WOODSON ENGINEERING PTY LTD
ABN: 45663987424
www.woodsondesign.com.au
PO BOX 2033, HILTON, SA 5033

Woodson Design
Engineering & Building Services

The Python Program for Reaching the Joint1~Joint6 Position

```
#! /usr/bin/env python
# coding=utf-8

import numpy as np
import math


a = [0,
   0,
   408,
   376,
   0,
   0]

alpha = [math.radians(0),
      math.radians(-90),
      math.radians(180),
      math.radians(180),
      math.radians(-90),
      math.radians(90)]

d = [98.5,
   121.5,
   0,
   0,
   102.5,
   94]

theta = [math.radians(180),
      math.radians(-90),
      math.radians(0),
      math.radians(-90),
      math.radians(0),
      math.radians(0)]

def getJointPose(jointList):

   theta = [math.radians(180),
      math.radians(-90),
      math.radians(0),
```

```
        math.radians(-90),
        math.radians(0),
        math.radians(0)]
    for i in range(6):
        theta[i] = theta[i] + jointList[i]

    T = np.identity(4)
    for i in range(6):
        Ti = np.mat([[math.cos(theta[i]),-math.sin(theta[i]),0,a[i]],
            [math.sin(theta[i])*math.cos(alpha[i]),math.cos(theta[i])*math.cos(alpha[i]),-math.sin(alpha[i]),-
d[i]*math.sin(alpha[i])],

[math.sin(theta[i])*math.sin(alpha[i]),math.cos(theta[i])*math.sin(alpha[i]),math.cos(alpha[i]),d[i]*math.cos(alpha[i])],
            [0,0,0,1]])
        T = T*Ti
    jointTool_pos = [T[0,3],T[1,3],T[2,3]]

    T = np.identity(4)
    for i in range(5):
        Ti = np.mat([[math.cos(theta[i]),-math.sin(theta[i]),0,a[i]],
            [math.sin(theta[i])*math.cos(alpha[i]),math.cos(theta[i])*math.cos(alpha[i]),-math.sin(alpha[i]),-
d[i]*math.sin(alpha[i])],

[math.sin(theta[i])*math.sin(alpha[i]),math.cos(theta[i])*math.sin(alpha[i]),math.cos(alpha[i]),d[i]*math.cos(alpha[i])],
            [0,0,0,1]])
        T = T*Ti
    joint6_pos = [T[0,3],T[1,3],T[2,3]]

    T = np.identity(4)
    for i in range(4):
        Ti = np.mat([[math.cos(theta[i]),-math.sin(theta[i]),0,a[i]],
            [math.sin(theta[i])*math.cos(alpha[i]),math.cos(theta[i])*math.cos(alpha[i]),-math.sin(alpha[i]),-
d[i]*math.sin(alpha[i])],

[math.sin(theta[i])*math.sin(alpha[i]),math.cos(theta[i])*math.sin(alpha[i]),math.cos(alpha[i]),d[i]*math.cos(alpha[i])],
            [0,0,0,1]])
        T = T*Ti
    joint5_pos = [T[0,3],T[1,3],T[2,3]]

    T = np.identity(4)
    d[3] = -102.5
    for i in range(4):
        Ti = np.mat([[math.cos(theta[i]),-math.sin(theta[i]),0,a[i]],
```

```
        [math.sin(theta[i])*math.cos(alpha[i]),math.cos(theta[i])*math.cos(alpha[i]),-math.sin(alpha[i]),-
d[i]*math.sin(alpha[i])],

[math.sin(theta[i])*math.sin(alpha[i]),math.cos(theta[i])*math.sin(alpha[i]),math.cos(alpha[i]),d[i]*math.cos(alpha[i])],
        [0,0,0,1]])
    T = T*Ti
  joint4_pos = [T[0,3],T[1,3],T[2,3]]

  T = np.identity(4)
  for i in range(3):
    Ti = np.mat([[math.cos(theta[i]),-math.sin(theta[i]),0,a[i]],
        [math.sin(theta[i])*math.cos(alpha[i]),math.cos(theta[i])*math.cos(alpha[i]),-math.sin(alpha[i]),-
d[i]*math.sin(alpha[i])],

