ME 397: Algorithm for Sensor-Based Robotics. Spring 2022
THA 1, Programming Assignment

Bryant Zhou / Jonathan Salfity

Below is a description of our functions, test cases, and output we used to answer in THA1 PA.
Links to functions:

[rotation_2_axis_angle]

[rotation_2_quaternion]

[rotation_2_ZYZ]

[rotation_2_rpy]

[axis_angle_2_rotation]

[quaternion_2_rotation]

[Screw Axis]

The input params, output values, and references for each function are within the function comments.

For most test cases, we compared the output values of our functions with the builtin Matlab functions. We picked specific values for the input and output values that reflect corner cases and singularities.

To account for numerical precision, we used a global epsilon value:

```
function eps = getGlobaleps
   eps = 1e-3;
return
```

For all of our functions that input a Rotation matrix, we wrote an `RinSO3` function to check the input rotation matrix is in SO(3). We used an epsilon value for tolerance, as there is precision error in computation. In most cases, if the Rotation matrix is not in SO(3), the functions calling `RinSO` will return NaN for their function output values.

```
function output = RinSO3(R)
%RINSO3 Check R is in SO(3)
   % 3 checks
   % norm(R) = 1 (at least by some epsilon)
   % R * R' = I  (at least by some epsilon)
   % det(R) = 1  (at least by some epsilon)
   output = (norm(R) - 1)          < getGlobaleps && ...
            norm(R*R' - eye(3))    < getGlobaleps && ...
            (det(R) - 1)           < getGlobaleps ;

end
```

rotation_2_axis_angle

This function takes in a rotation matrix and outputs the corresponding axis and angle of rotation. There are a few checks to perform: The first singularity case is checked if R = I, because there is no rotation in this case.The second case checks trace(R) == -1, as it would be a rotation by pi. Finally, if those checks pass, the axis and angle are returned.

```
% PA 1a
function [theta, omega] = rotation_2_axis_angle(R)
   % param: R (3x3 rotation matrix)
   % return: omega, theta (axis-angle representation)
   % reference: ASBR notes, W2L2, slide 9. MR Section 3.2

   % check R is in SO(3)
   if ~RinSO3(R)
       theta = nan;
       omega = nan;
       return
   end

   if R == eye(3)
       theta = 0;
       omega = NaN;

   elseif sum(diag(R)) == -1
       theta = pi;
       if R(3,3) ~= -1
           omega = (1/sqrt(2*(1+R(3,3)))) * [R(1,3); R(2,3); 1+R(3,3)];
       elseif R(2,2) ~= -1
           omega = (1/sqrt(2*(1+R(2,2)))) * [R(1,2); 1+R(2,2); R(3,2)];
       else %R(1,1) ~= 0
           omega = (1/sqrt(2*(1+R(1,1)))) * [1+R(1,1); R(2,1); 1+R(3,1)];
       end
   else
       theta = acos(0.5 * (sum(diag(R)) - 1 ));
       omega_hat = (R - R') / (2 * sin(theta)); % screw symmetric matrix
       omega_1 = omega_hat(3,2);
       omega_2 = omega_hat(1,3);
       omega_3 = omega_hat(2,1);

       omega = [omega_1; omega_2; omega_3];

   end
   return
end
```

Test:

```
% 1a rotation_2_axis_angle
% To find an axis angle from a rotation matrix:
```

```matlab
% The first singularity case is checked if R = I, because there is no
% rotation in this case.
% The second case checks trace(R) == -1, as it would be a rotation by pi.
% Finally, if those checks pass, the axis and angle are returned.
   %case 1
   R = eye(3);
   [theta, omega] = rotation_2_axis_angle(R);
   assert(isnan(omega))
   assert(theta == 0)

   % case 2
   R = [1 0 0; 0 -1 0; 0 0 -1];
   [theta, omega] = rotation_2_axis_angle(R);
   assert( abs(theta - pi) < eps)
   % case 3 (MR Example 3.12)
   R = [0.8658  -0.2502  0.4333;
        0.2502   0.9665  0.0581;
        -0.4333  0.0581  0.8994];
   [theta, omega] = rotation_2_axis_angle(R);
   assert(norm(omega - [0;0.866;0.5]) < eps);
   assert(abs(theta - 0.524) < eps);
```

