INVERSE KINEMATICS EXPLANATION OF 4 DOF TRRR(TRANSLATION, ROTATION,ROTATION,ROTATION) ROBOT

VISUAL COMPONENTS

Contents

Python API for Matrix manipulation – vcMatrix

vcMatrix is a 4x4 matrix and is generally used to perform linear transformations and contain the position and orientation of objects.

Property	Type	Access	Description
N	vcVector	RW	Defines **Normal** vector that is unit vector in X direction.
O	vcVector	RW	Defines **Orientation** vector that is unit vector in Y direction.
A	vcVector	RW	Defines **Approach** vector that is unit vector in Z direction.
P	vcVector	RW	Defines the **Position** vector.
WPR	vcVector	RW	Defines roll, pitch and yaw angles.

Now we will try to discuss the above properties with example so the concept can be clear to the users. We will take a robot which TCP can rotate in all roll-pitch-yaw direction. A good choice is Universal Robot as it has special 6 axis kinematics with python. We can load the attached layout [Kinematics_Solver_example.vcm], here is a Universal Robot UR3 loaded with a box in front of it and also a component called [Co-ordinate system] which will help the user to visualize the matrix manipulations.

Now open the python kinematics script and there you will see the following codes-

```
187     # N - Defines Normal vector that is unit vector in X direction.
188     print "projection of TCP > X on World Frame X axis ",t6.N.X
189
190     # O - Defines Orientation vector that is unit vector in Y direction.
191     print "projection of TCP > Y on World Frame Y axis ",t6.O.Y
192
193     # A - Defines Approach vector that is unit vector in Z direction.
194     print "projection of TCP > Z on World Frame Z axis ",t6.A.Z
195
```

These were written extra out of main inverse kinematics calculation. There are some points taught to the robot and those are P1, P2, P3 and P4. You can click on each of those points and see the values which are being printed. In a summary the following explanations will help.

Normal Vector

Target.N.X – projection of TCP > X axis on World Frame > X axis

Target.N.Y – projection of TCP > X axis on World Frame > Y axis

Target.N.Z – projection of TCP > X axis on World Frame > Z axis

Orientation Vector

Target.O.X – projection of TCP > Y axis on World Frame > X axis

Target.O.Y – projection of TCP > Y axis on World Frame > Y axis

Target.O.Z – projection of TCP > Y axis on World Frame > Z axis

Approach Vector

Target.A.X – projection of TCP > Z axis on World Frame > X axis

Target.A.Y – projection of TCP > Z axis on World Frame > Y axis

Target.A.Z – projection of TCP > Z axis on World Frame > Z axis

The following marked screenshots will further help to understand the concept of Normal, Orientation and Approach vectors. It's important to know and understand the different terms in VC python kinematics which includes Properties, Methods and Events. The VC python modules for solving kinematics are-

