CMPSCI 403: Introduction to Robotics: Perception, Mechanics, Dynamics, and Control

HW 03: 3D open-chain robot representation

Joshua Holden

October 3, 2021

1 Euler Angles to Orientation Matrices

Conversion of Euler angles to orientation matrices can be done by multiplying the angle along the respective elementary rotation matrices.

I implemented this in a MatLab function $eulerToOrientation$ that takes an ZYX Euler angle $eAng$ and returns the orientation matrix $rotMatrix$ in $eulerToOrientation.m$.

```
function rotMatrix = eulerToOrientation(eAng)
% Converts a ZYX Euler angle vector to an orientation(rotation) matrix
Rx = [1 0 0; 0 cos(eAng(3)) -sin(eAng(3)); 0 sin(eAng(3)) cos(eAng(3))];
Ry = [cos(eAng(2)) 0 sin(eAng(2)); 0 1 0; -sin(eAng(2)) 0 cos(eAng(2))];
Rz = [cos(eAng(1)) -sin(eAng(1)) 0; sin(eAng(1)) cos(eAng(1)) 0; 0 0 1];
rotMatrix = Rz*Ry*Rx;
end
```

In order to verify that the function was correct, I tested the function with this script $eulerToOrientationTest.m$:

```
zyx1 = [0.3 0.2 0.5];
zyx2 = [0.7 pi pi/2];
zyx3 = [pi/3 0 0];
disp(eulerToOrientation(zyx1));
disp(eulerToOrientation(zyx2));
disp(eulerToOrientation(zyx3));
```

and received this output:

```
>> eulerToOrientationTest
    0.9363   -0.1684    0.3082
    0.2896    0.8665   -0.4065
   -0.1987    0.4699    0.8601

   -0.7648    0.0000    0.6442
   -0.6442    0.0000   -0.7648
   -0.0000   -1.0000   -0.0000

    0.5000   -0.8660         0
    0.8660    0.5000         0
         0         0    1.0000
```

2 SE3 Manipulation and Frame of Reference

The frame of reference starts at SE3 at the origin (position (0,0,0)) and with YZX Euler angle (0,0,0). Each subsequent frame has a new frame of reference based on the SE3 of the previous frame.
Manipulation of frames in SE3 can be easily handled with 12 values, 9 for a rotation matrix in R^3 and 3 for a position vector in R^3, as such: $\begin{bmatrix} & M & & P \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix}$
The MatLab code to implement the SE3 class is found below:

```
classdef SE3
classdef SE3
    % Class to represent the Special Euclidean group in 3D.

    properties
        Matrix
    end

    methods
        function obj = SE3(posVector,rotMatrix)
            % Constructor for SE3 given rotation and translation
            obj.Matrix = [rotMatrix posVector(:); 0 0 0 1];
        end

        function newObj = mult(frame1, frame2)
            % Method for chaining SE3 frames in global perspective
            newObj.Matrix = frame1.Matrix * frame2.Matrix;
        end
    end

    methods (Static)
        function newObj = inverse(frame)
            % Method to invert SE3
            rotMatrix = frame.Matrix(1:3, 1:3);
            posVector = frame.Matrix(1:3, 4);
            rotInverted = transpose(rotMatrix);
            posInverted = -(rotInverted * posVector);
            newObj = SE3(posInverted, rotInverted);
        end

        function newObj = getPos(frame)
            % Returns position vector of a frame
            newObj = frame.Matrix(1:3, 4);
        end

        function newObj = getRot(frame)
            % Returns position vector of a frame
            newObj = frame.Matrix(1:3, 1:3);
        end
    end
end
```

The class only has a single property, the matrix containing the rotation and translation of the frame. The constructor populates the matrix with a position vector and rotation matrix, as output by *eulerToOrientation*. Multiplication is handled with *mult*, and the inverse frame is generated by *inverse*, a static method that transposes the rotation matrix(R^T) and inverts the position vector($-R^T * P$).

