdot c_{12} \tag{20} $$
$$ p_y = a_1 \cdot s_1 + a_2 \cdot s_{12} \tag{21} $$

or re-arranging

$$ p_x - a_1 \cdot c_1 = a_2 \cdot c_{12} \tag{22} $$
$$ p_y - a_1 \cdot s_1 = a_2 \cdot s_{12} \tag{23} $$

If we now take the square of each side and add them together, we get, since $cos()^2 + sin()^2 = 1$:

$$ p_x^2 - 2p_x a_1 c_1 + a_1^2 c_1^2 + p_y^2 - 2a_1 s_1 p_y + a_1^2 s_1^2 = a_2^2 \tag{24} $$

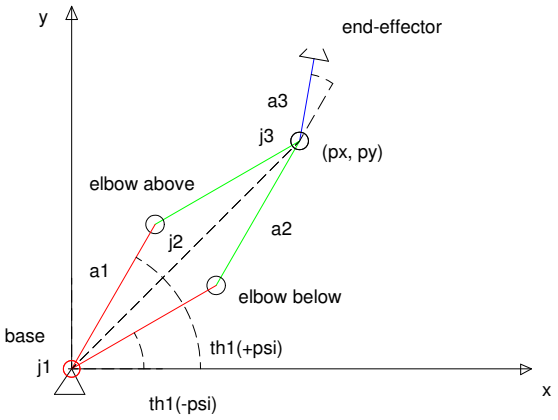

Figure 6: Two possible solutions for θ_1

or

$$c_1 p_x + s_1 p_y = \frac{p_x^2 + p_y^2 + a_1^2 - a_2^2}{2a_1} \tag{25}$$

To solve this equation for θ_1, we define

$$p_x = rcos(\beta), \quad p_y = rsin(\beta), \quad r = \sqrt{(p_x^2 + p_y^2)}, \quad r > 0 \tag{26}$$

then

$$tan(\beta) = \frac{p_y}{p_x}, \quad \text{and} \quad \beta = tan^{-1}\frac{p_y}{p_x} \tag{27}$$

Substituting (26) into (25)we get:

$$c_1 cos(\beta) + s_1 sin(\beta) = \frac{p_x^2 + p_y^2 + a_1^2 - a_2^2}{2a_1 r} \tag{28}$$

Let

$$\psi = cos^{-1}(\frac{p_x^2 + p_y^2 + a_1^2 - a_2^2}{2a_1 r}) \tag{29}$$

then

$$c_1 cos(\beta) + s_1 sin(\beta) = cos(\theta_1 - \beta) = cos(\psi) \tag{30}$$

and $\theta_1 = \beta + \psi$ or, since the cos^{-1} function is double valued, $\theta_1 = \beta - \psi$, ie.

$$\theta_1 = tan^{-1}\frac{p_y}{p_x} \pm \psi \tag{31}$$

The two possible solutions can be identified as:

- $+\psi$: joint 2 above the line from joint 1 to joint 3 (elbow above)
- $-\psi$: joint 2 below the line from joint 1 to joint 3 (elbow below)

These two configurations are shown in Fig. 5

Corresponding to each solution for θ_1 we can solve for θ_2 by expanding (22) and (23) as follows:

$$p_x - a_1c_1 = a_2(c_1c_2 - s_1s_2) = (a_2c_1)c_2 - (a_2s_1)s_2 \tag{32}$$
$$p_y - a_1s_1 = a_2(s_1c_2 + c_1s_2) = (a_2c_1)s_2 + (a_2s_1)c_2 \tag{33}$$

Multiplying the first equation by s_2 and the second one by c_2 we find

$$c_2(p_x - a_1c_1) = (a_2c_1)c_2^2 - (a_2s_1)s_2c_2 \tag{34}$$
$$s_2(p_y - a_1s_1) = (a_2c_1)s_2^2 + (a_2s_1)c_2s_2 \tag{35}$$

or after adding together

$$c_2(p_x - a_1c_1) + s_2(p_y - a_1s_1) = a_2c_1 \tag{36}$$

We now substitute

$$p_x - a_1c_1 = rcos(\beta), \quad p_y - a_1s_1 = rsin(\beta) \tag{37}$$

$$r = \sqrt{(p_x - a_1c_1)^2 + (p_y - a_1s_1)^2} = \sqrt{p_x^2 + p_y^2 - 2a_1(p_xc_1 + p_ys_1) + a_1^2} = a_2 \tag{38}$$