[math.sin(theta[i])*math.sin(alpha[i]),math.cos(theta[i])*math.sin(alpha[i]),math.cos(alpha[i]),d[i]*math.cos(alpha[i])],
        [0,0,0,1]])
    T = T*Ti
  joint3_pos = [T[0,3],T[1,3],T[2,3]]

  T = np.identity(4)
  for i in range(2):
    Ti = np.mat([[math.cos(theta[i]),-math.sin(theta[i]),0,a[i]],
        [math.sin(theta[i])*math.cos(alpha[i]),math.cos(theta[i])*math.cos(alpha[i]),-math.sin(alpha[i]),-
d[i]*math.sin(alpha[i])],

[math.sin(theta[i])*math.sin(alpha[i]),math.cos(theta[i])*math.sin(alpha[i]),math.cos(alpha[i]),d[i]*math.cos(alpha[i])],
        [0,0,0,1]])
    T = T*Ti
  joint2_pos = [T[0,3],T[1,3],T[2,3]]

  T = np.identity(4)
  for i in range(1):
    Ti = np.mat([[math.cos(theta[i]),-math.sin(theta[i]),0,a[i]],
        [math.sin(theta[i])*math.cos(alpha[i]),math.cos(theta[i])*math.cos(alpha[i]),-math.sin(alpha[i]),-
d[i]*math.sin(alpha[i])],

[math.sin(theta[i])*math.sin(alpha[i]),math.cos(theta[i])*math.sin(alpha[i]),math.cos(alpha[i]),d[i]*math.cos(alpha[i])],
        [0,0,0,1]])
    T = T*Ti
  joint1_pos = [T[0,3],T[1,3],T[2,3]]

  joint_pos = [joint1_pos,joint2_pos,joint3_pos,joint4_pos,joint5_pos,joint6_pos,jointTool_pos]
  return joint_pos
```

```
if __name__ == '__main__':
    jointAngle = [math.radians(20),
                  math.radians(-20),
                  math.radians(20),
                  math.radians(50),
                  math.radians(70),
                  math.radians(0)]
    pos = getJointPose(jointAngle)
    for i in range(7):
        print "joint[%d] XYZ = %f, %f, %f"%(i+1,pos[i][0],pos[i][1],pos[i][2])
```

The Python Program for Checking if in the Cylindric Space

```
#! /usr/bin/env python
# coding=utf-8

import math

point1 = [0,0,0]
point2 = [0,0,1]
r = 0.5

def getVector(list1,list2):
    list_ret = [0,0,0]
    for i in range(len(list1)):
        list_ret[i] = list2[i] - list1[i]
    return list_ret

def V1XV2(list1,list2):
    list_ret = [0,0,0]
    list_ret[0] = list1[1]*list2[2] - list1[2]*list2[1]
    list_ret[1] = list1[2]*list2[0] - list1[0]*list2[2]
    list_ret[2] = list1[0]*list2[1] - list1[1]*list2[0]
```

```
    return list_ret

def printVector(list1):
    print "Vector is: "
    for i in range(len(list1)):
        print "list[%d] = %f"%(i,list1[i])

def getVectorMod(list1):
    sum = 0
    for i in range(len(list1)):
        sum = sum + list1[i]*list1[i]
    return math.sqrt(sum)

def PlainX(point_target,point_O,Vector_n):
    x = point_target[0]
    y = point_target[1]
    z = point_target[2]

    ret = Vector_n[0]*(x - point_O[0]) + Vector_n[1]*(y - point_O[1]) + Vector_n[2]*(z - point_O[2])
    return ret

def getSign(num1):
    if(num1 >= 0):
        return 1
    else:
        return -1


def IsInBox(point_target,point1,point2,r):

    p1 = point1
    p2 = point2
    V_n = getVector(p1,p2)

    signBottom = getSign(PlainX(point2,p1,V_n))
    signBottom_t = getSign(PlainX(point_target,p1,V_n))
    signTop = getSign(PlainX(point1,p2,V_n))
    signTop_t = getSign(PlainX(point_target,p2,V_n))
    V_p1pt = getVector(p1,point_target)
    d = getVectorMod(V1XV2(V_n,V_p1pt))/getVectorMod(V_n)
