**Rotation Matrix**

syms theta

R=rot2(theta)

**rot2 provides orthonormal rotation matrix, whereas trot2 provides homogeneous transformation**

T1=transl2(1,2)\*trot2(30,'deg')

plotvol([0 5 0 5]);  
trplot2(T1, 'frame', '1', 'color', 'b')

T2 = transl2(2, 1)   
trplot2(T2, 'frame', '2', 'color', 'r');

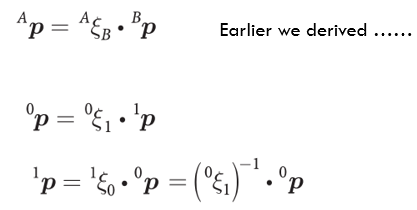
T3 = T1\*T2

trplot2(T3, 'frame', '3', 'color', 'g');

T4 = T2\*T1;

trplot2(T4, 'frame', '4', 'color', 'c');

P = [3 ; 2 ];   
plot\_point(P, 'label', 'P', 'solid', 'ko');



**To determine the coordinate of the point P wrt {1}**

P1 = inv(T1) \* [P; 1]

**Conversions from homogeneous to Euclidean space**

h2e( inv(T1) \* e2h(P) )

**Centers of Rotation**

plotvol([-5 4 -1 5]);  
T0 = eye(3,3);  
trplot2(T0, 'frame', '0');  
X = transl2(2, 3);  
trplot2(X, 'frame', 'X');

R = trot2(2);

trplot2(R\*X, 'framelabel', 'RX', 'color', 'r');  
trplot2(X\*R, 'framelabel', 'XR', 'color', 'r');

C = [1 2]';  
plot\_point(C, 'label', ' C', 'solid', 'ko')

% creating a transformation matrix for rotation about point C

% we first apply an origin shift, a translation from **C** to the origin of the reference frame, apply the rotation about that origin, and then apply the inverse origin shift, a translation from the reference frame origin back to **C.** A more better way to do this is through twists.

% We see that the frame {*RX*} has been rotated about the origin, while frame {*XR*} has been rotated about the origin of {*X*}.

RC = transl2(C) \* R \* transl2(-C)

trplot2(RC\*X, 'framelabel', 'XC', 'color', 'r');

**Twists**

**given any two frames we can find a rotational center that will rotate the first frame into the second**

**Rotational twist around a point C:**

**R stands for rotational. The twist object comprises of a 2-vector moment and a 1-vector rotation**

**Unit twist since magnitude of rotation = 1**

C = [1 2]';   
tw = Twist('R', C)

% to create transformation for a rotation about this unit twist by 2 radians

tw.T(2)

%to get center of rotation also known as pole

tw.pole’

**% For an arbitrary planar transform**

T = transl2(2, 3) \* trot2(0.5)

% we can compute twist as

tw = Twist(T)

tw.T

**Orthonormal Rotation Matrix in Toolbox**

syms theta

Rx = rotx(theta)

Ry = roty(theta)

Rz = rotz(theta)

Rx = rotx(pi/2)

trplot(Rx)

tranimate(Rx)

Rxy = rotx(pi/2) \* roty(pi/2)

tranimate(Rxy)

**Eulerian Rotation**

R = rotz(0.1) \* roty(0.2) \* rotz(0.3);

R = eul2r(0.1, 0.2, 0.3)

%rotation to euler

gamma = tr2eul(R)

%problem if theta is negative, different angles are returned but same rotation matrix

R = eul2r(0.1 , -0.2, 0.3)   
tr2eul(R)

% problematic since mapping from a rotation matrix to Euler angles is not unique and the Toolbox *always* returns a positive angle for θ

eul2r(ans)

**Roll Pitch Yaw Cardanian**

R = rpy2r(0.1, 0.2, 0.3)

% inverse

gamma= tr2rpy(R)

**oa2r - Convert orientation and approach vectors to rotation matrix**

a = [1 0 0]';  
o = [0 1 0]';  
R = oa2r(o, a)