then

$$c_2cos(\beta) + s_2sin(\beta) = cos(\beta - \theta_2) = c_1 = cos(\theta_1) \tag{39}$$

or

$$\theta_2 = \beta - \theta_1 \tag{40}$$

with, from (37)

$$\beta = tan^{-1}\frac{p_y - a_1s_1}{p_x - a_1c_1} \tag{41}$$

having solved for θ_1 and θ_2 the remaining angle θ_3 can be found from

$$\theta_3 = \phi - \theta_1 - \theta_2 \tag{42}$$

Example 3

Using the data from Example 2, we have as given $q_x = 290.57$, $q_y = 371.69$, $\phi = 80$ and link lengths $a_i = (200, 200, 100)$. Using (18,19) we get

$$\begin{aligned} p_x = &\quad 290.57 - 100 \cdot cos(80) &= 290.57 - 17.4 = 273.21 \\ p_y = &\quad 371.69 - 100 \cdot sin(80) &= 371.69 - 89.5 = 273.21 \end{aligned}$$

and with these values we get from (26,27)

$$r = \sqrt{273.21^2 + 273.21^2} = 386.37, \quad \beta = tan^{-1}\frac{273.21}{273.21} = 45.0$$

and from (26)

$$\psi = cos^{-1}(\frac{273.21^2 + 273.21^2 + 200^2 - 200^2}{2 \cdot 200 \cdot 386.37}) = 15.0$$

hence

$$\theta_1 = \beta \pm \psi = 45 + 15 = 60^0 \quad \text{(elbow above)} \quad \text{or} \quad 45 - 15 = 30^0 \quad \text{(elbow below)}$$

Solving for θ_2 we use (41,42)

$$\beta = tan^{-1}\frac{273.21 - 200 \cdot sin(30)}{273.21 - 200 \cdot cos(30)} = 60^0$$

and finally $\theta_2 = \beta - \theta_1 = 60 - 30 = 30^o$ and $\theta_3 = 80 - 30 - 30 = 20^o$.

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. 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). 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 (restricted to the X-Y plane):

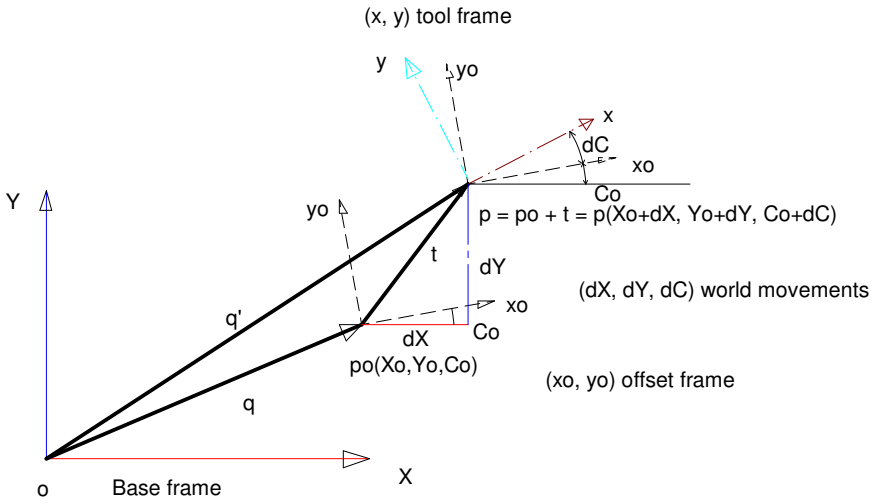

Figure 7: World coordinate move in the X-Y plane

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 20 degrees, 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 Y. 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. Our movements are then in terms of ”relative” or ”offset” coordinates

The world coordinate tool movement is depicted in Fig. 7. The ”offset” pose is indicated as $p_o(X_o, Y_o, C_o)$ and the new pose after the move $p(X_p, Y_p, C_p)$ is then:

$$\begin{aligned} X_p &= X_o + \Delta X \\ Y_p &= Y_o + \Delta Y \\ C_p &= C_o + \Delta C \end{aligned} \tag{43}$$

The angle of rotation in the X-Y plane is denoted by C (around the local Z-axis).

