Robotics Library

Minghe Jiang

RotInv:

```
function [ RotInv ] = RotInv( RotationMatrix )
if RotationMatrix*transpose(RotationMatrix) == eye(3) & det(RotationMatrix) == 1
    RotInv = transpose(RotationMatrix);
else
    RotInv = 'Error';
end
%Takes an Rotaition matrix which is in SO(3) and returns its inverse.
```

VecToso3:

```
function [ bracket_w ] = VecToso3( w )
    bracket_w = [0,-w(3),w(2);w(3),0,-w(1);-w(2),w(1),0];
end
%Takes a 3-vector (representing an angular velocity) and returns the 3 X 3
%skew-symmetric matrix version, an element of so(3).
```

so3ToVec:

```
function [ w ] = so3ToVec( bracket_w )
w = [bracket_w(3,2);bracket_w(1,3);bracket_w(2,1)];
end
%Takes a 3 x 3 skew-symmetric matrix (an element of so(3)) and returns the
%corresponding 3-vector
```

AxisAng3:

```
function [theta, w] = AxisAng3(r)
theta = sqrt(r(1)^2+r(2)^2+r(3)^2);
w = [r(1)/theta;r(2)/theta;r(3)/theta];
end
%Takes a 3-vector of exponential coordinates for rotation and returns the unit
%rotation axis w and the rotation amount theta
```

MatrixExp3:

```
function [ RotationMatrix ] = MatrixExp3( r )
[theta, w] = AxisAng3(r);
bracket_w = VecToso3( w );
RotationMatrix = eye(3)+sin(theta)*bracket_w+(1-cos(theta))*(bracket_w)^2;
end
%Takes a 3-vector of exponential coordinates r and returns R that is
%achieved by rotating about w by theta from an initial orientation R = I.
```

MatrixLog3:

```
function [ r ] = MatrixLog3( RotationMatrix )
if RotationMatrix == eye(3)
    r = 'theta = 0, w is undefined';
elseif trace(RotationMatrix) == -1
    r = 'theta = pi, w has three feasible solutions';
else
    theta = acos((trace(RotationMatrix)-1)/2);
    bracket_w = (RotationMatrix-transpose(RotationMatrix))/(2*sin(theta));
    w = so3ToVec( bracket_w );
    r = w*theta;
end
%Takes an R and returns the corresponding 3-vector of exponential coordinates r
```

RpToTrans:

```
function [ T ] = RpToTrans( RotationMatrix, p )
T = [RotationMatrix,p;0,0,0,1];
end
%Takes an R and a position p and returns the corresponding homogeneous
%transformation matrix T
```

TransToRp:

```
function [ RotationMatrix, p ] = TransToRp( T )
RotationMatrix = T(1:3,1:3);
p = T(1:3,4);
end
%Takes a T and returns the corresponding R and p
```

TransInv:

```
function [ inversed_T ] = TransInv( T )
[ RotationMatrix, p ] = TransToRp( T );
inversed_T = [transpose(RotationMatrix),-transpose(RotationMatrix)*p;0,0,0,1];
end
%Takes an element of SE(3) and returns the inverse.
```

VecTose3:

```
function [ bracket_S ] = VecTose3( S )
w = S(1:3);
v = S(4:6);
bracket_w = VecToso3( w );
bracket_S = [bracket_w,v;0,0,0,0];
end
%Takes a 6-vector (representing a spatial velocity) and returns the corresponding 4x4
matrix, an element of se(3)
```

se3ToVec:

```
function [ S ] = se3ToVec( bracket_S )
bracket_w =  bracket_S(1:3,1:3);
v =  bracket_S(1:3,4);
w = so3ToVec( bracket_w );
S = [w;v];
end
%Takes an element of se(3) and returns the corresponding spatial velocity
%as a 6-vector
```

Adjoint:

```
function [ AdT ] = Adjoint( T )
[ RotationMatrix, p ] = TransToRp( T );
[ bracket_p ] = VecToso3( p );
AdT = [RotationMatrix,zeros(3);bracket_p*RotationMatrix,RotationMatrix];
end
%Takes a T and returns the 6 x 6 adjoint representation [AdT].
```

ScrewToAxis:

```
function [ S ] = ScrewToAxis(q, s, h)
v = cross(s,-q)+h*s;
w = s;
S = [w;v];
end
%Takes a point q, a unit axis s and a screw pitch h and returns the
%corresponding screw axis S = (w,v), a normalized unit spatial velocity.
%The frame of this spatial velocity is the same frame that q and s are
%given in. This function only handles finite values of h with norm(w) = 1
```

