```
%==================================== SE3
 ====================================
%
%  class SE3
%
%  g = SE3(d, theta)
%
%
%  A Matlab class implementation of SE(3) [Special Euclidean 3-space].
%  Allows for the operations written down as math equations to be
%  reproduced in Matlab as code.  At least that's the idea.  It's
 about
%  as close as one can get to the math.
%
%==================================== SE3
 ====================================
classdef SE3 < handle


properties (Access = protected)
  M;             % Internal implementation is homogeneous.
end

%
%========================= Public Member Methods
 =========================
%

methods

  %-------------------------------- SE3
 --------------------------------
  %
  %  Constructor for the class.  Expects translation vector and
 rotation
  %  angle.  If both missing, then initialize as identity.
  %
  function g = SE3(d, R)

  if (nargin == 0)
    g.M = eye(4);
  else
    g.M = [R, d; 0 0 0 1];
  end

  end

  %
  %------------------------------ display
 ------------------------------
  %
  %  Function used by Matlab to "print" or display the object.
```

```
% Just outputs it in homogeneous form.
%
function display(g)

disp(g.M);

end

%------------------------------- plot
-------------------------------
%
% Plots the coordinate frame associated to g.  The figure is
cleared,
% so this will clear any existing graphic in the figure.  To plot
on
% top of an existing figure, set hold to on.  The label is the name
% of label given to the frame (if given is it writen out).  The
% linecolor is a valid plot linespec character.  Finally sc is the
% specification of the scale for plotting.  It will rescale the
% line segments associated with the frame axes and also with the
location
% of the label, if there is a label.
%
% Inputs:
%   g  - The SE2 coordinate frame to plot.
%   label - The label to assign the frame.
%   linecolor  - The line color to use for plotting.  (See `help
plot`)
%   sc  - scale to plot things at.
%   a 2x1 vector, first element is length of axes.
%     second element is a scalar indicating roughly how far
%     from the origin the label should be placed.
%
% Output:
%   The coordinate frame, and possibly a label, is plotted.
%
function plot(g, flabel, lcol, sc)

if ( (nargin < 2) )
  flabel = '';
end

if ( (nargin < 3) || isempty(lcol) )
  lcol = 'b';
end

if ( (nargin < 4) || isempty(sc) )
  sc = [1.0 0.5];
elseif (size(sc,2) == 1)
  sc = [sc 2];
end

d = getTranslation(g);
R = getRotation(g);
```

```
ex = R*[sc(1);0;0];      % get rotated x-axis.
ey = R*[0;sc(1);0];      % get rotated y-axis.
ez = R*[0;0;sc(1)];      % get rotated z-axis.

isheld = ishold;

pts = [d , d+ex];
plot3(pts(1,:), pts(2,:), pts(3,:),lcol);          % x-axis
hold on;
  pts = [d , d+ey];
  plot3(pts(1,:), pts(2,:), pts(3,:),lcol);      % y-axis
  pts = [d , d+ez];
  plot3(pts(1,:), pts(2,:), pts(3,:),lcol);      % z-axis

  plot3(d(1), d(2), d(3), [lcol 'o'],'MarkerSize',7); % origin

if (~isempty(flabel))
  pts = d - (sc(2)/sc(1))*(ex+ey+ez);
  text(pts(1), pts(2), pts(3),flabel);
end

if (~isheld)
 hold off;
end
end

%------------------------------- inv ------------------------------
%
%  Returns the inverse of the element g.  Can invoke in two ways:
%
%    g.inv();
%
%  or
%
%    inv(g);
%
%
function invg = inv(g)

invg = SE3();       % Create the return element as identity element.
invM = (g.M)^-1;        % Compute inverse of matrix.
invg.M = invM;      % Set matrix of newly created element to
inverse.

end

%------------------------------ times ------------------------------
%
%  This function is the operator overload that implements the left
%  action of g on the point p.
