2.12: Introduction to Robotics
Lab 3:
6 DoF Kinematics[*]

Spring 2021

Assigned on: 4th March 2021 Due by: 10th March 2021

Instructions:

1. **You will be working on this lab with a partner during your lab section. Please make sure to attend!**
2. **Time permitting, a TA will verbally run through the questions attached to this handout during the lab section. Please think about the questions as you go through the lab. If we are constrained for time, we may ask you to upload your answers to Canvas.**

1 Introduction

This week, we'll be looking at the use of serial robot manipulators within 3D space. In contrast to the simple, 2-link robots which we were simulating last week, you'll learn how to model the kinematics of full six degree-of-freedom manipulators in MATLAB. The code that we'll be using for the class can be found on the GitHub: `https://github.com/mit212/lab3_2021`.

2 Rotations in 3D Space

In lecture 3, you were introduced to the concept of using rotation matrices to perform transformation within 3D space. Euler angles in particular are a common way to represent physical rotations. However, it's important to remember that matrix multiplication is not commutative - meaning that the order in which we perform the three euler rotations is critical!

To visualise this, you're going to perform two transformations in MATLAB and observe the results. Follow the instructions in the file `EulerTransforms.mlx` and answer the questions.

Question 1 *What is the orientation of the initial [n,t,b] coordinate frame relative to the world frame?*

Question 2 *What are the n,t, and b vectors for the final coordinate frames? Are they different? Why do these two transformations lead to different orientations? Do you know of any alternate ways that these transformations could be represented?*

[*]

1. Version 1 - 2021: Cormac O'Neill

3 Creating a Robot in MATLAB

In Matlab, serial robots are represented by rigidBodyTree objects. These objects represent the series of joints and links which make up the arm and can contain both revolute and prismatic joints. To introduce the basics of rigid body trees, you are going to create one for the 2.12 hardware lab robot.

To do this, look at the instructions laid out in this tutorial: `https://www.mathworks.com/help/robotics/ug/build-a-robot-step-by-step.html`. Instead of creating the 4-link robot shown, we will produce a rigidBodyTree object with the same parameters as the 2.12 robot we explored last week in a new MATLAB script. Namely:

- $L_1 = 0.1524\text{m}$,
- $L_2 = 0.1524\text{m}$,
- $q1_{upper} = 113.7^\circ$,
- $q1_{lower} = -113.7^\circ$,
- $q2_{upper} = 161^\circ$, and
- $q2_{lower} = -161^\circ$.

Look at the partially completed code for this in `RigidBodyTree_Student.mlx`. Fill in the missing sections of this code in order to successfully create your rigid body tree. You can set the joint limits in radians by modifying the PositionLimits property of a joint, which is default [-pi, pi]. That is, for joint q1 of the 2.12 robot, in MATLAB we would want to set:

```
q1.PositionLimits = [deg2rad(-113.7), deg2rad(113.7)]
```

Note that you will need 3 bodies in total: one body which is coincident with the base and connects to the first joint, a second body representing link 1, and a third body representing link 2.

Question 3 *After creating your rigid body tree robot, try calling the command* `show(robot)` *within MATLAB's command window, where 'robot' here is the name of your rigid body tree. You should see it plotted in 3D space. Now try calling* `show(robot, randomConfiguration(robot))`*. Do you think that the random configuration respects the joint limits we set previously? Present a screenshot of your plotted robot.*

4 Six DoF Robot in MATLAB

Rigid body trees are a powerful way to represent robots, especially as they become increasingly complicated. MATLAB also allows for 3D meshes to be attached to rigid body trees, giving us a way to visualize a model of our robot generated from something like a CAD file. MATLAB comes with a library of existing rigid body trees for a list of real world robots. You can see details on the robots available here: `https://www.mathworks.com/help/robotics/ref/loadrobot.html`.

If you were working on a custom robot, or on a robot which isn't built in to MATLAB, you can also choose to represent it in a standard file format known as the Unified Robot Design Format (URDF). URDF files also follow a similar structure to rigid body trees, and can be imported into MATLAB using the importrobot() function. The added advantage of URDF files are that they can read by programs outside of MATLAB, such as Gazebo. For now, we won't go into the details of how to create your own URDF files - but it's still important for you to know they exist!

4.1 Inverse Kinematics and Trajectory Planning

For the remainder of this lab, we're going to be using a 6dof robot that you'll encounter during the final project. In particular, we're going to be looking at the UR5, by Universal Robots, `https://www.universal-robots.com/products/ur5-robot/`.

The UR5 comes prepackaged within MATLAB, so we can easily load its rigid body tree model by calling `robot = loadrobot('universalUR5')` as shown on line 1 of `ur5_trajMatlab.mlx`. This loads MATLAB's built-in model for the UR5 into the variable `robot`. We're going to use this model while we learn how to find the inverse kinematics and simulate a simple trajectory within MATLAB.

Open up the file `ur5_trajMatlab.mlx` and read the through the code. You can see how the inverse kinematics in particular can be found numerically using the `inverseKinematics` function provided by MATLAB's Robotics Toolbox on lines 17-19 and line 29. The trajectory you'll be making will try to move each joint from its initial position, to the desired final position with zero final velocity.

Question 4 *Using the above code for the inverse kinematics, can you find the joint coordinates needed to position the end effector at a position of [0 0.3 0.1] with a ZYX Euler representation of [0 0 0]?*

Question 5 *Why is the path of the robot not a straight line? Investigate this by plotting the joint coordinates over time in their own plots for both trajectories. The joint positions for the joint-controlled trajectory are stored in stateJoint(:,1:numJoints) and their velocities are stored in stateJoint(:,numJoints+1:end). If you want to learn more about these two approaches, you can read more here:* `https://blogs.mathworks.com/racing-lounge/2019/11/06/robot-manipulator-trajectory/`

References

[1] H. Asada and J.-J. E. Slotine, *Robot analysis and control.* John Wiley and Sons, 1986.