The tests for these were made in a test file $frameManipulationTest.m$:

```
frame0 = SE3([0 0 0], eulerToOrientation([0 0 0]));
frame1 = SE3([0.4 0.8 1.2], eulerToOrientation([0.3 0.2 0.5]));
frame2 = SE3([-0.4 0.5 1], eulerToOrientation([0.7 pi pi/2]));
frame3 = SE3([0.5 -0.8 1.2], eulerToOrientation([pi/3 0 0]));

disp("Frame Matrices");
disp(frame0.Matrix);
disp(frame1.Matrix);
disp(frame2.Matrix);
disp(frame3.Matrix);

endFrame = mult(mult(mult(frame0,frame1),frame2),frame3);

disp("End Frame");
disp(endFrame.Matrix);

invertedFrame = SE3.inverse(endFrame);

disp("Frame 3 to Frame 0");
disp(invertedFrame.Matrix);
```

To receive the output:

```
>> frameManipulationTest
Frame Matrices
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

    0.9363   -0.1684    0.3082    0.4000
    0.2896    0.8665   -0.4065    0.8000
   -0.1987    0.4699    0.8601    1.2000
         0         0         0    1.0000

   -0.7648    0.0000    0.6442   -0.4000
   -0.6442    0.0000   -0.7648    0.5000
   -0.0000   -1.0000   -0.0000    1.0000
         0         0         0    1.0000

    0.5000   -0.8660         0    0.5000
    0.8660    0.5000         0   -0.8000
         0         0    1.0000    1.2000
         0         0         0    1.0000

End Frame
   -0.5708    0.3721    0.7319    1.0706
   -0.0378    0.8785   -0.4762   -0.5756
   -0.8202   -0.2995   -0.4874    2.4024
         0         0         0    1.0000

Frame 3 to Frame 0
   -0.5708   -0.0378   -0.8202    2.5598
    0.3721    0.8785   -0.2995    0.8267
    0.7319   -0.4762   -0.4874    0.1131
         0         0         0    1.0000
```

3 Plotting Frames

Frames are plotted with the given *drawCoordinate3D*:

```
close all;
clc;
clf;

% initialize frames
frame0 = SE3([0 0 0], eulerToOrientation([0 0 0]));
frame1 = SE3([0.4 0.8 1.2], eulerToOrientation([0.3 0.2 0.5]));
frame2 = SE3([-0.4 0.5 1], eulerToOrientation([0.7 pi pi/2]));
frame3 = SE3([0.5 -0.8 1.2], eulerToOrientation([pi/3 0 0]));
% calculate positions
frame1_r = mult(frame0, frame1);
frame2_r = mult(frame1_r, frame2);
frame3_r = mult(frame2_r, frame3);

% draw figure
figure
hold on
grid on
drawLine3D(SE3.getPos(frame0),SE3.getPos(frame1_r));
drawLine3D(SE3.getPos(frame1_r),SE3.getPos(frame2_r));
drawLine3D(SE3.getPos(frame2_r),SE3.getPos(frame3_r));
drawCoordinate3D(SE3.getRot(frame0),SE3.getPos(frame0));
drawCoordinate3D(SE3.getRot(frame1_r),SE3.getPos(frame1_r));
drawCoordinate3D(SE3.getRot(frame2_r),SE3.getPos(frame2_r));
drawCoordinate3D(SE3.getRot(frame3_r),SE3.getPos(frame3_r));
xlabel('$x$','interpreter','latex','fontsize',20)
ylabel('$y$','interpreter','latex','fontsize',20)
zlabel('$z$','interpreter','latex','fontsize',20)
axis equal

view(25,30)
```

To receive a plot with T0, T01, T02, T03:

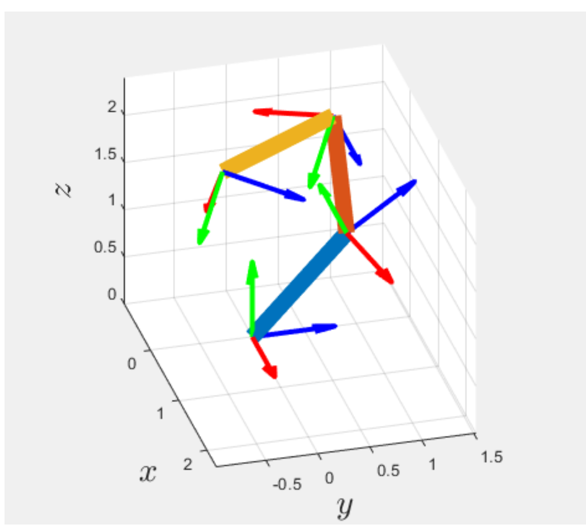

Figure 1: Frames with lines.