rotation_2_quaternion

This function takes in a rotation matrix and uses the formula provided by ASBR notes W3L1.

```
% PA 1b
function q = rotation_2_quaternion(R)
    % param: R (rotation matrix)
    % return: q (4x1 quaternion matrix)
    % reference: ASBR W3L1 pg11, MR Appendix B3

    % check R is in SO(3)
    if ~RinSO3(R)
        q = nan;
        return
    end

    q0 = 0.5 * sqrt(R(1,1) + R(2,2) + R(3,3) + 1);
    q1 = 0.5 * sign(R(3,2) - R(2,3)) * sqrt(R(1,1) - R(2,2) - R(3,3) + 1);
    q2 = 0.5 * sign(R(1,3) - R(3,1)) * sqrt(R(2,2) - R(3,3) - R(1,1) + 1);
    q3 = 0.5 * sign(R(2,1) - R(1,2)) * sqrt(R(3,3) - R(1,1) - R(2,2) + 1);
    q = [q0; q1; q2; q3];

    return
end
```

Test:

```
% 1b rotation_2_quaternion
% reference: rotm2quat() provided by matlab
    R = eye(3);
    q = rotation_2_quaternion(R);
    qmatlab = rotm2quat(R);
    assert(norm(q - qmatlab') < eps);

    R = [0.8658  -0.2502  0.4333;
         0.2502   0.9665  0.0581;
         -0.4333  0.0581  0.8994];
    q = rotation_2_quaternion(R);
    qmatlab = rotm2quat(R);
    assert(norm(q - qmatlab') < eps);
    R = [0 0 1; 0 1 0; -1 0 0];
    q = rotation_2_quaternion(R);
    qmatlab = rotm2quat(R);
    assert(norm(q - qmatlab') < eps);
```

rotation_2_ZYZ

This function takes in a rotation matrix and outputs the ZYZ angles. The additional input is the input angle, which designates the input angle quadrant to perform calculations.

```
% PA 1c
function [phi, theta, psi] = rotation_2_ZYZ(R, theta_in)
   % param: R (3x3 rotation matrix)
   % param: theta_in (designates the input angle quadrant
   % return: phi
   % return: theta
   % return: psi
   % reference: ASBR W3L1 Pg4

   % check R is in SO(3)
   if ~RinSO3(R)
       phi = nan;
       theta = nan;
       psi = nan;
       return
   end
   % check for range of theta_in
   if theta_in > 0
       phi = atan2(R(2,3), R(1,3));
       theta = atan2(sqrt(R(1,3)^2 + R(2,3)^2), R(3,3));
       psi = atan2(R(3,2), -R(3,1));

   else
      phi = atan2(-R(2,3), -R(1,3));
      theta = atan2(-sqrt(R(1,3)^2 + R(2,3)^2), R(3,3));
      psi = atan2(-R(3,2), R(3,1));

   end
end
```

Test:

TODO

```
% 1c rotation_2_ZYZ
   R = eye(3);
   [phi, theta, psi] = rotation_2_ZYZ(R, -pi/2);
   angles_matlab = rotm2eul(R, 'ZYZ');
   assert(norm([angles_matlab(1) - phi; ...
                angles_matlab(2) - theta; ...
                angles_matlab(3) - psi]) < eps);

   R = [0.8658  -0.2502  0.4333;
        0.2502   0.9665  0.0581;
        -0.4333  0.0581  0.8994];
   [phi, theta, psi] = rotation_2_ZYZ(R, -pi/2);
```

```matlab
angles_matlab = rotm2eul(R, 'ZYZ');
assert(norm([angles_matlab(1) - phi; ...
             angles_matlab(2) - theta; ...
             angles_matlab(3) - psi]) < eps);
R = [0 0 1; 0 1 0; -1 0 0];
[phi, theta, psi] = rotation_2_ZYZ(R, -pi/2);
angles_matlab = rotm2eul(R, 'ZYZ');
assert(norm([angles_matlab(1) - phi; ...
             angles_matlab(2) - theta; ...
             angles_matlab(3) - psi]) < eps);
```