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

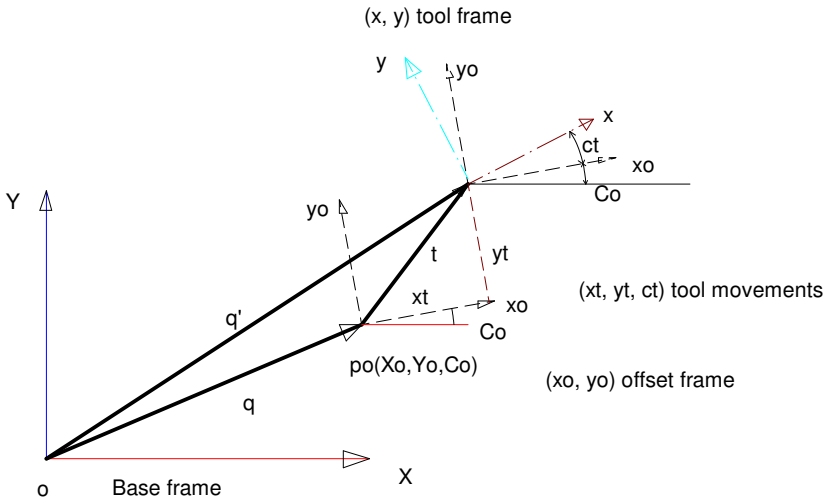

Figure 8: Tool coordinate move in the X-Y plane

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, or 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 in the X-Y plane 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, C_o or as a matrix:

$$ {}^{1}T_o = \begin{bmatrix} cos(C_o) & -sin(C_o) & X_o \\ sin(C_o) & cos(C_o) & Y_o \\ 0 & 0 & 1 \end{bmatrix} \tag{44} $$

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

$$ {}^{o}T_t = \begin{bmatrix} cos(c_t) & -sin(c_t) & x_t \\ sin(c_t) & cos(c_t) & y_t \\ 0 & 0 & 1 \end{bmatrix} \tag{45} $$

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:

$$ {}^{1}T_t = {}^{1}T_o \cdot {}^{o}T_t \tag{46} $$

or multiplied out:

$$ {}^1T_t = \begin{bmatrix} c_C c_c - s_C s_c & -c_C s_c - s_C c_c & X_o + c_C x_t - s_C y_t \\ s_C c_c + c_C s_c & s_C c_c + c_C c_c & Y_o + s_C x_t + c_C y_t \\ 0 & 0 & 1 \end{bmatrix} \tag{47} $$

which can be reduced to:

$$ {}^1T_t = \begin{bmatrix} cos(C_o + c_t) & -sin(C_o + c_t) & X_o + cos(C_o)x_t - sin(C_o)y_t \\ sin(C_o + c_t) & cos(C_o + c_t) & Y_o + sin(C_o)x_t + cos(C_o)y_t \\ 0 & 0 & 1 \end{bmatrix} \tag{48} $$

The matrix ${}^1T_t$ contains all the information used as input to the inverse kinematics calculation as described in section 1.6. Comparing the left hand side of (17) with (48), we note that (48) represents a new pose:

$$ \begin{aligned} q_x &= X_o + cos(C_o)x_t - sin(C_o)y_t \\ q_x &= Y_o + sin(C_o)x_t + cos(C_o)y_t \\ \phi &= C_o + c_t \end{aligned} \tag{49} $$

Example 4

Given an offset point pose of (200,100,30), (denoting $X_o, Y_o, C_o = \phi$) and a tool point move in ”world” mode of (15, 20, 5), the new pose of the tool point would end up being at, in accordance with (43):

$$ \begin{aligned} X_p &= X_o + \Delta X = 200 + 15 = 215 \\ Y_p &= Y_o + \Delta Y = 100 + 20 = 120 \\ C_p &= C_o + \Delta C = 30 + 5 = 35 \end{aligned} $$

