PROJECT REPORT

ECE 569 - INTRODUCTION TO ROBOTIC SYSTEMS

By: Manas Paldhe
Under Supervision of Prof. George Lee

Introduction:
OpenRAVE is an open-source platform used to simulate and plan motion of many robots. The software allows seamless creation and interface of many plugins. The software was developed by Rosen Diankov at CMU.

During the course of this project using a PUMA 560 robot, forward and inverse kinematics was studied. For various input angles the position and orientation of the gripper was calculated. Using this position and orientation and the decision equations the original configuration was re-calculated. The error is studied.

Forward Kinematics:
Notation:
theta[i-1], d[i-1], alpha[i-1] and a[i] are the joint parameters corresponding to the joint i. This notation was followed for the ease of programming (The array index begins from 0)

In order to obtain the orientation and position of a link with respect to another link, HTM matrices were used.
By applying chain rule the position and orientation of gripper was obtained.

```
(i-1)A(i)=
[float(cos(theta)   -float(cos(alpha)float(sin(theta)        float(sin(alpha)float(sin(theta)       afloat(cos(theta)
float(sin(theta)    float(cos(alpha)float(cos(theta)       -float(sin(alpha)float(cos(theta)      afloat(sin(theta)
0                   float(sin(alpha)                         float(cos(alpha)                       d
0                   0                                        0                                                        1]


0A1=
[float(cos(theta[0])),      -float(cos(alpha[0]))*float(sin(theta[0])),     float(sin(alpha[0]))*float(sin(theta[0])),
        float(a[0])*float(cos(theta[0])),
float(sin(theta[0])),       float(cos(alpha[0]))*float(cos(theta[0])),      -float(sin(alpha[0]))*float(cos(theta[0])),
        float(a[0])*float(sin(theta[0])),
0,              float(sin(alpha[0])),               float(cos(alpha[0])),               float(d[0]),
0,              0,                  0,                  1               ]


1A2=
[float(cos(theta[1])),      -float(cos(alpha[1]))*float(sin(theta[1])),     float(sin(alpha[1]))*float(sin(theta[1])),
        float(a[1])*float(cos(theta[1])),
float(sin(theta[1])),       float(cos(alpha[1]))*float(cos(theta[1])),      -float(sin(alpha[1]))*float(cos(theta[1])),
        float(a[1])*float(sin(theta[1])),
0,              float(sin(alpha[1])),               float(cos(alpha[1])),               float(d[1]),
0,              0,                  0,                  1               ]


2A3=
[float(cos(theta[2])),      -float(cos(alpha[2]))*float(sin(theta[2])),     float(sin(alpha[2]))*float(sin(theta[2])),
        float(a[2])*float(cos(theta[2])),
float(sin(theta[2])),       float(cos(alpha[2]))*float(cos(theta[2])),      -float(sin(alpha[2]))*float(cos(theta[2])),
        float(a[2])*float(sin(theta[2])),
0,              float(sin(alpha[2])),               float(cos(alpha[2])),               float(d[2]),
0,              0,                  0,                  1               ]
```

3A4=
[float(cos(theta[3])), -float(cos(alpha[3]))*float(sin(theta[3])), float(sin(alpha[3]))*float(sin(theta[3])), float(a[3])*float(cos(theta[3])),
float(sin(theta[3])), float(cos(alpha[3]))*float(cos(theta[3])), -float(sin(alpha[3]))*float(cos(theta[3])), float(a[3])*float(sin(theta[3])),
0, float(sin(alpha[4])), float(cos(alpha[4])), float(d[4]),
0, 0, 0, 1]

4A5=
[float(cos(theta[4])), -float(cos(alpha[4]))*float(sin(theta[4])), float(sin(alpha[4]))*float(sin(theta[4])), float(a[4])*float(cos(theta[4])),
float(sin(theta[4])), float(cos(alpha[4]))*float(cos(theta[4])), -float(sin(alpha[4]))*float(cos(theta[4])), float(a[4])*float(sin(theta[4])),
0, float(sin(alpha[4])), float(cos(alpha[4])), float(d[4]),
0, 0, 0, 1]

