CS498 Intelligent Robotics, Spring 2020
Homework 3: Inverse Kinematics and Grasping

Due date: 3/10/2020

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 answers, take written notes, or look up online solutions.

All coding will be done on the Klamp't Jupyter system. Setup is as follows:

1. Login and open up a terminal under the `Run` tab.
2. If this is your first time, run `git clone https://github.com/krishauser/cs498ir_s2020.git; cs498ir_s2020/setup.sh`. If this is not your first time, run `cs498ir_s2020/update.sh` to get the updated homework code. Go ahead and close the terminal tab, and return to the file browser. The homework code is in `cs498ir_s2020/HW2`.
3. When you run the notebooks, check the Kernel menu to make sure you are using the `cs498ir-virtualenv` kernel.
4. Make sure to periodically backup your code to your local machine. To do this, select the notebook in the file browser and choose `Download`. (We don't expect to have the server die, but Engineering IT won't make any guarantees!)

You may implement your calculations manually using Klampt, Numpy, or the elementary operations found in the Python math module, according to your personal preference.

To submit, upload your written answers to Gradescope by March 10 before 9:45am, and upload your programming answers in the form of Jupyter notebook files (*.ipynb) onto the class http://learn.illinois.edu site.

Written Problem 1: Coordinate Transformations for Manipulation

In this problem we will use the convention T_A to denote the transform of frame A relative to the world, T_A^B to denote the transform of A relative to frame B, $\boldsymbol{x}^A$ to denote the coordinates of a point expressed in the local coordinates of frame A, and $\boldsymbol{x}$ to denote its coordinates in the world frame.

Suppose a 6DOF industrial robot has a depth camera mounted on its 6th link. Let $T_i(q)$ denote the world transforms of its links at configuration q, as calculated by forward kinematics. Let T_C be the calibrated camera transformation, in world coordinates, when the robot is at the zero configuration.

A. Suppose the camera detects an object at coordinates $\boldsymbol{o}^C = \begin{bmatrix} x_O^C \\ y_O^C \\ z_O^C \end{bmatrix}$ in the local camera frame, while it is at configuration q. Give the symbolic expression for the world coordinates $\boldsymbol{o}$ of this point.

B. Now, the gripper needs to pick up the object. The center of the gripper is given by coordinates $\boldsymbol{g}^6$, expressed in the local coordinate frame of the 6th link. We would like to move to a new configuration q' so that the world coordinates of the gripper center matches the object position. Give the equation that needs to be solved in order for this to occur (but do not attempt to solve it, you do not have enough information.)

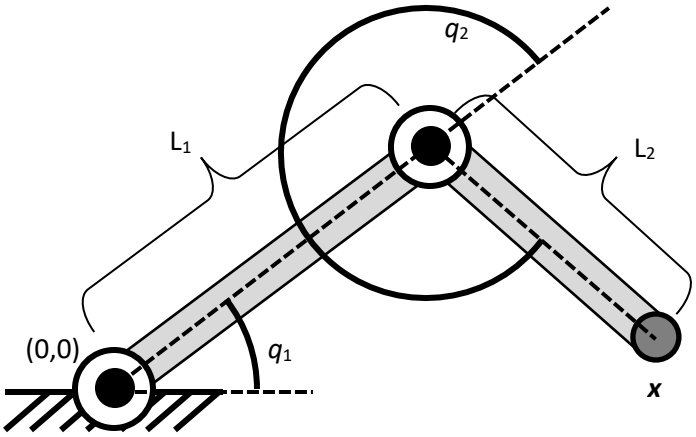
C. Recognizing that it is important to understand how the object is oriented in order to grasp it, the object detector has been improved to also provide the orientation of the object in the camera frame (this orientation maps some reference geometric features of the object onto the observed geometric features). Now, the detected object *pose* is a relative transform T_O^C in camera coordinates. Give the symbolic expression for the object transform in world coordinates.

D. The object is actually a coffee mug with a handle. To pick up the object, the gripper now must match the handle position and reach it at a given orientation. Relative to the object reference frame, the handle is known to be located at position $\boldsymbol{h}^O$, and the gripper needs to be oriented at orientation R_G^O to be able to grasp the handle. Now, give the equation that must be solved to determine q' so that the gripper center matches the handle world position *and* the gripper is oriented to pick up the object. (Again, do not attempt to solve it.)

Written Problem 2: Jacobians and Velocities

Consider the standard planar, 2R robot arm with link lengths L_1 and L_2 considered in class. In other words, the first joint lies at the origin, and the first joint angle q_1 measures the angle of the first link from the x-axis in the CCW direction. The second joint lies a distance L_1 from the first joint, translated along the x-axis of the first link. The joint angle q_2 measures the angle of the second link relative to the first link in the CCW direction. The robot's end effector lies a distance L_2 away from the second joint, translated along the x-axis of the second link.

(Ignore the possibility of joint limits and collisions.)



A. The forward kinematics of this manipulator, giving the end effector position & orientation, is

$$\boldsymbol{x} = f(\boldsymbol{q}) = \begin{bmatrix} c_1(c_2L_2 + L_1) - s_1s_2L_2 \\ s_1(c_2L_2 + L_1) + c_1s_2L_2 \end{bmatrix}.$$

Calculate the 2 x 2 Jacobian matrix $J(\boldsymbol{q}) = \frac{\partial f}{\partial \boldsymbol{q}}(\boldsymbol{q})$.

B. Suppose the robot's configuration is moving along a curve $\boldsymbol{q}(t)$. Write your forward kinematics map $\boldsymbol{x}(t) = f\big(\boldsymbol{q}(t)\big)$ as a function of t, and differentiate it using the chain rule. Verify that it equal to the product of the Jacobian and the joint velocity $\boldsymbol{q}'(t)$: $\boldsymbol{x}'(t) = J\big(\boldsymbol{q}(t)\big)\boldsymbol{q}'(t)$.