However, if we wanted to make the same move in ”tool” mode, with the tool coordinate system defined by the offset pose, then we would use eqn. (49):

$$ \begin{aligned} X_p &= X_o + cos(C_o)x_t - sin(C_o)y_t = 200 + cos(30) \cdot 15 - sin(30) \cdot 20 = 202.990 \\ Y_p &= Y_o + sin(C_o)x_t + cos(C_o)y_t = 100 + sin(30) \cdot 15 + cos(30) \cdot 20 = 124.821 \\ C_p &= C_o + c_t = 30 + 5 = 35 \end{aligned} $$

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
```

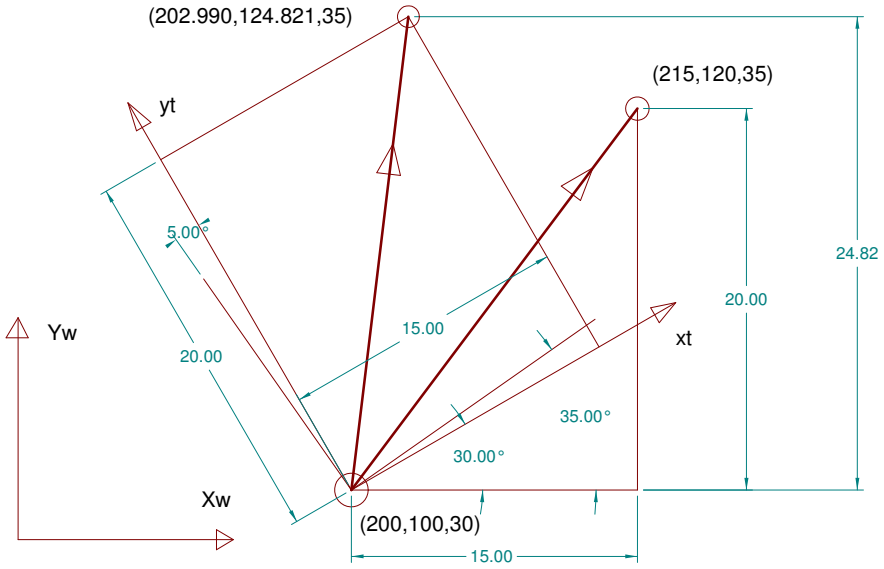

Figure 9: World and tool coordinate move in the X-Y plane

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 robot3kins
```

where ”robot3kins” is the name of our kinsname. 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 3-link robot the entries could be:

```
# set robot DH parameters
setp robot3kins.DH-a1 300
setp robot3kins.DH-a2 400
setp robot3kins.DH-a3 150
```

An example of a kinematics component, applicable to our 3-link serial planer robot, is given below. It has the following structure:

- A set of ”include” declarations which is need to link in other procedure of libraries
- A struct definition to define the ”hal” input links or pins. These are defined in your .hal file.
- 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: robot3kins.c
*   Kinematics for  planer 3 axis robot.
*   This serial arm robot has the following DH parameters:
*           DH_a1, DH_a2, DH_a3: lengths of the three serial links
*   Further parameters:
*   cmode: coordinate mode, ie. 0: "world" mode, 1: "tool" mode
*   setoffs: 0: no action, 1: set current XYC as offsets Xo, Yo, and Co.
*
* 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_bit_t *cmode;
    hal_bit_t *setoffs;
} *haldata;

double Xo = 0;
double Yo = 0;
double Co = 0;

//=======================================================================
int printlog(int np, double f1, double f2, double f3, double f4)
{
// to print stuff in the /var/log/kern.log file-
//            seems only type int can be printed!
int o1,o2,o3,o4;
```

```
      o1 = f1; o2 = f2; o3 = f3; o4 = f4;
      rtapi_print(">> %d %d %d %d %d\n",np, o1,o2,o3,o4);
      return 0;
}
//=================================================================
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);
    int setoffs  = *(haldata->setoffs);
    int cmode = *(haldata->cmode);

    double th1 = joint[0];
    double th2 = joint[1];
    double th3 = joint[2];

    double c1, s1, c12, s12, c123, s123;
    double ux, uy, vx, vy;
    double qx, qy, cw;
    static int prnt = 0;