5A6=
[float(cos(theta[5])), -float(cos(alpha[5]))*float(sin(theta[5])), float(sin(alpha[5]))*float(sin(theta[5])), float(a[5])*float(cos(theta[5])),
float(sin(theta[5])), float(cos(alpha[5]))*float(cos(theta[5])), -float(sin(alpha[5]))*float(cos(theta[5])), float(a[5])*float(sin(theta[5])),
0, float(sin(alpha[5])), float(cos(alpha[5])), float(d[5]),
0, 0, 0, 1]

Multiplying the matrices gives the position of the gripper.

A_02=matrixmult (A_01,A_12)
A_24=matrixmult (A_23,A_34)
A_46=matrixmult (A_45,A_56)
A_04=matrixmult (A_02,A_24)
A_06=matrixmult (A_04,A_46)

Here matrixmult is a function that multiplies the matrices.

In order to get the configuration the following equations were used:

```
arm_calc = (-d[3]*sin(theta[1]+theta[2]) -a[2]*cos(theta[1]+theta[2]) -a[1]*cos(theta[1]) )
if arm_calc>=0:
	ARM= 1
else:
	ARM=-1



elbow_calc= d[3]*cos(theta[2]) - a[2]*sin(theta[2])
if elbow_calc>=0:
	ELBOW=ARM
else:
	ELBOW=-ARM

wrist_calc_s= s[0]*z4[0] + s[1]*z4[1] +s[2]*z4[2]
wrist_calc_n= n[0]*z4[0] + n[1]*z4[1] +n[2]*z4[2]

if wrist_calc_s>0:
	WRIST=1
if wrist_calc_s<0:
	WRIST=-1
if wrist_calc_s==0:
	if wrist_calc_n >= 0:
		WRIST=1
	else:
		WRIST=-1
```

Inverse Kinematics:
Given the end effector position and orientation, obtaining the robot configuration is required quite many a times.
In order to do so we use three methods:
1. Circle Equation:

In the circle quation to solve a*sin(x) + b*cos(x) = c
We use
r= sqrt(a^2 + b^2)
and let a/r=cos(y) and b/r=sin(y)
Dividing both sides by r gives
sin (x+y) = c/r
or x = arcsin(c/r) - y

Thus we obtain the required angles.

