ECE 383/MEMS 442/ECE 590 Lab 2: Forward and Inverse Kinematics

Due date: 10/2/2015

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. To submit, upload your Lab2a.ipynb, Lab2b.ipynb, and Lab2c.ipynb files to Sakai.

Problem A: Analytical inverse kinematics

In Lab 2a, you will see a 3R robot at its zero position, with joint axes ZYY. You will also see a point that denotes a target position that should be reached by the robot's end effector. Your job is to compute **all** the IK solutions of the robot so that the robot's end effector position lies at the start of this segment, and the orientation of the third link matches the orientation of the segment.

Your code will go into the function `lab2a(L1,L2,L3,point,angle)`. Please consult Chapter 6 of the book for hints about how to computer a solution. Code for analytic IK of a 2R manipulator, as covered in Lecture 7, is provided and can be used as part of your solution. Also, the `math.atan2(y,x)` function will be useful here.

Make sure your solution follows the designated formatting of the output pair. The GUI will draw your solutions, which will help you debug. Remember, you will typically see either 0, 2, or 4 solutions.

Problem B: Numerical IK with both position and orientation

Lab 2b shows a single configuration of the 6DOF UR5 robot and a target cylinder in 3D space. As the target position moves, the robot's configuration should be optimized to place the end effector closer to the target, and it should also be oriented toward the target's orientation (as though it were picking it up from the side).

We've chosen the end effector local position and axis for you, and it is your job to find a configuration that matches this to the target world position and axis. Complete the function `lab2b(robot, qseed, ee_link, ee_local_position, ee_local_axis, target, target_axis)` which should use the Klamp't inverse kinematics functions to compute the IK solution for this problem.

It should return a triple:

- `q`: the solution configuration
- `errpos`: The distance between the end effector's world position and the target position at the solution.
- `erraxis`: The distance between the end effector's world axis and the target axis at the solution.

[Note: the current implementation for finding `errpos` and `erraxis` is incorrect]

You will need to understand the `RobotModel` class, the `RobotModelLink` class, and the IK module functionality. Please consult the intro to Klamp't slides and the Python API documentation, as well as the Klamp't IK tutorial at http://motion.pratt.duke.edu/klampt/tutorial_ik.html and the documentation of the IK module in http://motion.pratt.duke.edu/klampt/pyklampt_docs/namespaceklampt_1_1model_1_1ik.html

You will notice that even if your implementation is correct, many IK solve calls will fail or find configurations with self-collision. Why? By uncommenting lines in the `solve_ik` function, see what happens when you start from a zero configuration, and what happens when you run a random-restart technique. How do each of these initializations affect the ability to find a solution? [Place your answers to these questions in the marked cell at the bottom of the file]

Problem C. Pick and place

In Lab2C, the UR5 is presented with multiple colored blocks and a box on a table, and the task is to place each block into the box. The pose of each block and the box are assumed known, and the local coordinates are centered at its center and aligned with the axes of the block with z pointing upward. The robot starts at a "home configuration", and will return there after every placement operation.

1. Implement the `find_pregrasp_config` function, which finds a configuration in which the gripper center point is moved to a point 6cm above the object, with the gripper's fingers oriented to point straight downward and to grasp on opposite flat faces of the block. This configuration should also be close to the home configuration so that a linear joint space interpolation does not collide with anything. When you click on the "Move to Pregrasp" button, the robot should move to the pregrasp configuration of the given object.
2. Implement the `lower_and_close` function, which finds a sequence of milestones in which the gripper lowers its center to be exactly around the object, and closes its fingers to the desired width. The width value is between 0 (fully closed) and 1 (fully open). This milestone sequence move straight downward in Cartesian space, with no orientation change, and with each milestone spaced no more than 1cm apart. When you click the "Grasp" button, the robot should move along this milestone sequence.
3. Implement the `lift_and_place` function, which finds a sequence of milestones in which the robot moves a grasped object off the table by 10cm, brings it to a spot over the box, and then opens the gripper. When you click the "Place" button, the robot should move along this milestone sequence.

Test your that your code is able to place all of the items in the box using a sequence of Home->Pregrasp->Grasp->Place cycles. [Note: the "simulator" doesn't actually use physics, so objects can freely penetrate or be picked up in strange orientations. Full credit will be granted only if the blocks are picked up when the gripper is centered on the block with fingers against opposing sides, and that there are no collisions except for the placed objects.].