%
%  Can be invoked in the following equivalent ways:
%
```

```
%  >> p2 = g .* p;
%
%  >> p2 = times(g, p);
%
%  >> p2 = g.times(p);
%
function p2 = times(g, el)

p2 = g.leftact(el);

end

%----------------------------- mtimes -----------------------------
%
%  Computes and returns the product of g1 with g2.
%
%  Can be invoked in the following equivalent ways:
%
%  >> g3 = g1 * g2;
%
%  >> g3 = g1.mtimes(g2);
%
%  >> g3 = mtimes(g1, g2);
%
function g3 = mtimes(g1, g2)

g3 = SE3();           % Initialize return element as identity.
g3.M = g1.M * g2.M;    % Set the return element matrix to product.

end

%---------------------------- leftact -----------------------------
%
%  g.leftact(p)     --> same as g . p
%
%                with p a 2x1 specifying point coordinates.
%
%  g.leftact(v)     --> same as g . v
%
%                with v a 3x1 specifying a velocity.
%                This applies to pure translational velocities in
%                homogeneous form, or to SE3 velocities in vector
forn.
%
%  This function takes a change of coordinates and a point/velocity,
%  and returns the transformation of that point/velocity under the
%  change of coordinates.
%
%  Alternatively, one can think of the change of coordinates as a
%  transformation of the point to somewhere else, e.g., a
displacement
%  of the point.  It all depends on one's perspective of the
%  operation/situation.
%
```

```
function x2 = leftact(g, x)

if ( (size(x,1) == 3) && (size(x,2) == 1) )
  % two vector, this is product with a point.
  x = [x;1];
  x2 = g.M * x;
  x2(4) = [];
elseif ( (size(x,1) == 4) && (size(x,2) == 1) )
  % three vector, this is homogeneous representation.
  % fill out with proper product.
  % should return a homogenous point or vector.
  x2 = g.M * x;
end

end

%---------------------------- adjoint ----------------------------
%
%  h.adjoint(g)   --> same as Adjoint(h) . g
%
%  h.adjoint(xi)  --> same as Adjoint(h) . xi
%
%  Computes and returns the adjoint of g.  The adjoint is defined to
%  operate as:
%
%    Ad_h (g) = h * g2 * inverse(h)
%
function z = adjoint(g, x)

if (isa(x,'SE3'))
  % if x is a Lie group, ...
  z = x.M * g.M * inv(x.M);
elseif ( (size(x,1) == 6) && (size(x,2) == 1) )
  % if x is vector form of Lie algebra, ...
  % turn x/y/z/roll/pitch/yaw into a homogeneous matrix
  R = EulerXYZtoR(x(4), x(5), x(6));
  A = [R; 0 0 0];
  A = [R [x(4); x(5); x(6); 1]];
  z = A * g.M * inv(A);
elseif ( (size(x,1) == 4) && (size(x,2) == 4) )
  % if x is a homogeneous matrix form of Lie algebra, ...
  z = x * g.M * inv(x);
end

end

%--------------------------------- log
--------------------------------
%
%  Compute the log of a Lie group element.  Returns the vector form.
%
function xi = log(g, tau)

  if ((nargin < 2) || (isempty(tau)))    % No tau, assume unity.
```

```
            tau = 1;
        end

        R = getRotationMatrix(g);
        w = [R(3,2); R(1,3); R(2,1)];
        T = getTranslation(g);

        wmag = acos((trace(R) - 1)/2);
        what = wmag/(2*sin(wmag*tau)) * (R - transpose(R));
        v = wmag^2 * inv((eye(3) - R)*what + w*transpose(w)*tau)*T;

        xi = [v; w];

      end

      %
      %-------------------------- getTranslation
     -------------------------
      %
      %  Get the translation vector of the frame/object.
      %
      %
      function T = getTranslation(g)

      T = [g.M(1,4); g.M(2,4); g.M(3,4)];

      end

      %------------------------ getRotationMatrix
     ------------------------
      %
      %  Get the rotation or orientation of the frame/object.