Obtained solution for PUMA 560 robot:
All possible solutions:

```
calculated_thetaA[0] = 180/pi*(atan2(py,px) - atan2(d[1],+sqrt(px*px+py*py-d[1]*d[1])))
calculated_thetaB[0] = 180/pi*(atan2(py,px) - atan2(d[1],-sqrt(px*px+py*py-d[1]*d[1])))

g214 = cos(theta[0])*px + sin(theta[0])*py
g224 = -pz
d_const = g214*g214 + g224*g224 -d[3]*d[3] -a[2]*a[2] -a[1]*a[1]
e_const_square= 4*a[1]*a[1]*a[2]*a[2] + 4*a[1]*a[1]*d[3]*d[3]

calculated_thetaA[2] = 180/pi*(atan2(d_const, sqrt(e_const_square-d_const*d_const)) -
atan2(a[2],d[3]))
calculated_thetaB[2] = 180/pi*(atan2(d_const, -sqrt(e_const_square-d_const*d_const)) -
atan2(a[2],d[3]))

f=g214-a[1]*cos(theta[2])

pxx=-(d[3]*cos(theta[2]) -a[2]*sin(theta[2]) )
pyy=d[3]*sin(theta[2]) +a[2]*cos(theta[2]) + a[1]

rad=g214
```

```
calculated_thetaA[1]=180/pi*(atan2(pyy,pxx) -atan2(rad,sqrt((pxx*pxx + pyy*pyy)-rad*rad)) )
calculated_thetaB[1]=180/pi*(atan2(pyy,pxx) -atan2(rad,-sqrt((pxx*pxx + pyy*pyy)-rad*rad)) )

calculated_thetaA[3]= 180/pi*atan2( -sin(theta[0])*ax + cos(theta[0])*ay ,
cos(theta[1]+theta[2])*(cos(theta[0])*ax +sin(theta[0])*ay) - az*(sin(theta[1]+theta[2])) )
calculated_thetaB[3]= 180 + 180/pi*atan2( -sin(theta[0])*ax + cos(theta[0])*ay ,
cos(theta[1]+theta[2])*(cos(theta[0])*ax +sin(theta[0])*ay) - az*(sin(theta[1]+theta[2])) )

calculated_thetaA[4]=180/pi*atan2(
cos(theta[3])*(cos(theta[1]+theta[2])*(cos(theta[0])*ax+sin(theta[0])*ay)-
sin(theta[1]+theta[2])*az)+sin(theta[3])*(-sin(theta[0])*ax +cos(theta[0])*ay)    ,
sin(theta[1]+theta[2])*(cos(theta[0])*ax + sin(theta[0])*ay) +az*cos(theta[1]+theta[2])  )

calculated_thetaA[5]=180/pi*atan2(-
sin(theta[3])*(cos(theta[1]+theta[2])*(cos(theta[0])*nx+sin(theta[0])*ny)  -
sin(theta[1]+theta[2])*nz) + cos(theta[3])*(-sin(theta[0])*nx + cos(theta[0])*ny) , -
sin(theta[3])*(cos(theta[1]+theta[2])*(cos(theta[0])*sx+sin(theta[0])*sy)  -
sin(theta[1]+theta[2])*sz) + cos(theta[3])*(-sin(theta[0])*sx + cos(theta[0])*sy) )
```

The actual solution:

```
if ARM==1:
        theta[0] = calculated_thetaB[0]
        theta[1] = calculated_thetaB[1]
else:
        theta[0]=calculated_thetaA[0]
        theta[1]=calculated_thetaA[1]

if ARM*ELBOW==1:
        theta[2]=calculated_thetaA[2]
else:
        theta[2]=calculated_thetaB[2]

theta[3]=calculated_thetaA[3]
theta[4]=calculated_thetaA[4]
theta[5]=calculated_thetaA[5]
```

2. Using half angle method:
In this method to solve equation of the type a*sin(x) + b*cos(x) = c
we let
u = tan (theta/2)
then sin (theta) = 2u / (1 + u^2)
and cos (theta) = (1 - u^2) / (1 + u^2)

Finally we get a quadratic in u and the solution gives us tan (theta/2).
Taking inverse gives us theta.

All Possible Solutions:

calculated_theta_halfA[0] = 180/pi*2*atan2(-px + sqrt(px*px+py*py-d[1]*d[1]),d[1]+py)
calculated_theta_halfB[0] = 180/pi*2*atan2(-px - sqrt(px*px+py*py-d[1]*d[1]),d[1]+py)

g214 = cos(theta[0])*px + sin(theta[0])*py
g224 = -pz
d_const = g214*g214 + g224*g224 -d[3]*d[3] -a[2]*a[2] -a[1]*a[1]
e_const_square= 4*a[1]*a[1]*a[2]*a[2] + 4*a[1]*a[1]*d[3]*d[3]

calculated_theta_halfA[2] = 180/pi*2*atan2(2*a[1]*d[3] + sqrt(e_const_square - d_const*d_const), d_const+2*a[1]*a[2])
calculated_theta_halfB[2] = 180/pi*2*atan2(2*a[1]*d[3] - sqrt(e_const_square - d_const*d_const), d_const+2*a[1]*a[2])