    print "d = %f"%(d)

    if((signBottom == signBottom_t) and (signTop == signTop_t) and (d < r)):
```

```
        return 0
    else:
        return 1


if __name__ == '__main__':
    point1 = [0,0,0]
    point2 = [0,3,3]
    r = 2
    point = [0,4,4]
    print IsInBox(point,point1,point2,r)
```

C++ Program for Checking if in the Cylindric Space

```
#include "mathoperate.h"
#include <iostream>
#include <math.h>

mathoperate::mathoperate()
{
}

void mathoperate::getRobotPose(double jointangle[6], double robotpose[5][3])
{
   double theta[6] = {3.141592,   -1.570796,  0,  -1.570796,  0,  0};
   double d[6] = {98.5,   121.5,  0,  0,  102.5,  94};
   for(int i = 0; i<6; i++)
   {
      theta[i] = theta[i] + jointangle[i];
   }
   double T[4][4] = {{1,0,0,0},
                     {0,1,0,0},
                     {0,0,1,0},
                     {0,0,0,1}};
   for(int i=0; i<6; i++)
   {
      double mat[4][4] = { { cos(theta[i]),   -sin(theta[i]),  0,  a[i]},
                           {sin(theta[i])*cos(alpha[i]),   cos(theta[i])*cos(alpha[i]),   -sin(alpha[i]),    -d[i]*sin(alpha[i]) },
                           { sin(theta[i])*sin(alpha[i]),   cos(theta[i])*sin(alpha[i]),   cos(alpha[i]),    d[i]*cos(alpha[i])   },
                           {0, 0,  0,  1}
                        };
      arrayMulti(T,mat,T);
   }
   robotpose[0][0] = T[0][3];
   robotpose[0][1] = T[1][3];
   robotpose[0][2] = T[2][3];
    T[0][0] = 1;   T[0][1] = 0;   T[0][2] = 0;   T[0][3] = 0;
    T[1][0] = 0;   T[1][1] = 1;   T[1][2] = 0;   T[1][3] = 0;
    T[2][0] = 0;   T[2][1] = 0;   T[2][2] = 1;   T[2][3] = 0;
    T[3][0] = 0;   T[3][1] = 0;   T[3][2] = 0;   T[3][3] = 1;
   for(int i=0; i<5; i++)
   {
      double mat[4][4] = { { cos(theta[i]),   -sin(theta[i]),  0,  a[i]},
```

```
                    {sin(theta[i])*cos(alpha[i]),   cos(theta[i])*cos(alpha[i]),  -sin(alpha[i]),   -d[i]*sin(alpha[i]) },
                    { sin(theta[i])*sin(alpha[i]),  cos(theta[i])*sin(alpha[i]),  cos(alpha[i]),    d[i]*cos(alpha[i])  },
                    {0, 0,  0,  1}
                };
   arrayMulti(T,mat,T);
}
robotpose[1][0] = T[0][3];
robotpose[1][1] = T[1][3];
robotpose[1][2] = T[2][3];
 T[0][0] = 1;   T[0][1] = 0;   T[0][2] = 0;   T[0][3] = 0;
 T[1][0] = 0;   T[1][1] = 1;   T[1][2] = 0;   T[1][3] = 0;
 T[2][0] = 0;   T[2][1] = 0;   T[2][2] = 1;   T[2][3] = 0;
 T[3][0] = 0;   T[3][1] = 0;   T[3][2] = 0;   T[3][3] = 1;
for(int i=0; i<4; i++)
{
   double mat[4][4] = { { cos(theta[i]),  -sin(theta[i]),  0,  a[i]},
                    {sin(theta[i])*cos(alpha[i]),   cos(theta[i])*cos(alpha[i]),  -sin(alpha[i]),   -d[i]*sin(alpha[i]) },
                    { sin(theta[i])*sin(alpha[i]),  cos(theta[i])*sin(alpha[i]),  cos(alpha[i]),    d[i]*cos(alpha[i])  },
                    {0, 0,  0,  1}
                };
   arrayMulti(T,mat,T);
}
robotpose[2][0] = T[0][3];
robotpose[2][1] = T[1][3];
robotpose[2][2] = T[2][3];
d[3] = -102.5;
 T[0][0] = 1;   T[0][1] = 0;   T[0][2] = 0;   T[0][3] = 0;
 T[1][0] = 0;   T[1][1] = 1;   T[1][2] = 0;   T[1][3] = 0;