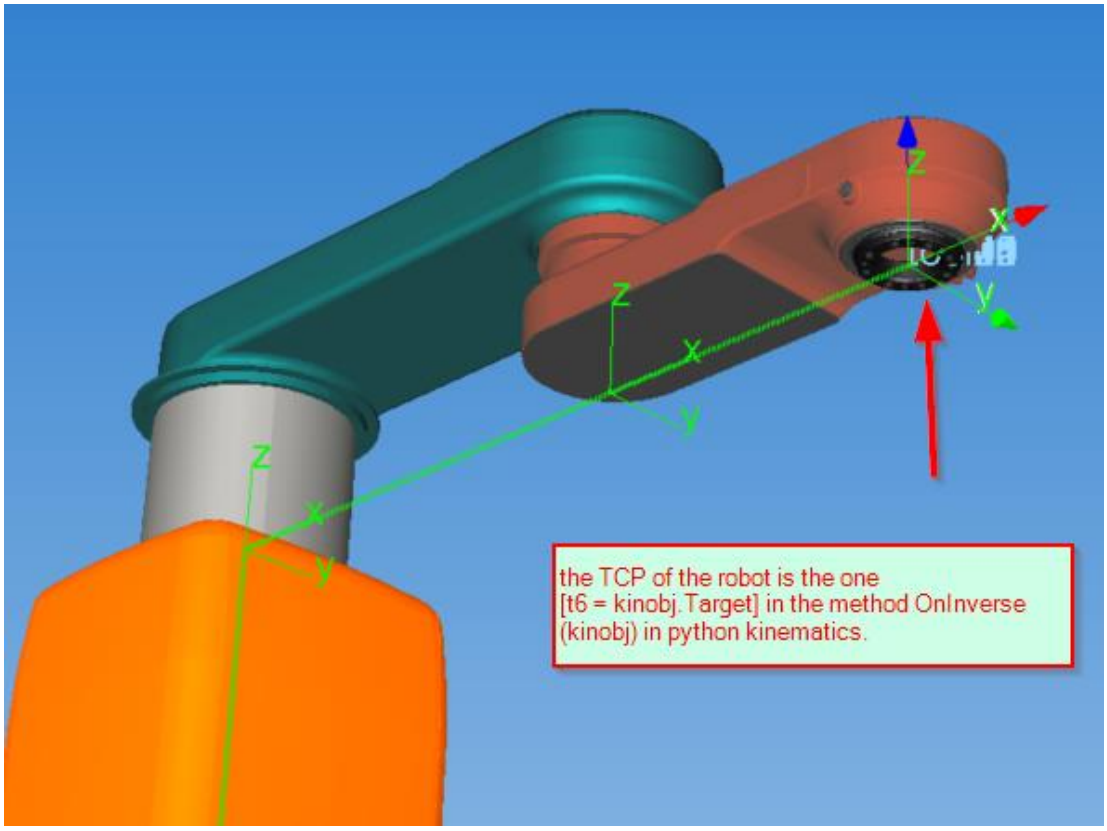
vcMatrix

vcKinematics

vcKinObject

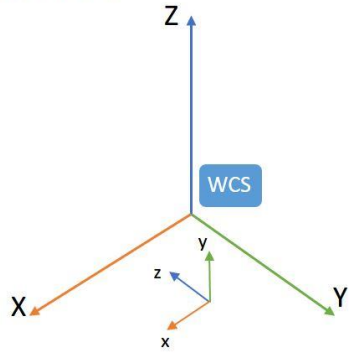
vcMotionTarget

vcMotionInterpolator



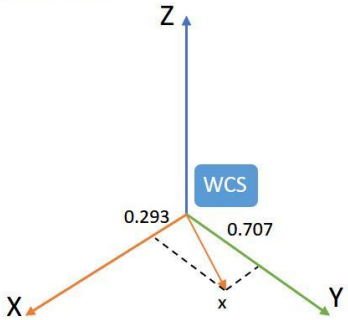
World Coordinate Systems = WCS

Normal Vector



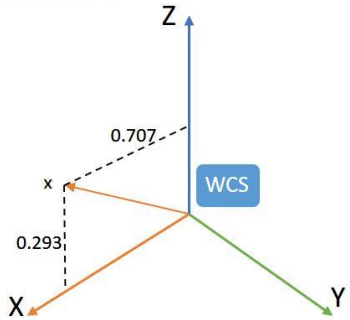
Projection of x on WCS > X axis: 1
Projection of x on WCS > Y axis: 0
Projection of x on WCS > Z axis: 0

Normal Vector



Normal Vector:
target.N.X = 0.293
target.N.Y = 0.707

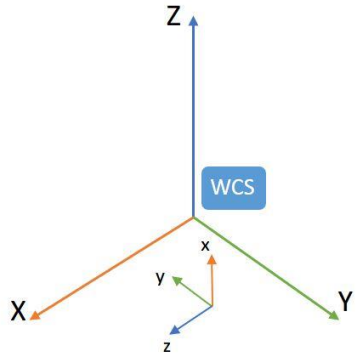
Normal Vector



Normal Vector:
target.N.X = 0.707
target.N.Y = 0.293

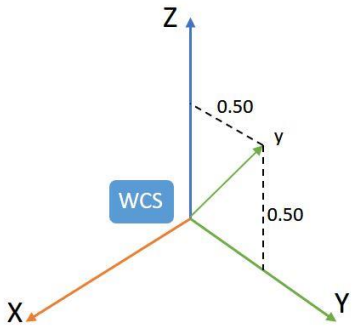
World Coordinate Systems = WCS

Orientation Vector



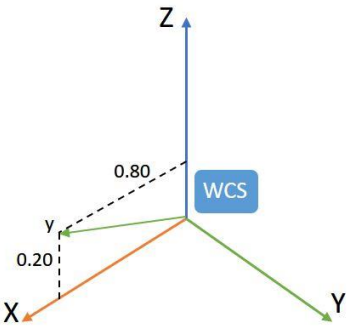
Projection of y on WCS > Y axis: -1
Projection of y on WCS > X axis: 0
Projection of y on WCS > Z axis: 0