rotation_2_rpy

This function takes in a rotation matrix and outputs the roll-pitch-yaw angles. The additional input is the input angle, which designates the input angle quadrant to perform calculations.

```
% PA 1c
function [phi, theta, psi] = rotation_2_rpy(R, theta_in)
   % param: R (3x3 rotation matrix)
   % param: theta_in (designates the input angle quadrant
   % return: phi
   % return: theta
   % return: psi
   % reference: ASBR W3L1 Pg7
   % check R is in SO(3)
   if ~RinSO3(R)
       phi = nan;
       theta = nan;
       psi = nan;
       return
   end
   if theta_in > -pi/2 && theta_in < pi/2
       phi = atan2(R(2,1), R(1,1));
       theta = atan2(-R(3,1), sqrt(R(3,2)^2 + R(3,3)^2));
       psi = atan2(R(3,2), R(3,3));
   elseif theta_in > pi/2 && theta_in < 3*pi/2
       phi = atan2(-R(2,1), -R(1,1));
       theta = atan2(-R(3,1), -sqrt(R(3,2)^2 + R(3,3)^2));
       psi = atan2(-R(3,2), -R(3,3));
   end
end
```

Test:

```
% 1d rotation_2_rpy
% reference: rotm2eul() provided by matlab
   R = eye(3);
   [phi, theta, psi] = rotation_2_rpy(R, 0);
   angles_matlab = rotm2eul(R);
   assert(norm([angles_matlab(1) - phi; ...
                angles_matlab(2) - theta; ...
                angles_matlab(3) - psi]) < eps);

   R = [0.8658  -0.2502  0.4333;
        0.2502   0.9665  0.0581;
        -0.4333  0.0581  0.8994];
   [phi, theta, psi] = rotation_2_rpy(R, 0);
   angles_matlab = rotm2eul(R);
   assert(norm([angles_matlab(1) - phi; ...
                angles_matlab(2) - theta; ...
```

```
             angles_matlab(3) - psi]) < eps);
R = [0 0 1; 0 1 0; -1 0 0];
[phi, theta, psi] = rotation_2_rpy(R, 0);
angles_matlab = rotm2eul(R);
assert(norm([angles_matlab(1) - phi; ...
             angles_matlab(2) - theta; ...
             angles_matlab(3) - psi]) < eps);
```

axis_angle_2_rotation

This function takes in an axis angle and rotation angle to compute Rodrigues' formula for a rotation matrix

```
% PA 2a
function R = axis_angle_2_rotation(omega, theta)
   % param: omega (3x1 orientation vector)
   % param: theta (angle [radian])
   % return: R (rotation matrix)
   % reference: MR Proposition 3.11, Example
   omega_hat = [   0          -omega(3)   omega(2);
                omega(3)        0         -omega(1);
                -omega(2)     omega(1)      0      ];
   R = eye(3) + sin(theta) * omega_hat + (1 -  cos(theta))*omega_hat^2;

   return
end
```

Test:

```
% 2a axis_angle_2_rotation (MR Example 3.12)
   omega = [0 0.866 0.5];
   theta = 0.524;
   R = axis_angle_2_rotation(omega, theta);
   R_book = [0.8658  -0.2502  0.4333;
             0.2502   0.9665  0.0581;
            -0.4333  0.0581  0.8994];
   assert(norm(R - R_book) < eps);
```

quaternion_2_rotation

This function takes in an axis angle and rotation angle and computes the quaternion output

```
% PA 2b
function R = quaternion_2_rotation(q)
   % param: q (4x1 quaternion)
   % return: R (3x3 rotation matrix)
   % reference: ASBR W3L1 pg11, MR Appendix B.3
   q0 = q(1);
   q1 = q(2);
   q2 = q(3);
   q3 = q(4);

   R = zeros(3,3);
   % 1st column
   R(1, 1) = q0*q0 + q1*q1 - q2*q2 - q3*q3;
   R(2, 1) = 2 * (q0*q3 + q1*q2);
   R(3, 1) = 2 * (q1*q3 - q0*q2);
   % 2nd column
   R(1, 2) = 2 * (q1*q2 - q0*q3);
   R(2, 2) = q0*q0 - q1*q1 + q2*q2 - q3*q3;
   R(3, 2) = 2 * (q0*q1 + q2*q3);
   % 3rd column
   R(1, 3) = 2 * (q0*q2 + q1*q3);
   R(2, 3) = 2 * (q2*q3 - q0*q1);
   R(3, 3) = q0*q0 - q1*q1 - q2*q2 + q3*q3;
   return
end
```