C. Draw the robot at configuration $q_1 = \pi/4$, $q_2 = \pi/2$. Compute the values of the Jacobian matrix $J\left(\begin{bmatrix}\pi/4 \\ \pi/2\end{bmatrix}\right)$. Illustrate on your drawing how each **column** of the Jacobian has a specific geometric significance.

D. Find a configuration $\boldsymbol{q}$ where the columns of the Jacobian become linearly dependent, i.e., the matrix drops rank, or the determinant becomes zero. Describe the significance of this situation in terms of the allowable motion of the robot's end effector.

Written Problem 3: Optimization with Line Search

Consider the problem of minimizing an error function $g(\boldsymbol{q})$ using an iterative method. The general pseudocode for doing so is as follows.

Input: function g, initial seed $\boldsymbol{q}_0$, maximum iterations N, step tolerance ϵ_x
Output: solution $\boldsymbol{q}$ and a termination reason

1. **For** $i = 0,1,2,\dots,N$:
    a. Determine a step direction $\Delta\boldsymbol{q}_i$
    b. Determine a step size α_i so that $g(\boldsymbol{q}_i + \alpha_i\Delta\boldsymbol{q}_i) \leq g(\boldsymbol{q}_i)$

c. **If** $\|\alpha\Delta\boldsymbol{q}_i\| < \epsilon_x$, **return** $\boldsymbol{q}_i$ and "converged"
d. Set $\boldsymbol{q}_{i+1} \leftarrow \boldsymbol{q}_i + \alpha_i\Delta\boldsymbol{q}_i$

2. Return $\boldsymbol{q}_N$ and "maximum iterations reached"

A. Gradient descent methods use the rule $\Delta\boldsymbol{q}_i = -\nabla g(\boldsymbol{q}_i)$. Under the conditions that the gradient is nonzero, g is smooth, and you take a small enough step in the negated gradient direction, this step is guaranteed to find a decrease in g. Based on this property, outline a method for determining a step size that is large when possible but small enough to ensure a decrease in g.
B. If the error function is the norm of a vector error $g(\boldsymbol{q}) = \|\boldsymbol{f}(\boldsymbol{q})\|$ with the IK problem solved at the root $\boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{0}$, the Newton-Raphson method for root-finding can be applied. Modify the pseudocode above to describe a Newton-Raphson method, such that a) it applies a line search to ensure that the g function decreases at each iteration, AND b) so that it terminates with "success" when the error decreases below a threshold ϵ_f.
C. Consider the IK problem of handling joint limit constraints $\boldsymbol{q}_{min} \leq \boldsymbol{q} \leq \boldsymbol{q}_{max}$ where these are all element-wise inequalities. How might you modify the above procedure so that the solution never exceeds the joint limits, but also makes progress toward reducing the IK error?

Written Problem 4: Formulating IK constraints

You have a numerical IK interface that accepts position, plane, axis, and rotation constraints on arbitrary links of a robot. Specifically, each constraint specifies the following parameters:

- Position constraint $(k, \boldsymbol{x}^k, \boldsymbol{x}_D)$ specifies $\boldsymbol{x}(q) \equiv T_k(q)\boldsymbol{x}^k = \boldsymbol{x}_D$.
- Plane constraint $(k, \boldsymbol{x}^k, \boldsymbol{n}_D, o_D)$ specifies a plane $\boldsymbol{n}_D^T\, x(q) \;=\; o_D$, or in other words, $\boldsymbol{n}_D^T T_k(q)\boldsymbol{x}^k = o_D$.
- Axis constraint $(k, \boldsymbol{a}^k, \boldsymbol{a}_D)$ specifies $\boldsymbol{a}(q) \equiv R_k(q)\boldsymbol{a}^k = \boldsymbol{a}_D$.
- Rotation constraint (k, R_D) specifies $R_k(q) = R_D$.

Here T_k is the transform of the k'th link and R_k is the rotation of the k'th link. An IK problem is defined by a list of constraint types and parameters. The IK solver will then solve all the equations specified for the given robot.

A. If you constrain two points $\boldsymbol{x}_1^k$, $\boldsymbol{x}_2^k$ on link k with two position constraints $\boldsymbol{x}_{D,1}, \boldsymbol{x}_{D,2}$, then the link still has one degree of freedom of movement, namely, rotating about the axis through the points. In other words, the stacked Jacobian of these two constraints has 6 rows, but only has rank 5. This may lead to some numerical issues, such as slower convergence. Describe how you could represent the same constraint on movement using one position constraint and one axis constraint.

B. A rotation constraint, if it were set on the 3x3 rotation matrix, would add 9 constraints to the IK problem. But as we have seen, the space of rotations is only inherently 3D. Adding redundant equations tends to reduce the solver's convergence speed. If you were designing an IK solver, how might you encode a rotation constraint without introducing redundant equations?

Programming Problem A: 3D Rotations

Lab 3a displays an interpolation of coordinate frames where rotations are represented by ZYX Euler angles. These, by convention, take on values in the range $[0,2\pi)x[-\pi/2,\pi/2]x[0,2\pi)$.

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)

Programming Problem B: Numerical IK with both position and orientation

Lab 3b 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.cs.illinois.edu/software/klampt/latest/pyklampt_docs/Manual-IK.html and the documentation of the IK module in http://motion.cs.illinois.edu/software/klampt/latest/pyklampt_docs/klampt.model.ik.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]

Programming Problem C. Pick and place

In Lab3c, 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.].