Orientation Vector

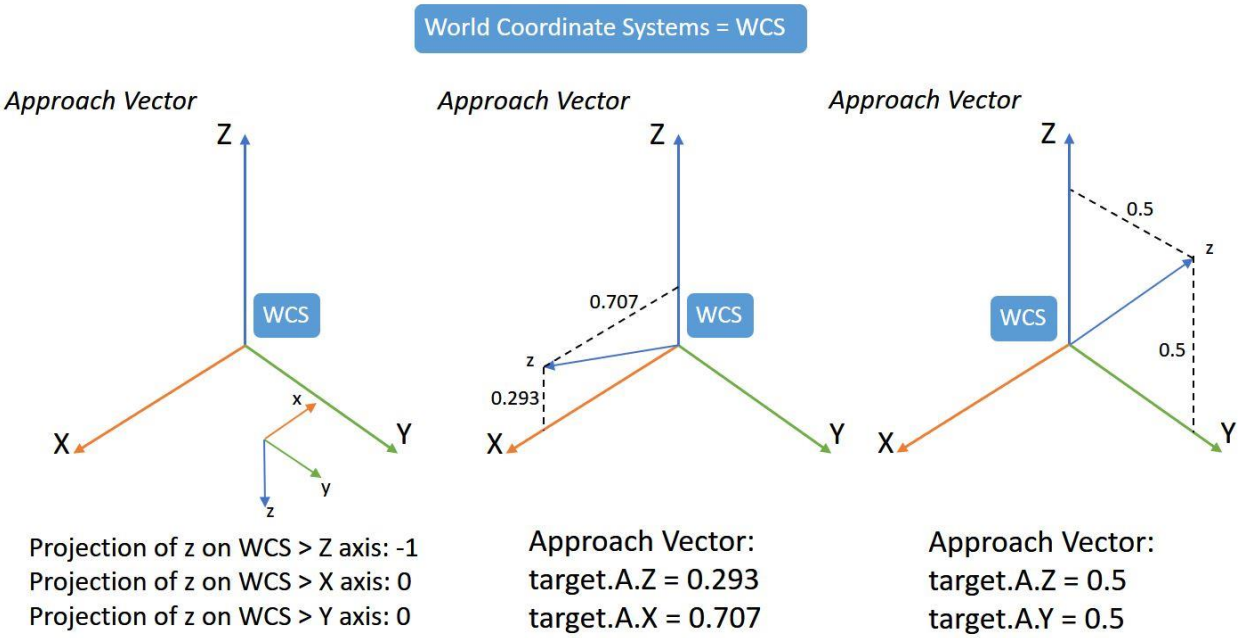


Orientation Vector:
target.O.Y = 0.5
target.O.Z = 0.5

Orientation Vector



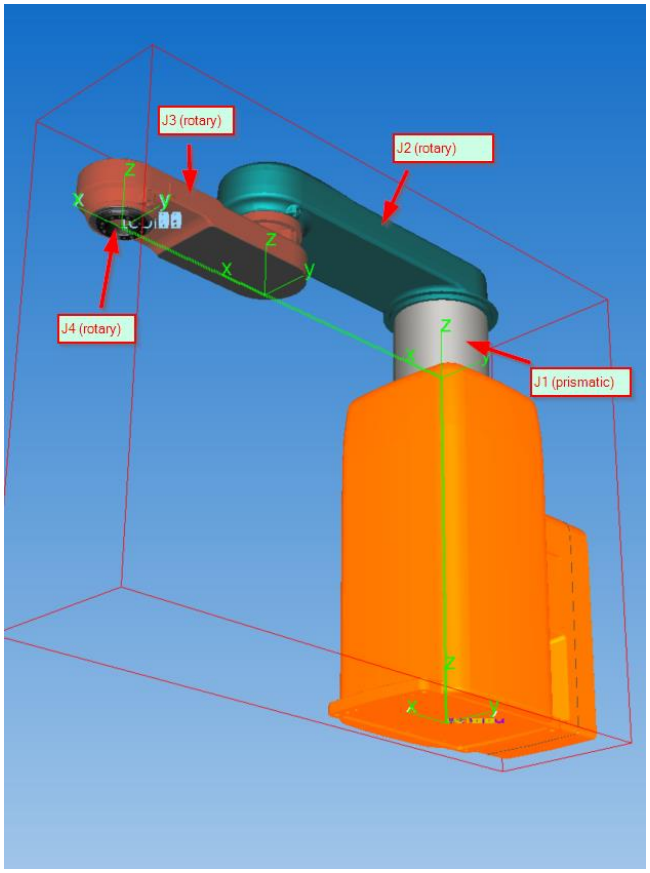
Orientation Vector:
target.O.Y = 0
target.O.Z = 0.2
Target.O.X = 0.8