 T[2][0] = 0;   T[2][1] = 0;   T[2][2] = 1;   T[2][3] = 0;
 T[3][0] = 0;   T[3][1] = 0;   T[3][2] = 0;   T[3][3] = 1;
for(int i=0; i<4; i++)
{
   double mat[4][4] = { { cos(theta[i]),  -sin(theta[i]),  0,  a[i]},
                    {sin(theta[i])*cos(alpha[i]),   cos(theta[i])*cos(alpha[i]),  -sin(alpha[i]),   -d[i]*sin(alpha[i]) },
                    { sin(theta[i])*sin(alpha[i]),  cos(theta[i])*sin(alpha[i]),  cos(alpha[i]),    d[i]*cos(alpha[i])  },
                    {0, 0,  0,  1}
                };
   arrayMulti(T,mat,T);
}
robotpose[3][0] = T[0][3];
robotpose[3][1] = T[1][3];
robotpose[3][2] = T[2][3];
 T[0][0] = 1;   T[0][1] = 0;   T[0][2] = 0;   T[0][3] = 0;
```

```
    T[1][0] = 0;   T[1][1] = 1;    T[1][2] = 0;    T[1][3] = 0;
    T[2][0] = 0;   T[2][1] = 0;    T[2][2] = 1;    T[2][3] = 0;
    T[3][0] = 0;   T[3][1] = 0;    T[3][2] = 0;    T[3][3] = 1;
   for(int i=0; i<3; i++)
   {
      double mat[4][4] = { { cos(theta[i]),   -sin(theta[i]),  0,  a[i]},
                           {sin(theta[i])*cos(alpha[i]),   cos(theta[i])*cos(alpha[i]),   -sin(alpha[i]),    -d[i]*sin(alpha[i]) },
                           { sin(theta[i])*sin(alpha[i]),   cos(theta[i])*sin(alpha[i]),   cos(alpha[i]),    d[i]*cos(alpha[i])   },
                           {0, 0,  0,  1}
                        };
      arrayMulti(T,mat,T);
   }
   robotpose[4][0] = T[0][3];
   robotpose[4][1] = T[1][3];
   robotpose[4][2] = T[2][3];

}

void mathoperate::arrayMulti(double a[][4], double b[][4], double c[][4])
{
   int i,j,k;
   int N=4;
   double d[4][4] = {};

   for(i=0;i<N;i++)
   {
      for(j=0;j<N;j++)
      {
         d[i][j] = 0;
      }
   }

   for (k = 0; k < N; k++) {
      for(i = 0; i < N; i++){
         for (j = 0; j < N; j++) {
            d[i][k] += a[i][j] * b[j][k];
         }
      }
   }

   for(i = 0; i<N; i++)
   {
      for(j=0; j<N; j++)
```

```
      {
         c[i][j] = d[i][j];
      }
   }
}

void mathoperate::getVector(double list1[], double list2[], double *list_ret)
{
   for(int i=0; i<3; i++)
   {
      list_ret[i] = list2[i] - list1[i];
   }
}

void mathoperate::V1xV2(double list1[], double list2[], double *list_ret)
{
   list_ret[0] = list1[1]*list2[2] - list1[2]*list2[1];
   list_ret[1] = list1[2]*list2[0] - list1[0]*list2[2];
   list_ret[2] = list1[0]*list2[1] - list1[1]*list2[0];
}

double mathoperate::getVectorMod(double list[])
{
   double sum = 0;
   for(int i=0; i<3; i++)
   {
      sum = sum + list[i]*list[i];
   }
   return sqrt(sum);
}

double mathoperate::PlainX(double targetPoint[], double pointO[], double vectorN[])
{
   double x = targetPoint[0];
   double y = targetPoint[1];
   double z = targetPoint[2];
   if( vectorN[0]*(x - pointO[0]) + vectorN[1]*(y - pointO[1]) + vectorN[2]*(z - pointO[2]) >= 0)
      return 1;
   else
      return -1;
}

int mathoperate::IsInBox(double targetPoint[], double pointBottom[], double pointTop[], double r)
```

```
{
   double V_n[3] = {};
   getVector(pointBottom,pointTop,V_n);
   int signBottom = PlainX(pointTop,pointBottom,V_n);