f=g214-a[1]*cos(theta[2])
pxx=-(d[3]*cos(theta[2]) -a[2]*sin(theta[2]))
pyy=d[3]*sin(theta[2]) +a[2]*cos(theta[2]) + a[1]
rad=g214
calculated_theta_halfA[1]=180/pi*2*(atan2(-pxx + sqrt(pxx*pxx+pyy*pyy-rad*rad) , rad + pyy))
calculated_theta_halfB[1]=180/pi*2*(atan2(-pxx - sqrt(pxx*pxx+pyy*pyy-rad*rad) , rad + pyy))

calculated_theta_halfA[3]= 180/pi*atan2(-sin(theta[0])*ax + cos(theta[0])*ay , cos(theta[1]+theta[2])*(cos(theta[0])*ax +sin(theta[0])*ay) - az*(sin(theta[1]+theta[2])))

calculated_theta_halfB[3]= 180 + 180/pi*atan2(-sin(theta[0])*ax + cos(theta[0])*ay , cos(theta[1]+theta[2])*(cos(theta[0])*ax +sin(theta[0])*ay) - az*(sin(theta[1]+theta[2])))

calculated_theta_halfA[4]=180/pi*atan2(
cos(theta[3])*(cos(theta[1]+theta[2])*(cos(theta[0])*ax+sin(theta[0])*ay)-
sin(theta[1]+theta[2])*az)+sin(theta[3])*(-sin(theta[0])*ax +cos(theta[0])*ay) ,
sin(theta[1]+theta[2])*(cos(theta[0])*ax + sin(theta[0])*ay) +az*cos(theta[1]+theta[2]))

calculated_theta_halfA[5]=180/pi*atan2(-
sin(theta[3])*(cos(theta[1]+theta[2])*(cos(theta[0])*nx+sin(theta[0])*ny) -
sin(theta[1]+theta[2])*nz) + cos(theta[3])*(-sin(theta[0])*nx + cos(theta[0])*ny) , -
sin(theta[3])*(cos(theta[1]+theta[2])*(cos(theta[0])*sx+sin(theta[0])*sy) -
sin(theta[1]+theta[2])*sz) + cos(theta[3])*(-sin(theta[0])*sx + cos(theta[0])*sy))

The actual solution is:

```
if ARM==1:
	theta[0]=calculated_theta_halfB[0]
	theta[1]=calculated_theta_halfB[1]
else:
	theta[0]=calculated_theta_halfA[0]
	theta[1]=calculated_theta_halfA[1]
if ARM*ELBOW==1:
	theta[2]=calculated_theta_halfB[2]
else:
	theta[2]=calculated_theta_halfA[2]
theta[3]=calculated_theta_halfA[3]
theta[4]=calculated_theta_halfA[4]
theta[5]=calculated_theta_halfA[5]
```

3. Using geometric method:

In this method the geometric structure of the robot is used to calculate the configuration.

p=[A_04[0][3], A_04[1][3], A_04[2][3]]
px=p[0]
py=p[1]
pz=p[2]

temp=sqrt(px*px+py*py-d[1]*d[1])