Test:

```
% 2b quaternion_2_rotation
   q = [0.7071 0.7071 0 0];
   R = quaternion_2_rotation(q);
   Rmatlab = quat2rotm(q);
   assert(norm(R - Rmatlab) < eps);

   q = [1 0 0 0];
   R = quaternion_2_rotation(q);
   Rmatlab = quat2rotm(q);
   assert(norm(R - Rmatlab) < eps);
   q = [0.9659 0 0.2243 0.1295];
   R = quaternion_2_rotation(q);
   Rmatlab = quat2rotm(q);
   assert(norm(R - Rmatlab) < eps);
```

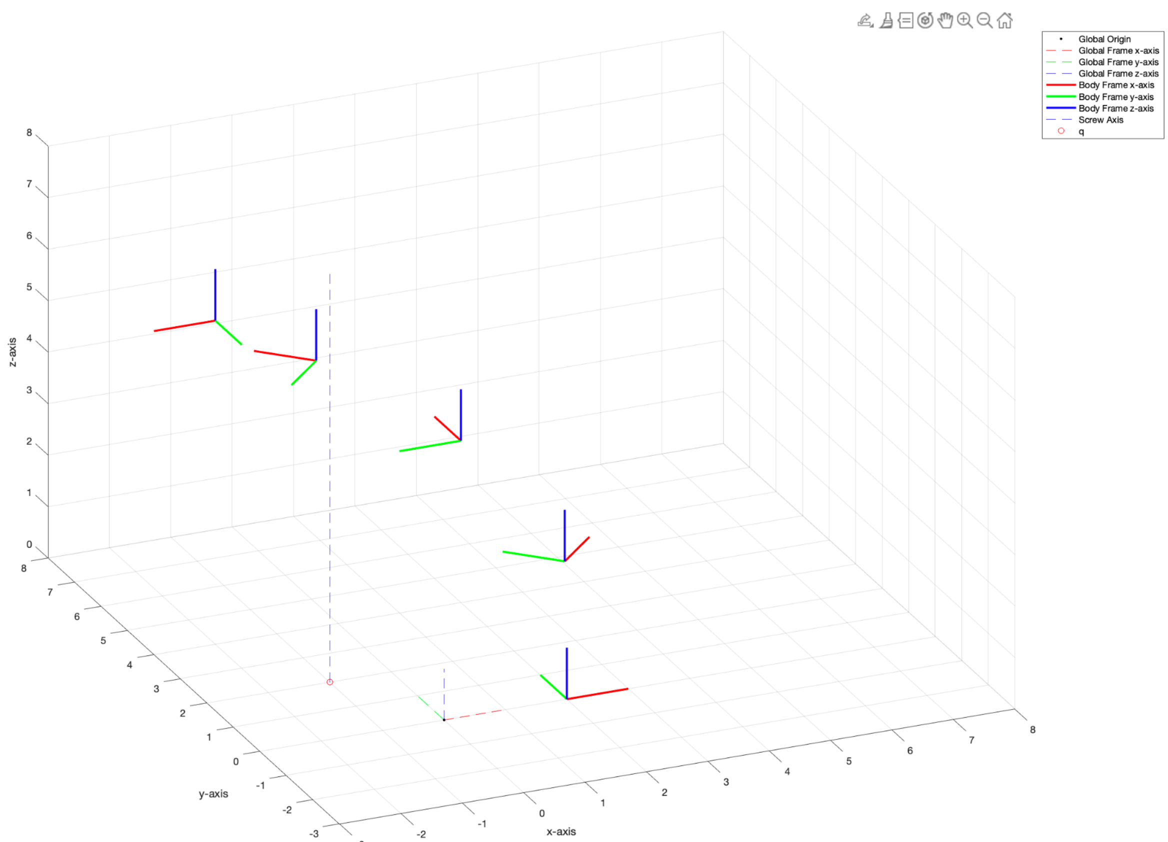
Screw Axis

In this part of the programming exercise, we used the functions we wrote to plot transformations given an initial configuration T and screw axis specified by $\{q, \hat{s}, h\}$. The algorithm steps are in comments embedded in the code.

