Exercise 1: Euler angles (ex1.py)

This program displays an interpolation of coordinate frames where rotations are represented by ZYX Euler angles. These, by convention, take on values in the range [0,2π)x[-π/2,π/2]x[0,2π).

1. Notice that the current linear Euler angle interpolation function does not interpolate between the two endpoints (π/4,0,0) and (7π/4,0,0) along a minimal-length curve (a geodesic) – it rotates 270° instead of 90°. Modify the interpolate_euler_angles function so that the path does indeed interpolate the first angle along a geodesic – rotating 90° as desired.

   Make sure it also does so for other “simple” interpolations, such as from (0,0,π/4) and (0,0, 7π/4). [You may test different endpoints by modifying the self.ea and self.eb values in the constructor of GLEulerRotationTest.]

2. Specify a different set of interpolation endpoints where simple interpolation of Euler angles fails to produce a geodesic – that is, the frame rotates an excessive amount to blend between the endpoints. In your program, take snapshots of the interpolation and describe what is happening.

Exercise 2: Rotation matrices (ex2.py)

This program represents rotations as 3x3 matrices in the format specified in the klampt.so3 module (a list of 9 numbers in column-major order).

To interpolate between two matrices, it is currently converting both matrices to a moment (aka exponential map) representation and interpolating linearly in that space. This does not in general interpolate along a geodesic. Modify the interpolate_rotation function so that it indeed performs geodesic interpolation.

[No peeking at the klampt.so3.interpolate function! However, you may use other functions in klampt.so3, such as mul, inv, matrix, moment, from_moment, axis_angle and from_axis_angle].

Verify that your function is indeed correct by printing out the absolute angle (klampt.so3.angle) between the interpolated rotation matrix and the endpoints. This angle should prove to be a linear interpolation.