geometry_calculated_theta[0] = 180/pi*atan2((-ARM*py*temp-px*d[1])/(px*px+py*py) , (-
ARM*px*temp+py*d[1])/(px*px+py*py))

```
R=sqrt(px*px+py*py+pz*pz-d[1]*d[1])
r=sqrt(px*px+py*py-d[1]*d[1])
sin_alpha=-pz/R
cos_alpha=-ARM*r/R

cos_beta=(a[1]*a[1]+R*R-d[3]*d[3]-a[2]*a[2])/(2*a[1]*R)
sin_beta=sqrt(1-cos_beta*cos_beta)

sin_theta_2=sin_alpha*cos_beta+(ARM*ELBOW)*cos_alpha*sin_beta
cos_theta_2=cos_alpha*cos_beta-(ARM*ELBOW)*sin_alpha*sin_beta

geometry_calculated_theta[1]=180/pi*atan2(sin_theta_2, cos_theta_2)

R=sqrt(px*px+py*py+pz*pz-d[1]*d[1])
cos_phi = (a[1]*a[1] + d[3]*d[3] + a[2]*a[2] - R*R)/( 2*a[1]*sqrt(d[3]*d[3] + a[2]*a[2]) )
sin_phi = ARM*ELBOW*sqrt(1-cos_phi*cos_phi)

sin_beta= d[3] / (sqrt(d[3]*d[3] + a[2]*a[2]) )
cos_beta= fabs(a[2]) / (sqrt(d[3]*d[3] + a[2]*a[2]) )

sin_theta_3 = sin_phi*cos_beta - cos_phi*sin_beta
cos_theta_3 = cos_phi*cos_beta + sin_phi*sin_beta

geometry_calculated_theta[2] = 180/pi*atan2(sin_theta_3, cos_theta_3)## theta 4, theta 5
and theta 6

z3= [A_03[0][2], A_03[1][2], A_03[2][2] ]
cross=[z3[1]*approach[2] - approach[1]*z3[2]  , approach[0]*z3[2] - z3[0]*approach[2]  ,
z3[0]*approach[1] - z3[1]*approach[0] ]
cross_mod =  sqrt(cross[0]*cross[0] +cross[1]*cross[1] +cross[2]*cross[2])

if cross_mod==0:
	sigma = 0
else:
	cross= [cross[0]/cross_mod, cross[1]/cross_mod, cross[2]/cross_mod]
	s_prod = [s[0]*cross[0] +s[1]*cross[1] +s[2]*cross[2]]
	if (s_prod == 0):
		sigma=[n[0]*cross[0] +n[1]*cross[1] +n[2]*cross[2]]
	else:
		sigma = s_prod
```

```
if sigma>=0:
        M= WRIST
else:
        M= -WRIST

temp1= (M* (cos(pi/180*geometry_calculated_theta[0])* approach[1] –
sin(pi/180*geometry_calculated_theta[0])* approach[0] ))

geometry_calculated_theta[3] = 180/pi*atan2( temp1 ,
M*((cos(pi/180*geometry_calculated_theta[0])*cos(pi/180*geometry_calculated_theta[1]+pi
/180*geometry_calculated_theta[2])*approach[0]) +
(sin(pi/180*geometry_calculated_theta[0])*cos(pi/180*geometry_calculated_theta[1]+pi/180
*geometry_calculated_theta[2])*approach[1]) -
(sin(pi/180*geometry_calculated_theta[1]+pi/180*geometry_calculated_theta[2])*approach[
2])) )

C1=cos(pi/180*geometry_calculated_theta[0])
S1=sin(pi/180*geometry_calculated_theta[0])
C23=cos(pi/180*geometry_calculated_theta[1] + pi/180*geometry_calculated_theta[2])
S23=sin(pi/180*geometry_calculated_theta[1] + pi/180*geometry_calculated_theta[2])
C4=cos(pi/180*geometry_calculated_theta[3])
S4=sin(pi/180*geometry_calculated_theta[3])

geometry_calculated_theta[4] = 180/pi*atan2 ( (C1*C23*C4 - S1*S4)*approach[0] +
(S1*C23*C4 + C1*S4)*approach[1] - C4*S23*approach[2] , C1*S23*approach[0] +
S1*S23*approach[1] + C23*approach[2] )

geometry_calculated_theta[5] = 180/pi*atan2 ((-S1*C4 - C1*C23*S4)*n[0] + (C1*C4 -
S1*C23*S4)*n[1] + (S4*S23)*n[2] , (-S1*C4 - C1*C23*S4)*s[0] + (C1*C4 - S1*C23*S4)*s[1] +
(S4*S23)*s[2] )

if FLIP==1:
        geometry_calculated_theta[3] = geometry_calculated_theta[3]+pi
        geometry_calculated_theta[4] = - geometry_calculated_theta[4]
        geometry_calculated_theta[5] = geometry_calculated_theta[5]+pi
```

The geometry_calculated_theta is the actual theta