4DOF TRRR robot

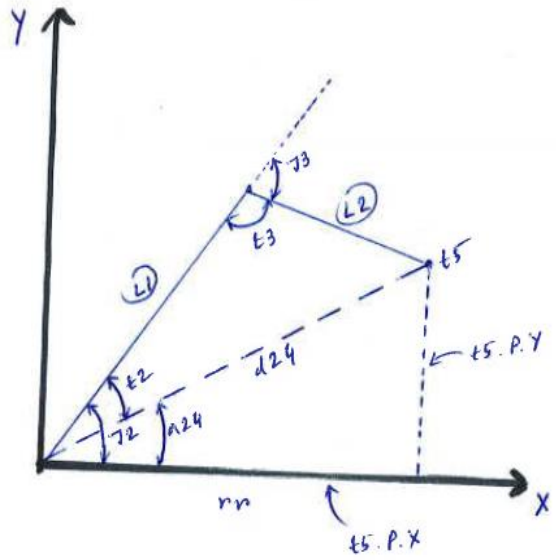
The robot is from Nachi [EZ03-5525] and the Kinematic structure of this robot is:

Translation Joint (J1) > Rotary Joint (J2) > Rotary Joint (J3) > Rotary Joint (J4)

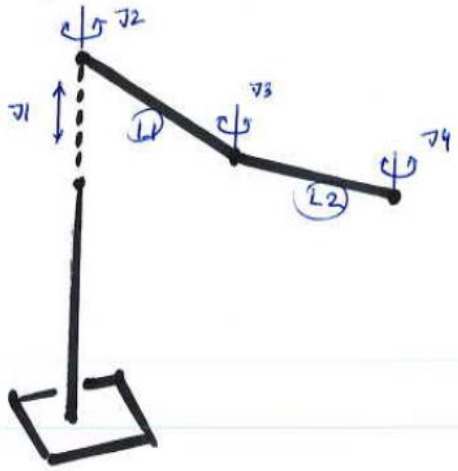


Below is a detailed breakdown of kinematic structure of the robot for solving the inverse kinematic problem.