In this example, the input parameters are:

```
q = [0,2,0]';
shat = [0, 0, 1]';
h = 2;
theta = pi;
T = [1 0 0 2;
    0 1 0 0;
    0 0 1 0;
    0 0 0 1];
```

The output looks like below:

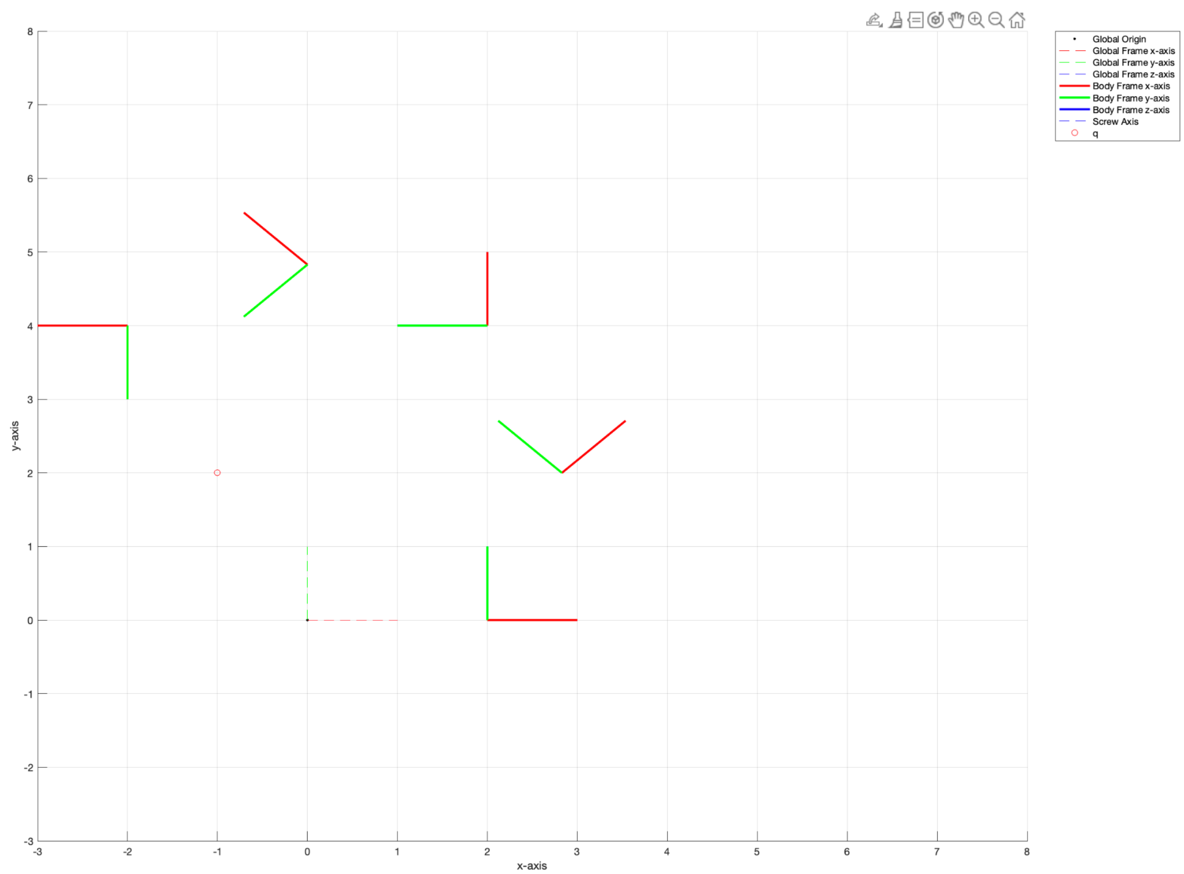



```
screw axis:
          0
          0
```

```
    1.0000
    2.0000
    1.0000
    2.0000

theta:
    3.1416
```

