DEPARTMENT OF MECHANICAL ENGINEERING
RICE UNIVERSITY

MECH 498/598: INTRODUCTION TO ROBOTICS

Lab #1: Homogeneous Transforms and Robot Forward Kinematics

Due Thursday, 2/17/2026

Assignment Description
The purpose of this assignment is for you to learn more about different representations of orientation in 3D, to think closely about the structure of the D-H parameterization, and to practice the use of the D-H convention for describing forward kinematics on a real robot, the PHANToM Premium. Be sure to **read the entire assignment** before beginning, so that you are aware of all instructions.

Submission Instructions
Your submission for this assignment will have 2 electronic components and 1 paper component.
The paper submission can be turned in to the ME dropbox, or in class.
This assignment uses an auto-grader for most of the code checking. This autograder is available through gradescope. You will receive an invitation to join the class and see instructions for submitting your code to the grader. You must upload all of your code via .zip file that you need for your code to run. For this lab, that is likely just lab1.py. The auto-grader can be run as many times as you want. Each submission it will run and give you a report on where it is failing. The result from the auto-grader IS your score for the electronic portion of the lab. If it says 70/70, that is what you will get.
Finally, there is a video submission that you must do through canvas. Please upload this to the lab1 assignment on canvas.

`Note`: I have provided a template file in canvas to start from. This names your functions and specifies their inputs and outputs to get you started. Be sure to follow the exact naming convention as specified in the problem and template in order for the auto-grader to work.
Also provided to your are two additional utility files that you can use to test and validate your answers.

- **`general_utility.py`** : contains a few simple checks and conversions. This file will grow throughout the labs but these functions will continue to exist.
- **`lab1_util.py`** : This contains plotting tools for plotting individual frames and the Phantom robot for you to debug with. These require properly formatted student functions to work.

`Last Note:` Your final answer can NOT utilize the PyKDL library, it should instead use Numpy for all of it's arrays. PyKDL is an outstanding tool for doing frame transformation math in Python. However, you are learning how to do frame transformation math. You should use numpy arrays for your final submission to learn how PyKDL works under the hood. If a final solution has PyKDL in it, it will receive 0 points. I do however encourage you to use PyKDL to debug your code as you go.
Tip for using Numpy:

- The "@" symbol is how to multiply numpy arrays together in the way discussed in class.
- Typical nomenclature is to import numpy as `import numpy as np` and use "np." throughout the code

---

1. [5 pts.] Consider two frames, frame {A} and frame {B}. The orientation of {B} can be obtained by the following sequence of operations:

   a. First, place {B} coincident with {A}
   b. Second, rotate {B} by $\theta_y = \frac{\pi}{2}$ radians about $\hat{Y}_A$.
   c. Third, rotate {B} by $\theta_z = \frac{\pi}{4}$ radians about $\hat{Z}_B$.

   Write a function **`rotate(P_B)`** that accepts the position vector **`P_B`**, referenced to frame {B}, and returns the vector **`P_A`**, which is the same vector now referenced to frame {A}. Here we are strictly adhering to the conventions of the textbook, i.e., ${}^{A}P = {}^{A}_{B}R \ {}^{B}P$.

2. [5 pts.] Write a function **`(T,T_inv) = euler_to_ht(angles,pos)`** that accepts a 3-element numpy vector called **`angles`** of Z-Y-X Euler angles and a 3-element numpy vector **`pos`** representing the position vector ${}^{A}P_{B}$. The function should return the appropriate homogeneous transformation matrix ${}^{A}_{B}T$ and it's inverse ${}^{A}_{B}T^{-1} = {}^{B}_{A}T$, using the symbolic formula – don't use numpy's built-in matrix inverse operator. Be sure to test it for any reasonable choice of input arguments (always a good idea.)