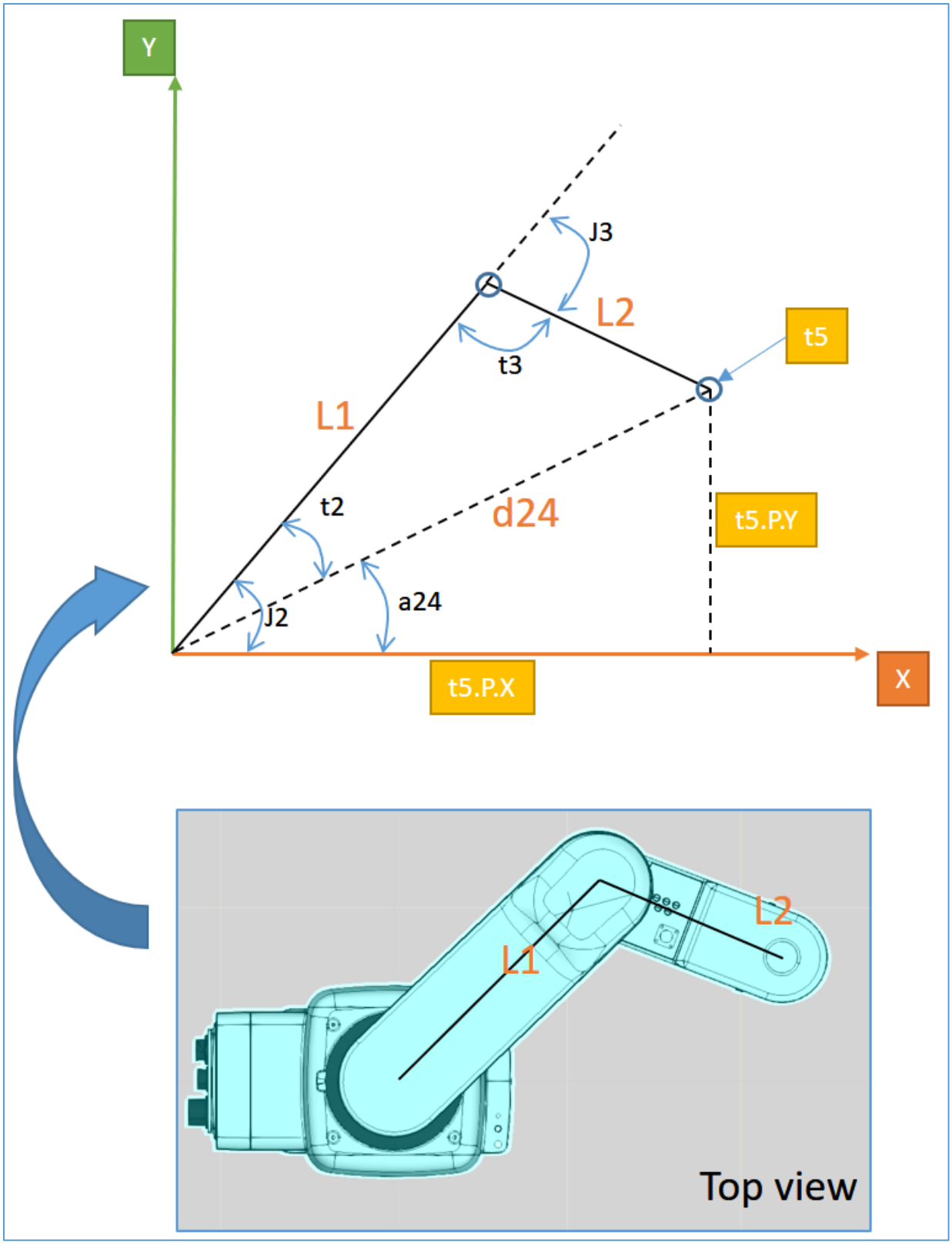
Kinematic solver notations t2, t3, a24, d24, L1, L2



Kinematic Structure



J1 - Translation
J2 - Rotation
J3 - Rotation
J4 - Rotation


Y
J3
L2
t5
t3
L1
t2
d24
t5.P.Y
a24
J2
t5.P.X
X
L2
L1
Top view

Forward and Inverse Kinematics

This is how the forward kinematics looks like inside the python kinematics behavior of the robot component.

```
43  # Returns Kinematic chain target (matrix) value based on joint values
44  def OnForward(kinobj):
45    jv = kinobj.JointValues
46
47    m = vcMatrix.new()
48    m.translateRel( 0, 0, Zoffset+jv[0] )
49    m.rotateRelZ( jv[1] )
50    m.translateRel( L1, 0, 0 )
51    m.rotateRelZ( jv[2] )
52    m.translateRel( L2, 0, 0 )
53    m.rotateRelZ( jv[3] )
54
55    if jv[1] < 0.0:
56      cc = 1
57    else:
58      cc = 0
59
60    kinobj.Configuration = CONFIGS[cc]
61    kinobj.Target = m
62    return True
```

If you go into each link of the robot you will notice that it is exactly the same as the Forward kinematics defined inside the OnForward() method inside python kinematics. It is important to make the OnInverse() method inactive and then select Jog in the [Program] tab and see that the TCP is in right place, if not we need to review our forward kinematics.