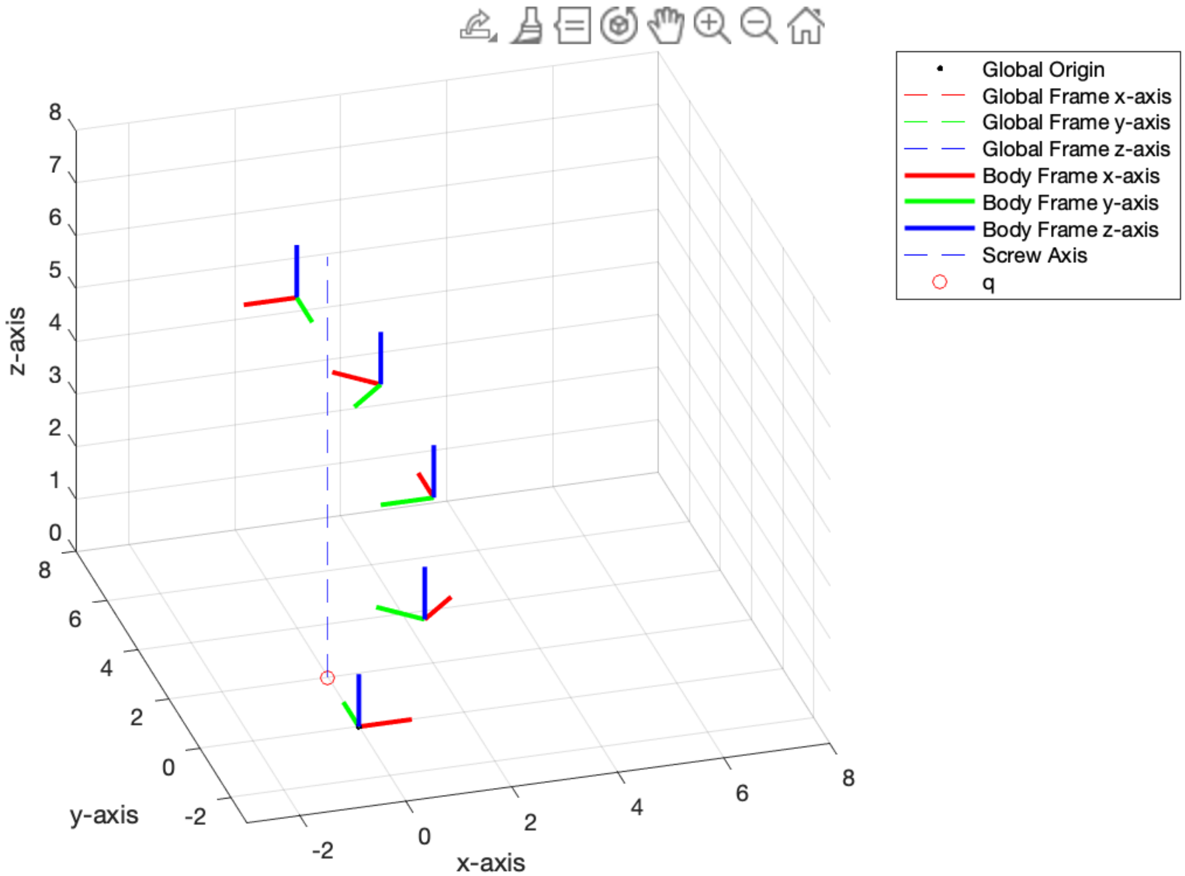
Because there was only rotation around z, we found it interesting to view from the top down view, i.e. the x,y plane.