3. [15 pts.] Write a function **`rollr(roll)`** which accepts a scalar roll value (in radians) and returns the corresponding 3x3 rotation matrix as a np.ndarray of shape (3,3). [1] You should test your function and your familiarity with Python's plotting features by creating the following figure plots. (These do not need to be submitted.). I find that matplotlib is the best plotting resource, especially if you're already familiar with Matlab.

   (a) Plot all three of the column vectors of this matrix on a single figure. Test this with a few different rotations about the roll axis.

   (b) Plot a 3-element column vector. Experiment with plotting various vectors and transformed (rotated and translated) vectors.

   (c) Take two 3-element column vectors and plot the vector connecting the two points identified by those vectors.

   Repeat these steps twice, creating new functions **`pitchr(pitch)`** and **`yawr(yaw)`** that accept pitch and yaw angles, respectively, and return the appropriate rotation matrices.

4. [5 pts.] Use the functions you created above to write a new function **`rpyr(angles)`** that accepts a numpy vector of 3 [*roll, pitch, yaw*] (in radians) and returns the corresponding 3x3 rotation matrix. Test your function by plotting various general rotations.

5. [5 pts.] Write a function **`rpytf(twist)`** that accepts a single numpy vector of 6, **`twist,`** containing [*x, y, z, roll, pitch, yaw*] (angles in radians) and returns the corresponding 4x4 homogeneous transformation matrix. Test your function with the function **`draw_frame`** provided to you by plotting various general transformations.

6. [10 pts.] One way to think of DH parameters is as two pairs of translations and rotations, each pair being along the same axis. We will informally describe a paired translation and rotation along the same axis as a screw transform. Write a function **`screw_tf(translation,rotation,ax)`** that accepts two scalars, **`translation`** and **`rotation`**, and a 3D unit vector **`ax`** representing the axis of translation/rotation. In the book, you will find a description of the angle-axis representation of 3D orientation, which will allow you to generate a rotation about an arbitrary axis. Use the function **`draw_screw`** provided to you to test the function you have written, which it calls, and see some fun visualization with color.

[1] We define roll, pitch, and yaw as the X, Y, and Z fixed angles, as in Craig, section 2.8.

7. [5 pts.] Using your **`screw_tf`** function, write a function **`screw_dh(a,alpha,d,theta)`** that accepts four scalar DH parameters and returns a 4x4 homogeneous transformation matrix according to the DH convention. You can check your result using the formula for this matrix given in the book and in class.

8. [50 pts.] One popular robot configuration in haptics research is a 3-R manipulator. Examples of such robots include the PHANToM Premium haptic devices by 3D systems (https://www.3dsystems.com/haptics-devices/3d-systems-phantom-premium), shown in Figure 1, and the MacroRobot at Vanderbilt University (http://research.vuse.vanderbilt.edu/cim/research_haptics.html). Some versions of these robots attach a 3-R wrist to the end effector to support a stylus. Visit the Mechatronics and Haptic Interfaces (MAHI) Lab in to see the PHANToM robots that use this configuration. Note that although the PHANToM Premium has a parallel configuration, it can be treated as a serial robot with a simple geometric relationship between the angle of the last link (to which the stylus is attached) and the angles of the links to which the motors are attached. Assuming what we can actually specify are the angular positions of the motor shafts, and not the joints themselves, you will also need to specify the transmission ratios of the cable transmission used by the PHANToM, the ratios are provided in the code, but you will need to incorporate them into your functions properly.

