ECE383/MEMS442/ECE590 Fall 2018
Lab 1: Python, Linear Algebra, and Transformations

Due date: 9/12/2018

Instructions: This assignment is to be completed *individually*. You are allowed to discuss the problems with your classmates, but all work must be your own. You are not permitted to copy code, take written notes, or look up online solutions.

All coding will be done on the Klamp't Jupyter system. You may code your calculations manually using the elementary operations found in the Python math module, or you may use the functions in the klampt.vectorops or klampt.so2 module.

To submit, upload your files Lab1a.ipynb, Lab1b.ipynb, Lab1c.ipynb, and Lab1d.ipynb, and upload them to Sakai.

Problem 1: Direction and Magnitude

Lab 1a displays two endpoints of a line segment, which move according to the functions source_motion(t) and target_motion(t). The on-screen display incorrectly calculates the length and angle of this segment, printing out 0 and 0, respectively. Your job is to properly calculate these values.

Complete the function lab1a(point1,point2) which should return a tuple (length,angle) describing the magnitude and heading of the vector from point1 to point2.

Hint: run the "selfTest()" function to verify that you are performing the calculations properly.

Problem 2: Rotation of Points

Lab 1b displays a clock-like object with a center point and two peripheral points. They should be rotating simultaneously about the center point, but the display is broken.

Complete the function lab1b(point,angle) which should return a new point rotated about the origin by the given angle. Both input and output points are given by a length-2 tuple (or list) of floats.

Hint: run the "selfTest()" function to verify that you are performing the calculations properly.

Problem 3: Composition of Rigid Transformations

Lab 1c displays a car-like vehicle. By default the car will use automatic controls (chosen at random by default), but you can also drive it around using the arrow buttons.

The display of the car is broken! It does not properly rotate, and the front wheels do not properly indicate the steering angle.

1. Complete the function get_rotation_matrix(xform). Here an xform is a tuple (tx,ty,angle) indicating the translation and rotation of a rigid body transformation. get_rotation_matrix

should return a 2x2 matrix so that matrix-vector multiplication with a point should give the rotated point (i.e., duplicate the behavior of lab1b()). When this is correctly implemented, the car body should rotate as the car turns.

2. Complete lab1c(xform1,xform2). This function composes two xforms together so that applying the result to a point is equivalent to first applying xform2, and then applying xform1. When this is correctly implemented, the front wheels should rotate to indicate steering angle.

To help you debug, you can change the automatic controls by editing the control() function. Return values 'up' and 'down' increase and decrease the velocity, and 'left' and 'right' increase and decrease the steering angle.

Hint: run the "selfTest()" function to verify that you are performing the calculations properly.

Problem 4: 3D Rotations

Lab 1d displays an interpolation of coordinate frames where rotations are represented by ZYX Euler angles. These, by convention, take on values in the range $[0,2\pi)\times[-\pi/2,\pi/2]\times[0,2\pi)$.

1. Notice that the current linear Euler angle interpolation function does not interpolate between the two endpoints $(\pi/4,0,0)$ and $(7\pi/4,0,0)$ 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 $(0,0,\pi/4)$ and $(0,0, 7\pi/4)$. You should test different endpoints by modifying the ea and eb values.

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. Observe the results.

   The do_interpolate(u) function has a commented-out line that uses the built-in interpolation function in the klampt.so3 module. Temporarily modify the function so it returns the value produced by this line rather than Euler angle interpolation, and observe the results.

   Explain this discrepancy. Why does Euler angle interpolation seem to rotate an excessive amount? What is klampt.so3 doing differently?

   (Use the commented-out space at the bottom of the file for your answers)