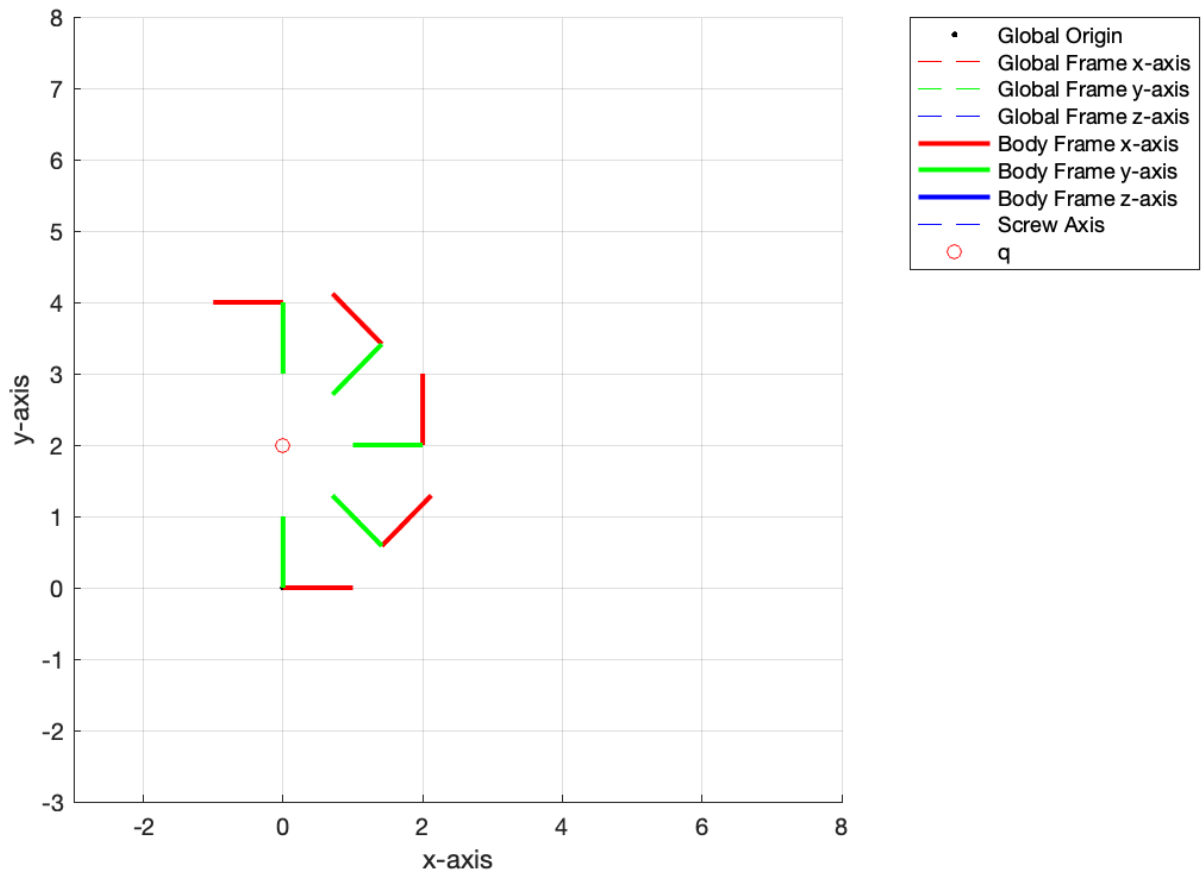
One method to check that our matlab code was correct was to set the input parameters to rotate around the global origin.

```
q = [0,2,0]';
shat = [0, 0, 1]';
h = 2;
theta = pi;
T = [1 0 0 0;
    0 1 0 0;
    0 0 1 0;
```

```
0 0 0 1];
```



Global Origin
Global Frame x-axis
Global Frame y-axis
Global Frame z-axis
Body Frame x-axis
Body Frame y-axis
Body Frame z-axis
Screw Axis
q
z-axis
y-axis
x-axis