AxisAng6:

```
function [ S, theta, w, v ] = AxisAng6( V )
w = V(1:3);
v = V(4:6);
if w == [0;0;0]
    S = [0;0;0;v/norm(v)];
    [theta] = norm(w);
else
    S = [w/norm(w);v/norm(w)];
    [theta] = AxisAng3(w);
end
%Takes a 6-vector of exponential coordinates for rigid-body motion,
%S*theta, and returns the screw axis S and the distance traveled
%along/about the screw axis theta
```

MatrixExp6:

```
function [ expT ] = MatrixExp6( V )
[ S, theta, w, v ] = AxisAng6( V );
if w == [0;0;0]
    expT = [eye(3),v;0,0,0,1];
else
    [ bracket_w ] = VecToso3( w/theta );
    [ RotationMatrix ] = MatrixExp3( w );
    expT =
[RotationMatrix,(eye(3)*theta+(1-cos(theta))*bracket_w+(theta-sin(theta))*bracket
_w^2)*(v/theta);0,0,0,1];
end
%Takes a 6-vector of exponential coordinates S*theta and returns T that is achieved
by traveling along/about the screw axis S a distance theta from an initial configuration
T = I.
```

MatrixLog6:

```
function [ V ] = MatrixLog6( T )
[ RotationMatrix, p ] = TransToRp( T );
if RotationMatrix == eye(3)
    w = [0;0;0];
    v = p/norm(p);
    theta = norm(p);
    S = [w;v];
    V = S*theta;
elseif trace(RotationMatrix) == -1
    theta = pi;
    bracket_w = log(RotationMatrix);
    G_inverse =
(1/theta)*eye(3)-0.5*bracket_w+((1/theta)-0.5*cot(theta/2))*(bracket_w)^2;
    v = G_inverse*p;
    [ w ] = so3ToVec( bracket_w );
    S = [w;v];
    V = S*theta;
else
    theta = acos((trace(RotationMatrix)-1)/2) ;
    bracket_w = 0.5*(RotationMatrix-transpose(RotationMatrix))/(sin(theta));
    [ w ] = so3ToVec( bracket_w );
    G_inverse =
(1/theta)*eye(3)-0.5*bracket_w+((1/theta)-0.5*cot(theta/2))*(bracket_w)^2;
    v = G_inverse*p;
    S = [w; v];
    V = S*theta;
end
```

```
%Takes a T and returns the corresponding 6-vector of exponential
%coordinates S*theta
```

FKinFixed:

```
function [ Ts ] = FKinFixed( M, Si, thetai )
joints = size(thetai, 2);
multi = eye(4);
for i = 1:joints
    V = Si(1:6,i)*thetai(i);
    [ expT ] = MatrixExp6( V );
    multi = multi* expT;
end
Ts = multi*M;
end
%Takes M representing the position and orientation of the end-effector
%frame when the manipulator is at its home position (theta_i = 0 for all n
%joints of the robot); a list of screw axes Si for the n joints of the
%robot, expressed in a fixed world frame; and a list of joint coordinates
%theta_i; and returns T representing the end-effector frame when the joints
%are at the angles specified.
```

FKinBody:

```
function [ Tb ] = FKinBody( M, Bi, thetai )
joints = size(thetai, 2);
multi = eye(4);
for i = 1:joints
    V = Bi(1:6,i)*thetai(i);
    [ expT ] = MatrixExp6( V );
    multi = multi* expT;
end
Tb = M*multi;
end
%now the screw axes are expressed in the end-effector frame as Bi
```

FixedJacobian:

```
function [ Js ] = FixedJacobian( S,theta )
joints = size(S,2);
Js=[];
for i=1:joints
bracket_Si=VecTose3(S(:,i));
k=i-1;
if i==1
Ji=se3ToVec(bracket_Si);
else
```

```
Ji_=bracket_Si;
for j=1:k
Sj=S(:,k-j+1);
thetaj=theta(k-j+1);
Vj = Sj*thetaj;
Ji_=MatrixExp6(Vj)*Ji_*TransInv(MatrixExp6(Vj));
end
Ji=se3ToVec(Ji_);
end
Js=[Js,Ji];
end
end
%Takes a set of joint angles theta and screw axes Si for the robot joints
%expressed in the fixed space frame, and returns the space Jacobian Js(theta)
```