      %
      %
      function R = getRotationMatrix(g)

      R = g.M;
      R(4,:) = [];
      R(:,4) = [];

      end

    end

    %
    %====================== Static (Helper) Methods
     ======================
    %
    %  These methods are helper functions for the class.  Typically they
    %  do not involve actual SE(3) Lie group elements, but are functions
    %  that are related to the SE3() Lie group.  Even though they do not
    %  take elements of the class, they still may return elements of the
    %  class.
    %
```

```
%  They get run by invoking as follows:
%
%  output = SE3.funcName(input);
%

methods(Static)

  %-------------------------------- hat
 --------------------------------
  %
  %  Hat a vector element representation of the Lie algebra se(3).
  %
  function xiHat = hat(xiVec)

      w1 = xiVec(4);
      w2 = xiVec(5);
      w3 = xiVec(6);
      xiHat = [0 -w3 w2;
            w3 0 -w1;
            -w2 w1 0];

  end

  %------------------------------- unhat
 -------------------------------
  %
  %  Unhat a homogeneous matrix element of the Lie algebra se(3).
  %
  function xiVec = unhat(xiHat)

      w1 = xiHat(3,2);
      w2 = xiHat(1,3);
      w3 = xiHat(2,1);
      xiVec = [w1; w2; w3];

  end

  %-------------------------------- exp
 --------------------------------
  %
  %  Takes in an element of the Lie algebra and compute the
 exponential
  %  of it.  Should return an actual SE3 element.
  %
  function expXi = exp(xi, tau)

  if ((nargin < 2) || (isempty(tau)))
    tau = 1;
  end

  expXi = SE3();

  v = xi(1:3);
  w = xi(4:6);
```

```
  wmag = sqrt(w(1)^2 + w(2)^2 + w(3)^2);
  what = SE3.hat(xi);

  Rexp = eye(3) + (what/wmag)*sin(wmag*tau) + (what^2/wmag^2)*(1-
cos(wmag*tau));
  Texp = (eye(3) - Rexp) * ((what*v)/wmag^2) + w*transpose(w)/wmag^2 *
 v*tau;

  expXi.M = [Rexp Texp; 0 0 0 1];

  end


  %-------------------------------- RotX
 ------------------------------
  %
  %  Takes an angle and generates rotation matrix about that angle,
  %  with respect to x-axis.
  %
  function Rmat = RotX(theta)

  Rmat = [1         0           0;
          0         cos(theta)  -sin(theta);
          0         sin(theta)  cos(theta) ];

  end

  %-------------------------------- RotY
 ------------------------------
  %
  %  Takes an angle and generates rotation matrix about that angle.
  %  with respect to y-axis.
  %
  function Rmat = RotY(theta)

  Rmat = [cos(theta)    0      sin(theta);
          0             1      0;
          -sin(theta)   0      cos(theta)];
  end

  %-------------------------------- RotZ
 ------------------------------
  %
  %  Takes an angle and generates rotation matrix about that angle.
  %  with respect to z-axis.
  %
  function Rmat = RotZ(theta)

  Rmat = [cos(theta)  -sin(theta)   0;
          sin(theta)  cos(theta)    0;
          0           0             1];

  end
```

```
 %---------------------------- EulerXYZtoR
----------------------------
 %
 %  Generates a rotation matrix given the x-y-z Euler angle
convention.
 %
 function Rmat = EulerXYZtoR(thX, thY, thZ)

 % Should be RotX * RotY * RotZ
 Rmat = RotX(thX) * RotY(thY) * RotZ(thZ);

 end

 %---------------------------- RtoEulerXYZ
----------------------------
 %
 %  Generates x-y-z Euler angle convention given a rotation matrix.
 %
 function [x, y, z] = RtoEulerXYZ(Rmat)

 % Should be RotX * RotY * RotZ
 % TO BE DONE LATER, AS PART OF INVERSE KINEMATICS.

 end


end


end
```

```
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Published with MATLAB® R2017a