Now we will solve the inverse kinematics of this robot. The easiest ones to solve are the J1 and J4 of this robot as one joint J1 translates in only Z direction and another joint J4 rotates along only one axis. J1 and J4 are solved as below.

```
j1 = t5.P.Z-Zoffset
j4 = degrees(atan2(Ny,Nx))
```

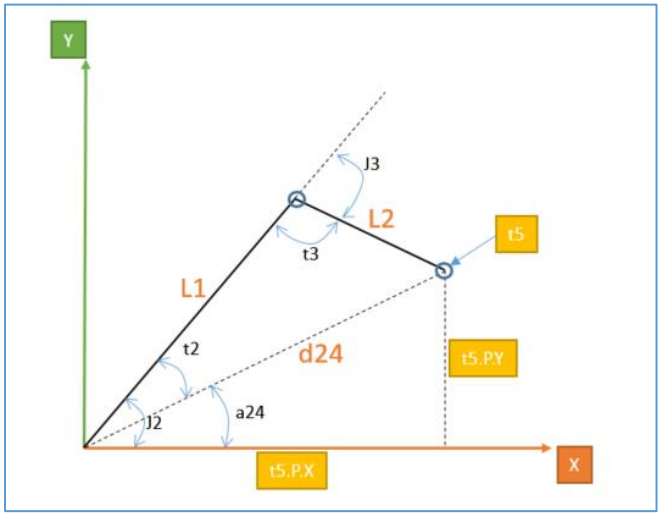
To realize what Ny and Nx means please see the explanation of Normal Vector in the beginning of this documentation. T5.P.Z is simply the World Position Matrix > Z value of the target position.

Now we will solve the rest J2 and J4 as below.

```
for i in range(2):
  if( d24 > L1 + L2 ):
    t2 =   0.0
    t3 = 180.0
    warning = VC_SOLUTION_UNREACHABLE
  elif( d24 < abs(L1 - L2) ):
    t2 = 180.0
    t3 =   0.0
    warning = VC_SOLUTION_UNREACHABLE
  else:
    warning = VC_SOLUTION_REACHABLE
    t2 = degrees(sssa( L2, L1,  d24 ))
    t3 = degrees(sssa(  d24, L1, L2 ))

  j2 = a24-elbow*t2
  j3 = elbow*(180 - t3)
```

Basically what is done here are first the values a24, d24, t5.P.X, t5.P.Y are calculated for every target and then utilizing these values and the trigonometric functions sssa() the value of J2 and J3 calculated.



As this robot has 2 configuration for most positions so there is the iterative loop of –

[..for i in range(2)..] Inside which resides the solution of J2 and J3. The variable **elbow** changes in between [+1,-1] for giving [Righty] or [Lefty] solution. Complete inverse solution looks like below.

```
64  # Returns Kinematic chain joint values based on the target (matrix)
65  def OnInverse(kinobj):
66      t5 = kinobj.Target
67
68      Nx = t5.N.X
69      Ny = t5.N.Y
70      Nz = t5.N.Z
71      #print "Nx,Ny,Nz: ",Nx,Ny,Nz
72      Ax = t5.A.X
73      Ay = t5.A.Y
74      Az = t5.A.Z
75      #print "Ax,Ay,Az: ",Ax,Ay,Az
76
77      solutions = []
78      px = t5.P.X
79      py = t5.P.Y
80      pz = t5.P.Z-Zoffset
81      d24 = sqrt( px*px + py*py )
82      a24 = degrees(atan2( py, px ))
83
84      j1 = t5.P.Z-Zoffset
85      j4 = degrees(atan2(Ny,Nx))
86      elbow = 1
87      for i in range(2):
88          if( d24 > L1 + L2 ):
89              t2 =   0.0
90              t3 = 180.0
91              warning = VC_SOLUTION_UNREACHABLE
92          elif( d24 < abs(L1 - L2) ):
93              t2 = 180.0
94              t3 =   0.0
95              warning = VC_SOLUTION_UNREACHABLE
96          else:
97              warning = VC_SOLUTION_REACHABLE
98              t2 = degrees(sssa( L2, L1,  d24 ))
99              t3 = degrees(sssa(  d24, L1, L2 ))
100
101         j2 = a24-elbow*t2
102         j3 = elbow*(180 - t3)
103         solutions.append( (warning,[j1,j2,j3,j4-j2-j3]) )
104         elbow = -1
105     kinobj.Solutions = solutions
```