```matlab
%================================== SE2
 ==================================
%
%  class SE2
%
%  g = SE2(d, theta)
%
%
%  A Matlab class implementation of SE(2) [Special Euclidean 2-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.
%
%================================== SE2
 ==================================
classdef SE2 < handle


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

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

    methods

        %------------------------------- SE2
 --------------------------------
        %
        %  Constructor for the class.  Expects translation vector and
 rotation
        %  angle.  If both missing, then initialize as identity.
        %
        function g = SE2(d, theta)

            if (nargin == 0)
                g.M = eye(3);
            else
                g.M = [cos(theta), -sin(theta), d(1);  ...
                    sin(theta),  cos(theta), d(2);  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
-------------------------------
        %
        %  plot(label, linecolor)
        %
        %  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.
        %
        %  Optional Inputs:
        %    label      - The label to assign the frame. [default:
blank]
        %    linecolor  - The line color to use for plotting.  (See
`help plot`)
        %                  [default: 'b'  <- blue]
        %
        %  Output:
        %    The coordinate frame, and possibly a label,  is plotted.
        %
        function plot(g, flabel, lcol)

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

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

            o = g.M([1 2],3);         % Get the translation part for
origin.

            x = g.M(1:2,1:2)*[2;0];   % Rotate axes into plot frame.
            y = g.M(1:2,1:2)*[0;2];

            isheld = ishold;          % Record whether on hold or not.

            plot(o(1)+[0 x(1)],o(2) + [0 x(2)],lcol);
            hold on;
            plot(o(1)+[0 y(1)],o(2) + [0 y(2)],lcol);
            plot(o(1), o(2), [lcol 'o'],'MarkerSize',7);

            if (~isempty(flabel))
```

```
                text(o(1) - (x(1)+y(1))/6, o(2) - (x(2)+y(2))/6,
flabel);
            end

            if (~isheld)
                hold off;
            end

            axis equal;

        end

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

            invg = SE2();       % 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)

            % Initialize return element as identity.
            if (isa(g2,'SE2'))     % Set the return element matrix to
 product.
                g3 = SE2();
                g3.M = g1.M * g2.M;
            else
                g3 = eye(3);
                g3 = g1.M * g2;
            end

        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 SE2 velocities in
 vector form.
        %
        %  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) == 2) && (size(x,2) == 1) )
                % two vector, this is product with a point.
                x = [x;1];
                x2 = g.M * x;
            elseif ( (size(x,1) == 3) && (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;
            elseif ( (size(x,1) == 3) && (size(x,2) == 3) )
                % homogeneous matrix
                x2 = g.M * x;
            elseif (isa(x, 'SE2'))
                x2 = g.M * x.M;
            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,'SE2') && isa(g,'SE2'))
                % if x and g are Lie group (SE2)
                z = x.M * g.M * inv(x.M);
            elseif ( (size(x,1) == 3) && (size(x,2) == 1) &&
isa(g, 'SE2'))
                % if x is vector form of Lie algebra and g is SE2
                x = [ R(x(3)) [x(1); x(2)]; 0 0 1 ];
                z = x * g.M * inv(x);
            elseif ( (size(x,1) == 3) && (size(x,2) == 3) &&
isa(g, 'SE2'))
                % if x is a homogeneous matrix and g is SE2
                z = x * g.M * inv(x);
            elseif ( (size(g,1) == 3) && (size(g,2) == 1) &&
isa(x, 'SE2'))
                % if g is a velocity vector and x is SE2
                ghat = SE2.hat(g);
```

```
                adj = x.M * ghat * inv(x.M);
                z = [adj(1,3); adj(2,3); adj(2,1)];
            elseif ( (size(g,1) == 3) && (size(g,2) == 3) &&
isa(x, 'SE2'))
                % if g is a hatted velocity vector and x is SE2
                z = x.M * g * inv(x.M);
            end

        end

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

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

        end

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

            R = [g.M(1,1) g.M(1,2); g.M(2,1) g.M(2,2)];

        end

        %------------------------ getRotationAngle
--------------------------
        %
        %  Get the rotation or orientation of the frame/object.
        %
        %
        function theta = getTheta(g)

            theta = acos(g.M(1,1));
            if (asin(g.M(2,1)) ~= theta)
                theta = theta + pi;
            end

        end


    end

    methods(Static)
```

```
%------------------------- hat -------------------------
%
% Perform the hat operation with a vector form of se(2).
% Output is the homogeneous matrix form of se(2).
%
%
function xiHat = hat(xiVec)
    J = [0 -1; 1 0];
    if (isa(xiVec, 'SE2'))
        %if xiVec is SE2
        xiV = [xiVec.M(1,3); xiVec.M(2,3); xiVec.M(2,1)];
        xiHat = [J*xiV(3) [xiV(1); xiV(2)]; 0 0 0];
    elseif ((size(xiVec,1) == 3) && (size(xiVec,2) == 3))
        %if xiVec is a homogeneous matrix
        xiHat = [J*xiVec(2,1) [xiVec(1,3); xiVec(2,3)]; 0 0
0];

    elseif ((size(xiVec,1) == 3) && (size(xiVec,2) == 1))
        %if xiVec is a velocity vector
        xiHat = [J*xiVec(3) [xiVec(1); xiVec(2)]; 0 0 0];
    end
end


%------------------------ unhat ------------------------
%
% Perform the unhat operation with a matrix form of se(2).
% Output is the vector form of se(2).
%
%
function xiVec = unhat(xiHat)
    if (isa(xiHat, 'SE2'))
        x = xiHat.M(1,3);
        y = xiHat.M(2,3);
        w = xiHat.M(2,1);
        xiVec = [x; y; w];
    else
        x = xiHat(1,3);
        y = xiHat(2,3);
        w = xiHat(2,1);
        xiVec = [x; y; w];
    end
end


%------------------------ exp ------------------------
%
%  Takes in an element of the Lie algebra and compute the
%  exponential of it.  Should return an actual SE2 element.
%
%
function expXi = exp(xi, tau)

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

```
            end

            expXi = SE2();

            J = [0, 1; -1, 0];
            w = xi(3);
            what = -J*w;
            v = [xi(1); xi(2)];

            %Rodrigues' formula
            Rexp = eye(2) + (what/w)*sin(w*tau) + (what^2/w^2)*(1-
cos(w*tau));
            Texp = (-1/w) * (eye(2)-Rexp) * J * v;

            %exception if omega = 0 (handle divide by 0)
            if(w == 0) %handle divide by 0 if omega is 0
                Rexp = eye(2);
                Texp = v * tau;
            end

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

        end

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

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

            J = [0 1; -1 0];

            R = getRotationMatrix(g);
            T = getTranslation(g);
            w = atan2(R(2,1), R(1,1));
            v = w * J * inv(eye(2) - R) * T;

            xi = [v; w];
        end
    end


end

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

Published with MATLAB® R2017a