BodyJacobian:

```
function [ Jb ] = BodyJacobian( B,theta )
joints = size(B,2);
Jb=[];
for i=1:joints
    bracket_Bi=VecTose3(B(:,joints-i+1));
    k=i-1;
    if i==1
        Ji=se3ToVec(bracket_Bi);
    else
        Ji_=bracket_Bi;
        for j=joints-i+2:joints
            Bj=B(:,j);
            thetaj=theta(j);
            Vj = Bj*thetaj;
            Ji_=TransInv(MatrixExp6(Vj))*Ji_*MatrixExp6(Vj);
        end
         Ji=se3ToVec(Ji_);
    end
    Jb=[Ji,Jb];
end
end
%Takes a set of joint angles theta and screw axes Bi for the robot joints
%expressed in the end-effector body frame, and returns the body Jacobian
%Jb(theta)
```

IKinBody:

```
function [ matrix_jointangles ] = IKinBody( B, M, Tsd, theta0, ew, ev )
thetalist=[theta0];
i = 0;
Vb = MatrixLog6(TransInv(FKinBody(M, B, theta0))*Tsd);
wb = Vb(1:3);
vb = Vb(4:6);
while (norm(wb) > ew||norm(vb) > ev)&&(i < 100)
    if i == 0
        Jb = BodyJacobian(B,theta0);
    else
        Jb = BodyJacobian(B,thetaiplus1);
    end
    m = size(Jb,1);
    n = size(Jb,2);
    if n > m
        Jb_ = transpose(Jb)*pinv(Jb*transpose(Jb));
    else
        Jb_= pinv(transpose(Jb)*Jb)*transpose(Jb);
    end
    if i == 0
        thetaiplus1 = theta0+transpose(Jb_*Vb);
    else
        thetaiplus1 = thetaiplus1+transpose(Jb_*Vb);
    end
    thetalist = [thetalist;thetaiplus1];
    i = i + 1;
    Vb = MatrixLog6(TransInv(FKinBody(M, B, thetaiplus1))*Tsd);
    wb = Vb(1:3);
    vb = Vb(4:6);
end
thetalist
csvwrite('bodyq.csv',thetalist);
end

%Takes a set of screw axes Bi for the robot joints expressed in the
%end-effector body frame, the end-effector zero configuration M, the
%desired end-effector configuration Tsd, an initial guess theta0 that is
%"close" to satisfying T(theta0) = Tsd, and small scalar values ε_w > 0 and
%ε_v > 0 controlling how close the final solution thetak must be to the
%desired answer.There is a maximum number of iterations 'maxiterates'
%before stopping. Returns a matrix of joint angles.
```

IKinFixed:

```
function [ matrix_jointangles ] = IKinFixed( S, M, Tsd, theta0, ew, ev )
thetalist=[theta0];
i = 0;
Tsb = FKinFixed(M, S, theta0);
Vb = MatrixLog6(TransInv(Tsb)*Tsd);
Vs = Adjoint(Tsb)*Vb;
ws = Vs(1:3);
vs = Vs(4:6);
while (norm(ws) > ew||norm(vs) > ev)&&(i < 100)
    if i == 0
        Js = FixedJacobian(S,theta0);
    else
        Js = FixedJacobian(S,thetaiplus1);

    end
    m = size(Js,1);
    n = size(Js,2);
    Js_ = pinv(Js);
    if i == 0
        thetaiplus1 = theta0+transpose(Js_*Vs);
    else
        thetaiplus1 = thetaiplus1+transpose(Js_*Vs);
    end
    thetalist = [thetalist;thetaiplus1];
    i = i + 1;
    Tsb = FKinFixed(M, S, thetaiplus1);
    Vb = MatrixLog6(TransInv(Tsb)*Tsd);
    Vs = Adjoint(Tsb)*Vb;
    ws = Vs(1:3);
    vs = Vs(4:6);
end
thetalist
csvwrite('fixedq.csv',thetalist);
end

%Takes a set of screw axes Si for the robot joints expressed in the
%space frame, the end-effector zero configuration M, the
%desired end-effector configuration Tsd, an initial guess theta0 that is
%"close" to satisfying T(theta0) = Tsd, and small scalar values ε_w > 0 and
%ε_v > 0 controlling how close the final solution thetak must be to the
%desired answer.There is a maximum number of iterations 'maxiterates'
%before stopping. Returns a matrix of joint angles.
```