In this case, the x-y plane is symmetrical about the z axis, rotated by pi.

```
function [T_final, S_final, q_final] = screw(T_input, q, shat, h, thetas)
    % param: T_input (4x4 transformation matrix)
    % param: s (screw axis  [q, shat, h] in fixed frame s)
    % param: thetas (range of distances traveled along screw axis theta)
    % return: T_final
    % reference: ASBR W5L1
    S_input = [shat; -cross(shat,q)+h*shat];
    omega = S_input(1:3);
    v = S_input(4:6);
    omega_hat =  [    0          -omega(3)   omega(2);
                 omega(3)        0         -omega(1);
                 -omega(2)     omega(1)      0       ];
    figure(1)
    % origin frame
    plot_handle(1) = plot3(0, 0, 0, '.k');
    hold on
    plot_handle(2) = line([0 1], [0 0], [0 0], 'Color', 'r', 'LineStyle', '--');
    plot_handle(3) =line([0 0], [0 1], [0 0], 'Color', 'g', 'LineStyle', '--');
    plot_handle(4) = line([0 0], [0 0], [0 1], 'Color', 'b', 'LineStyle', '--');
    T_final = nan; % return value
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%$$$ plot rotated frames %%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for theta_ = thetas
        R = axis_angle_2_rotation(shat,theta_);
        p =
(eye(3)*theta_+(1-cos(theta_))*omega_hat+(theta_-sin(theta_))*omega_hat^2)*v;
        T_t = [R p;0 0 0 1]*T_input;
        R_t = T_t(1:3,1:3);
        P_t = [T_t(1,4), T_t(2,4), T_t(3,4)];
        P_x = P_t(1);
        P_y = P_t(2);
        P_z = P_t(3);
        % plot rotated frame
        plot_handle(5) = line(([0 R_t(1,1)]+P_x), ...
                              ([0 R_t(2,1)]+P_y), ...
                              ([0 R_t(3,1)]+P_z), ...
                              'Color', 'r', 'LineWidth', 2);
        plot_handle(6) = line(([0 R_t(1,2)]+P_x), ...
                              ([0 R_t(2,2)]+P_y), ...
                              ([0 R_t(3,2)]+P_z), ...
                              'Color', 'g', 'LineWidth', 2);
        plot_handle(7) = line(([0 R_t(1,3)]+P_x), ...
                              ([0 R_t(2,3)]+P_y), ...
                              ([0 R_t(3,3)]+P_z), ...
                              'Color', 'b', 'LineWidth', 2);
        hold on
        if theta_ == thetas(end)
            T_final = T_t;
        end
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%% plot screw axis for T_final %%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    R_final = T_final(1:3,1:3);
    p_final = T_final(1:3,4);
    S_final = zeros(6,1); % Return value
    % Reference MR 3.3.3.2 Matrix Logarithm of Rigid-Body Motions
    if R_final == eye(3)
        omega = [0;0;0];
        v = p_final/norm(p_final);
        theta = norm(p_final);
        theta_dot = norm(v);
        h = omega'*v/theta_dot;
        q = cross(omega,v)/theta_dot;
        % screw axis
        S_final = [omega;v];
    else
        [theta, omega] = rotation_2_axis_angle(R_final);
```

```
        omega_hat = [0 -omega(3) omega(2);
                     omega(3) 0 -omega(1);
                     -omega(2) omega(1) 0];
        % Rodrigues Formula to obtain axis and angle
        G_inv = 1/theta*eye(3) - 1/2*omega_hat +
(1/theta-1/2*cot(theta/2))*omega_hat^2;
        v = G_inv * p_final;
        theta_dot = norm(omega);
        h = omega'*v/theta_dot;
        q = cross(omega,v)/theta_dot;
        % screw axis
        S_final = [omega;v] / norm(omega);
    end
    disp('screw axis: ')
    disp(S_final)
    disp('theta: ')
    disp(theta)
    x1 = q(1);
    y1 = q(2);
    z1 = q(3);
    if omega == [0;0;0]
        omega = v;
    end
    x2 = q(1) + omega(1);
    y2 = q(2) + omega(2);
    z2 = q(3) + omega(3);
    nx = x2 - x1;
    ny = y2 - y1;
    nz = z2 - z1;
    len = 100;
    xx = [x1 - len*nx, x2 + len*nx];
    yy = [y1 - len*ny, y2 + len*ny];
    zz = [z1 - len*nz, z2 + len*nz];
    plot_handle(8) = plot3(xx, yy, zz,'--','Color','b');
    hold on
    plot_handle(9) = plot3(q(1),q(2),q(3),'o','Color','r');
    xlim([-3 8])
    ylim([-3 8])
    zlim([0 8])
    xlabel('x-axis')
    ylabel('y-axis')
    zlabel('z-axis')
    grid on
    legend(plot_handle([1, 2, 3, 4, 5, 6, 7, 8, 9]), ...
           {'Global Origin', ...
           'Global Frame x-axis', ...
           'Global Frame y-axis', ...
           'Global Frame z-axis', ...
           'Body Frame x-axis', ...
```

```
          'Body Frame y-axis', ...
          'Body Frame z-axis', ...
          'Screw Axis', ...
          'q'})
end
```