**Rotation about an arbitrary vector**

R = rpy2r(0.1 , 0.2, 0.3); % roll-pitch-yaw to rotation matrix

% we can determine such an angle and vector by

[theta, v] = tr2angvec(R) %Convert rotation matrix to angle-vector form

% where theta is the angle of rotation and v is the vector around which the rotation occurs

**Quarternion**

%converts rotation matrix to a unit quaternion

q = UnitQuaternion( rpy2tr(0.1, 0.2, 0.3) )

%convert quaternion to rotation matrix

q.R

**Pose in 3D**

**We can think of this expression as representing a walk along the *x*-axis for 1 unit, then a rotation by 90° about the *x*-axis and then a walk of 1 unit along the new *y*-axis which was the previous *z*-axis.**

T = transl(1, 0, 0) \* trotx(pi/2) \* transl(0, 1, 0)

trplot(T)

t2r(T)

transl(T)’

**Forward Kinematics**

**% ETS stands for elementary transform sequence in 2D**

Import ETS2.\*

a1 = 1

E = Rz(‘q1’) \* Tx(a1)

E.fkine( 30, ‘deg’)

**Forward Kinematics - Single Joint**

a1=1;

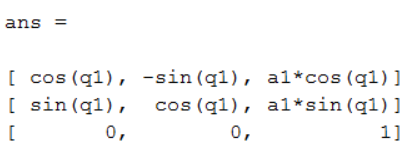
q1=0.2;

trchain2 (‘R(q1) Tx(a1)’, q1)

%Generic

syms q1 a1

trchain2 (‘R(q1) Tx(a1)’, q1)



A number with numbers and letters

Description automatically generated with medium confidence

**Forward Kinematics – 2 Joint**

a1=1;

a2=2;

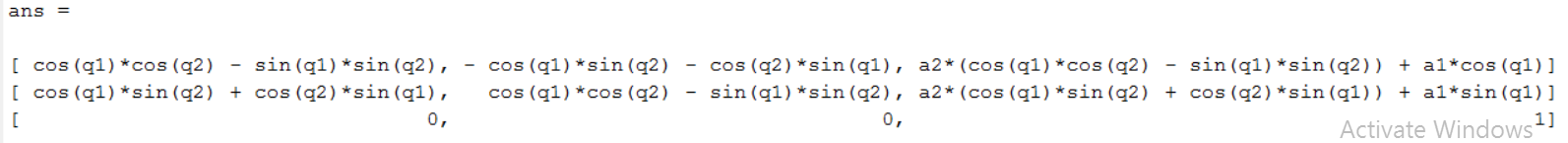
q1=0.2;

q2=0.3;

trchain2(‘R(q1) Tx(a1) R(q2) Tx(a2)’, [q1 q2])

syms q1 q2 a1 a2

trchain2(‘R(q1) Tx(a1) R(q2) Tx(a2)’, [q1 q2])



A number and a number

Description automatically generated with medium confidence

**Forward Kinematics – 3 Joint**

syms q1 q2 q3 a1 a2 a3

trchain2(‘R(q1) Tx(a1) R(q2) Tx(a2) R(q3) Tx(a3)’, [q1 q2 q3])

**Forward Kinematics – 3D**

syms q1 q2 q3 q4 a1 a2 a3 a4

trchain2(‘Rz(q1) Tz(a1) Ry(q2) Tz(a2) Ry(q3) Tz(a3) Ry(q4) Tz(a4)’, [q1 q2 q3 q4])

**DH Table in Toolbox**

dh = [ 0 0 1 0;

0 0 1 0 ]

% each row corresponds to one row of DH table i.e theta, d, a and alpha parameters of each joint

r = SerialLink(dh)

r.tool

r.gravity

r.base

dh = [ 0 0 1 0;

0 0 1 0 ]

% each row corresponds to one row of DH table i.e. theta, d, a and alpha parameters of each joint

r = SerialLink(dh)

r.tool

r.gravity

r.base

r.plot([0.2 0.3])

r.teach

r.fkine([0.2 0.3])

%returns the pose of end-effector

**PUMA in Toolbox**

mdl\_puma560

p560

p560.plot(qz)

p560.plot(qr)

p560.teach

p560.fkine ([0.1 0.2 0.3 0.4 0.5 0.6])