4 Frame Animation

I cannot figure out how to make the animation look nice, but end-effector moves as follows:

```
close all;
clc;

% initialize frames
frame0 = SE3([0 0 0], eulerToOrientation([0 0 0]));
frame1 = SE3([0.4 0.8 1.2], eulerToOrientation([0.3 0.2 0.5]));
frame2 = SE3([-0.4 0.5 1], eulerToOrientation([0.7 pi pi/2]));
frame3 = SE3([0.5 -0.8 1.2], eulerToOrientation([pi/3 0 0]));
% calculate positions
frame1_r = mult(frame0, frame1);
frame2_r = mult(frame1_r, frame2);
frame3_r = mult(frame2_r, frame3);
t = 0;

figure

for i=1:50
    clf;
    t = t+0.02;
    theta = pi*t;
    hold on

    % draw the arm
    drawLine3D(SE3.getPos(frame0),SE3.getPos(frame1_r));
    drawLine3D(SE3.getPos(frame1_r),SE3.getPos(frame2_r));
    drawLine3D(SE3.getPos(frame2_r),SE3.getPos(frame3_r));
    drawCoordinate3D(SE3.getRot(frame0),SE3.getPos(frame0));
    drawCoordinate3D(SE3.getRot(frame1_r),SE3.getPos(frame1_r));
    drawCoordinate3D(SE3.getRot(frame2_r),SE3.getPos(frame2_r));
    drawCoordinate3D(SE3.getRot(frame3_r),SE3.getPos(frame3_r));

    % find end effector
    posVecEE = [(sin(theta)*0.1+0.05); (cos(theta)*0.3+0.08); (sin(theta)+0.5)];
    rotMatEE = SE3.getRot(frame3);
    frameEE = SE3(posVecEE, rotMatEE);
    frameEE_r = mult(frame3_r, frameEE);

    % draw the end effector
    drawLine3D(SE3.getPos(frame3_r),SE3.getPos(frameEE_r));
    drawCoordinate3D(SE3.getRot(frameEE_r),SE3.getPos(frameEE_r));

    % axis label
    xlabel('$x$','interpreter','latex','fontsize',20)
    ylabel('$y$','interpreter','latex','fontsize',20)
    zlabel('$z$','interpreter','latex','fontsize',20)
    axis equal
    view(85,10)

    pause(0.001);
    hold off
    grid on
end
```

Some frames of the animation are:

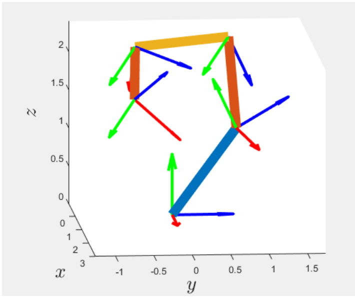

Figure 2: Arm animation part 1

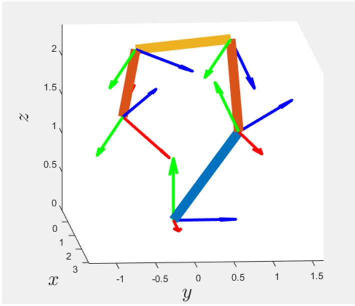

Figure 3: Arm animation part 2

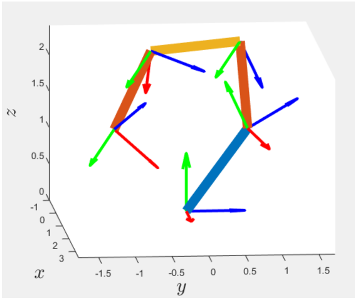

Figure 4: Arm animation part 3

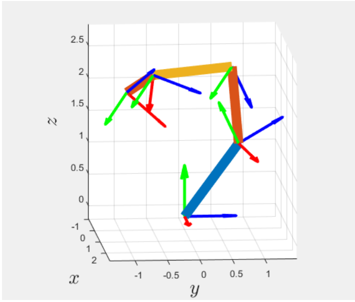

Figure 5: Arm animation part 4 (with grid)