   int signBottom_t = PlainX(targetPoint,pointBottom,V_n);
   int signTop = PlainX(pointBottom,pointTop,V_n);
   int signTop_t = PlainX(targetPoint,pointTop,V_n);
   double V_pBpT[3] = {};
   getVector(pointBottom,targetPoint,V_pBpT);
   double V_tmp[3] = {};
   V1xV2(V_n,V_pBpT,V_tmp);
   double distance = getVectorMod(V_tmp)/getVectorMod(V_n);

   if((signBottom == signBottom_t) && (signTop == signTop_t) && (distance < r))
      return 0;
   else
      return 1;
}
```

Python Program for Checking whether the Point is in the Box

```
def getVector(list1,list2):
   list_ret = [0,0,0]
   for i in range(len(list1)):
      list_ret[i] = list2[i] - list1[i]
   return list_ret

def printVector(list1):
   print "Vector is: "
   for i in range(len(list1)):
      print "list[%d] = %f"%(i,list1[i])

def V1XV2(list1,list2):
   list_ret = [0,0,0]
   list_ret[0] = list1[1]*list2[2] - list1[2]*list2[1]
   list_ret[1] = list1[2]*list2[0] - list1[0]*list2[2]
   list_ret[2] = list1[0]*list2[1] - list1[1]*list2[0]
   return list_ret
```

```
def getSign(num1):
   if(num1 >= 0):
      return 1
   else:
      return -1

def PlainX(point_target,point_O,point_X,point_Y):
   x = point_target[0]
   y = point_target[1]
   z = point_target[2]
   v_X = getVector(point_O,point_X)
   v_Y = getVector(point_O,point_Y)
   V_n = V1XV2(v_X,v_Y)

   ret = V_n[0]*(x - point_O[0]) + V_n[1]*(y - point_O[1]) + V_n[2]*(z - point_O[2])
   return ret

def IsInBox(point_target,point1,point2,point3,point4):
      p1 = point1
      p2 = point2
      p3 = [point2[0] + point3[0] - point1[0], point2[1] + point3[1] - point1[1], point2[2] + point3[2] - point1[2]]
      p4 = point3
      p5 = point4
      p6 = [p2[0] + p5[0] - p1[0], p2[1] + p5[1] - p1[1], p2[2] + p5[2] - p1[2]]
      p7 = [p6[0] + p3[0] - p2[0], p6[1] + p3[1] - p2[1], p6[2] + p3[2] - p2[2]]
      p8 = [p5[0] + p7[0] - p6[0], p5[1] + p7[1] - p6[1], p5[2] + p7[2] - p6[2]]
      signFront = getSign(PlainX(point3,p1,p2,p5))
      signFront_t = getSign(PlainX(point_target,p1,p2,p5))
      signBack = getSign(PlainX(point1,p4,p3,p8))
      signBack_t = getSign(PlainX(point_target,p4,p3,p8))
      signLeft = getSign(PlainX(point2,p1,p4,p5))
      signLeft_t = getSign(PlainX(point_target,p1,p4,p5))
      signRight = getSign(PlainX(point1,p2,p3,p6))
      signRight_t = getSign(PlainX(point_target,p2,p3,p6))
      signTop = getSign(PlainX(point2,p5,p6,p8))
      signTop_t = getSign(PlainX(point_target,p5,p6,p8))
      signBottom = getSign(PlainX(point4,p1,p2,p4))
      signBottom_t = getSign(PlainX(point_target,p1,p2,p4))

      if((signFront == signFront_t) and (signBack == signBack_t) and
            (signLeft == signLeft_t) and (signRight == signRight_t) and
            (signTop == signTop_t) and (signBottom == signBottom_t)):
```

```
            print "Inside the Box!"
        else:
            print "Warning!! Outside the Box!"


if __name__ == '__main__':
    point1 = [0,0,0]
    point2 = [1,0,0]
    point3 = [0,4,0]
    point4 = [0,0,2]
    point = [0.2,2.5,1.2]
    IsInBox(point,point1,point2,point3,point4)
```