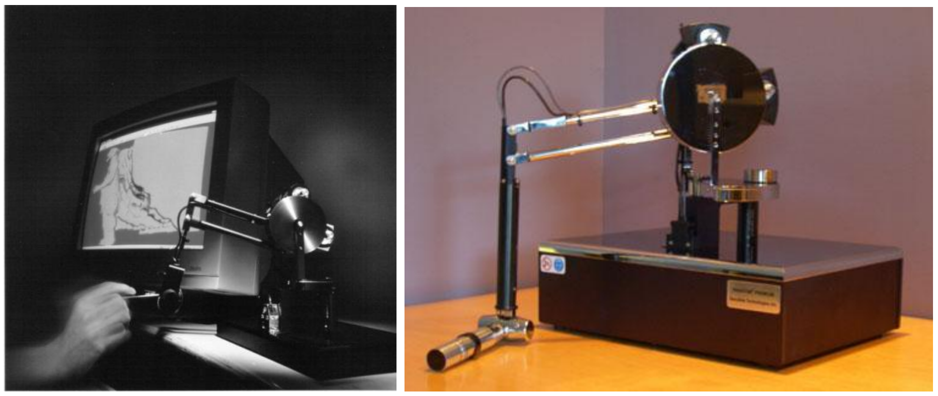

Figure 1. PHANToM Premium

(a). (20 of 50 points) Our real version of the robot only consists of the 3-R robot arm, and we will be adding the wrist (or gimbal) in simulation in part (b). The lengths l_1, l_2 and l_3 (in **millimeters**) are provided to you in the template lab1.py file. Also note the transmission ratios, motor input does not equal joint input. The transmission ratio values are provided in the lab1.py file. Make a table of the D-H parameters and develop the homogeneous transformation matrix describing the forward kinematic equations of the serial arm. Place the world frame (the zero frame) at the actual robot base, so that the first link length is included as a parameter. Draw the robot in its zero position so that the links are perpendicular to their neighboring links, as it is approximately depicted in the figure above on the right. In addition to assigning a world frame, and one to each of the three links. Also attach an *end-effector* frame to the robot's wrist center (the tip of the stylus in the figure) with its z axis pointing along the third link. Turn in your written solution to the PHANToM forward kinematics, including a 3D drawing of the robot and the table of D-H parameters, to the ME Drop Box OR in class.

(b). (20 of 50 points) For the following exercise in python, we will attach an additional frame to the serial manipulator, called the *gimbal* frame. The transform from the end-effector frame to the gimbal frame is a pure rotation generated from *roll-pitch-yaw* angles that we can choose arbitrarily. Write a function for the forward kinematics of the serial version of the PHANToM called **`[phantom_T_0_g,phantom_T]= phantom_fk(joint_angles,gimbal_angles)`** that takes as its input a vector **`joint_angles`** of the form [θ_1, θ_2, θ_3] describing the position of each link relative to the previous one, and a vector **`gimbal_angles`** of roll-pitch-yaw angles, and outputs a 4x4 homogeneous transformation **`phantom_T_0_g`** from the base frame to the gimbal frame and an array of arrays, **`phantom_T`** of the 4x4 homogeneous transformations for this robot (with positions in **millimeters**). The transforms needed are { ${}^0T_1$, ${}^1T_2$, ${}^2T_3$, ${}^3T_e$, ${}^eT_g$ }, where the subscripts *e* and *g* describe the "end-effector" and "gimbal" frames. Test your function using various inputs and the function **`draw_phantom(actuator_angles,gimbal_angles)`** and the function **`move_phantom(path_file)`** provided to you. In order to convert the input **`actuator_angles`** into joint angles, you must also write a function **`actuator_to_joint(actuator_angles)`** that takes a 3-vector of motor angles and converts them to PHANToM joint angles using the transmission ratios that you measure on the actual robot (HINT: this mapping contains more than just scalar multiplication!)

Part (b) will be graded by the auto-grader.

(c) (10 of 50 points) Please submit a video of the robot moving through the path.yaml and any code you used to create it to canvas to demonstrate the working simulation you developed in lab1. If your FK is not correct in the auto-grader, but your video still shows the robot moving through space, you will receive all of the points.

Reminders:

- Be sure to comment your code and use variable names and function names that clearly identify what you are trying to do.
- Put your name in a comment at the top of each file that you create.
- Remember, you are NOT ALLOWED to use PyKDL to complete this assignment. Feel free to use it as a check, but all mathematics in the code must use numpy. Any answers that use PyKDL will not be counted. (Yes, I use PyKDL in the lab1_utility functions, but this is not my first time doing these types of calculations)