//          convert degrees to radians
    th1 = th1/180*M_PI;
    th2 = th2/180*M_PI;
    th3 = th3/180*M_PI;

    c1 = cos(th1); s1 = sin(th1);
    c12 = cos(th1+th2); s12 = sin(th1+th2);
    c123 = cos(th1+th2+th3); s123 = sin(th1+th2+th3);

    ux = c123;
    uy = s123;

    vx = -s123;
    vy = c123;

    qx = c1*a1 + c12*a2 + c123*a3;
    qy = s1*a1 + s12*a2 + s123*a3;
    cw = th1+th2+th3;
//
    if (setoffs) {
       Xo = qx; Yo = qy; Co = cw;
    }
```

```
    if (cmode) {
      pos->tran.x = Xo + ux*(qx-Xo) + uy*(qy-Yo);
      pos->tran.y = Yo + vx*(qx-Xo) + vy*(qy-Yo);
      pos->c = cw/M_PI*180;
    } else {
      pos->tran.x = qx;
      pos->tran.y = qy;
      pos->c = cw/M_PI*180;
    }
//
    if (prnt > 0) {
       printlog(1,qx,qy,Xo,Yo);
       prnt = prnt - 1;
    }
    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);
    int cmode = *(haldata->cmode);

    double Cmve = pos->c/180*M_PI;
    double Xmve = pos->tran.x;
    double Ymve = pos->tran.y;

    double th1, th2, th3;
    double cC, sC;
    double ux, uy, vx, vy, px, py, qx, qy;
    double r, k1, k2;
    double c1, s1;
    static int prnt = 0;

    int n1 = 1;
//-----------------------------------------
    if (cmode) {
//                tool coordinates
       sC = sin(Co); cC = cos(Co);
       ux = cC;     vx = -sC;
       uy = sC;     vy =  cC;
       qx = Xo + ux*(Xmve-Xo) + vx*(Ymve-Yo);
```

```
       qy = Yo + uy*(Xmve-Xo) + vy*(Ymve-Yo);
       sC = sin(Cmve); cC = cos(Cmve);
       ux = cC;     vx = -sC;
       uy = sC;     vy =  cC;

       joint[3] = Xo;
       joint[4] = Yo;
       joint[5] = Co/M_PI*180;
       joint[6] = Xmve-Xo;
       joint[7] = Ymve-Yo;
       joint[8] = (Cmve-Co)/M_PI*180;

//-----------------------------------------------
    } else {
//                world coordinates
       sC = sin(Cmve); cC = cos(Cmve);
       ux = cC;     vx = -sC;
       uy = sC;     vy =  cC;
       qx = Xmve;
       qy = Ymve;
//
       joint[3] = Xo;
       joint[4] = Yo;
       joint[5] = Co/M_PI*180;
       joint[6] = Xmve;
       joint[7] = Ymve;
       joint[8] = Cmve/M_PI*180;
    }
//
      px = qx - a3*ux;
      py = qy - a3*uy;

//-------------------------------------------------------------
    if (prnt > 0) {
       printlog(2, Xmve, Ymve, Cmve/M_PI*180, 0);
       printlog(3, ux*100, uy*100, vx*100, vy*100);
       printlog(4, qx, qy, Xo, Yo);
       prnt = prnt - 1;
    }
//-------------------------------------------------------------
//--- th1
      th1 = 0;
      r = sqrt(px*px + py*py);
      if (r == 0) {
//          'ERROR:--------- point not reachable';
         return 1;
      }
```

```
      k1 = atan2(py, px);
      k2 = (px*px + py*py + a1*a1 - a2*a2)/(2*a1*r);

      if (k2 > 1.0) {
//          list('ERROR:--------- point not reachable');
         return 1;
      }
      k2 = acos(k2);
        if (n1 == 1) { th1 = k1 + k2;}
           else      { th1 = k1 - k2 + M_PI;}
//
      c1 = cos(th1); s1 = sin(th1);
//--- th2
      th2 =  0;
      k1 = py - a1*s1;
      k2 = px - a1*c1;
      th2 = atan2(k1, k2) - th1;
//
//--- th3
      th3 = atan2(uy, ux) - th1 - th2;
//
   joint[0] = th1*180/M_PI;
   joint[1] = th2*180/M_PI;
   joint[2] = th3*180/M_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("robot3kins");
    if(comp_id < 0) return comp_id;

    haldata = hal_malloc(sizeof(struct haldata));

    if((res = hal_pin_float_new("robot3kins.DH-a1", HAL_IO,
             &(haldata->DH_a1), comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("robot3kins.DH-a2", HAL_IO,
             &(haldata->DH_a2), comp_id)) < 0) goto error;
    if((res = hal_pin_float_new("robot3kins.DH-a3", HAL_IN,
             &(haldata->DH_a3), comp_id)) < 0) goto error;
    if((res = hal_pin_bit_new("robot3kins.cmode", HAL_IN,
             &(haldata->cmode), comp_id)) < 0) goto error;
    if((res = hal_pin_bit_new("robot3kins.setoffs", HAL_IO,
             &(haldata->setoffs), 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.cmode and robot3kins.setoffs. The ”cmode” pin should be connected to a VCP button (toggle) that can be used to change between ”tool” and ”world” coordinates. The ”setoffs” 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 added VCP):

1. Move in ”world” mode to a offset pose X_o, Y_o, C_o.
2. Do a ”touch off” for all three axes X, Y, C.
3. Activate the ”set offset” button.
4. Toggle the ”tool” mode button to change into tool mode.
5. Perform moves in tool coordinates until finished.
6. Move to zero tool pose.
7. Toggle ”tool” mode button to go back to ”world” mode.

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 3-joint serial robot the following is an example of a Vismach script. The link lengths a1=300, a2=400, and a3=150 mm.

```
#!/usr/bin/python
#     vismach gui for robot-3
#------------------ imports-----------------------------

from vismach import *
import hal

#-------------------HAL pins----------------------------
c = hal.component("robot3gui")
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.ready()

#-------------------model-------------------------------

# finger and axes
tooltip = Capture()
finger = CylinderX(0, 5, -50, 10)
xaxis  = Color([1,0,0,1],[CylinderX(0,3,100,3) ])
yaxis  = Color([0,1,0,1],[CylinderZ(0,3,100,3) ])
zaxis  = Color([0,0,1,1],[CylinderY(0,3,-100,3) ])
finger = Collection([tooltip, finger, xaxis, yaxis, zaxis])

# link 3
link3  = CylinderX(-150,15,-50,15)
joint3 = CylinderY(-25,25,25,25)
joint3 = Translate([joint3],-150,0,0)
link3  = Collection([link3, joint3])
```

```
link3  = Color([0.0,0.9,0.9,1],[link3])
link3  = Collection([finger,link3])
#   move link3 so axis 3 is at origin
link3  = Translate([link3],150,0,0)
#   make joint3 rotate
link3  = HalRotate([link3],c,"joint3",-1,0,1,0)
#   move back to position
link3  = Translate([link3],-150,0,0)

# link 2
link2  = CylinderX(-550,20,-150,20)
joint2 = CylinderY(-30,30,30,30)
joint2 = Translate([joint2],-550,0,0)
link2  = Collection([link2, joint2])
link2  = Color([0.6,0.4,0.9,1],[link2])
link2  = Collection([link2,link3])
#       move link2 so axis 2 is at origin
link2  = Translate([link2],550,0,0)
#   make joint2 rotate
link2  = HalRotate([link2],c,"joint2",-1,0,1,0)
#    move back to position
link2  = Translate([link2],-550,0,0)

# link 1
link1  = CylinderX(-850,20,-550,20)
joint1 = CylinderY(-30,30,30,30)
joint1 = Translate([joint1],-850,0,0)
link1  = Collection([link1, joint1])
link1  = Color([0.0,1.0,0.0,1],[link1])
link1  = Collection([link1,link2])
#          move link1 so axis 1 is at origin
link1  = Translate([link1],850,0,0)
#    make joint1 rotate
link1  = HalRotate([link1],c,"joint1",-1,0,1,0)
#          move back to position
link1  = Translate([link1],-850,0,0)
#    add a base
base   = CylinderZ(-200,60,-15,30)
base   = Color([0.6,0.2,0.4,1],[base])
base   = Translate([base],-850,0,0)
#    add a floor
floor  = Box(-100,-100,-220,100,100,-200)
floor  = Translate([floor],-850,0,0)

work   = Capture()

model  = Collection([link1, base, floor, work])
```

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

The Vismach model resulting from this script is shown in Fig. 1.