HW 3: Inverse Kinematics

Matthew Hong

Instructions

- Complete all the questions, *including* the "Resources Consulted" question. We expect that most students will use about a paragraph, or a few bullet points, to answer that question, but you can go longer or shorter.
- Submit the PDF *and* the code separately on Gradescope (look for "HW3: PDF" and "HW3: Code").
- For the PDF: To make things simpler for us to grade / inspect, please answer the questions in order (resources consulted first, then question one, then two, etc.). There is no page limit, but (as a high-level piece of advice) avoid writing excessively long-winded solutions. Use LaTeX for this; you can build upon this file. If you do that, you can remove the text that describes the actual questions if you want, please just make it *really easy* for us to see which question you are answering.
- For the code: some questions have code, which you will need to write. **Please use Python 3** for your code. Just submit everything as a single `.zip` file. Please include a brief README or brief comments in the code that make it *very easy* for us to run.

Resources Consulted

Question: Please describe which resources you used while working on the assignment. You do not need to cite anything directly part of the class (e.g., a lecture, the CSCI 545 course staff, or the readings from a particular lecture). Some examples of things that could be applicable to cite here are: (1) did you collaborate with a classmate; (2) did you use resources like Wikipedia, StackExchange, or Google Bard in any capacity; (3) did you use someone's code? When you write your answers, explain not only the resources you used but HOW you used them. If you believe you did not use anything worth citing, *you must still state that below in your answer* to get full credit.

Answer:
- scipy docs (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html) for definition of constraint input

In this assignment, you will compute numerically the inverse kinematics of a planar robot with $n = 3$ rotational joints. Specifically, our goal is to compute the joint-positions that bring the end-effector to a desired position $\mathbf{p_d}$. In this exercise we will ignore the orientation of the end-effector. We will formulate the problem as an optimization program as follows:

$$\begin{aligned} \underset{\mathbf{q}}{\text{minimize}} \quad & \|FK(\mathbf{q}) - \mathbf{p_d}\|^2 \\ \text{subject to} \quad & l_i \leq q_i \leq u_i,\ i = \{1, \ldots, n\}. \end{aligned}$$

The objective is to find the values of the joints $\mathbf{q}$ that minimize the difference between the actual position of the end-effector computed using the forward kinematics $FK(\mathbf{q})$, and the desired position $\mathbf{p_d}$, subject to the joint limits l_i, u_i.

1. First, implement the forward kinematics function in the file **fk.py**. The function should receive the joint values $\mathbf{q}$ and the lengths of the 3 links $\mathbf{L}$, and it should return the position of the robot's end-effector. Report the results for the following values. Note that $\mathbf{q}$ is in radians.

   (a) $\mathbf{q} = [0.0, 0.0, 0.0], \mathbf{L} = [1.0, 1.0, 1.0]$
   $[\mathbf{x}, \mathbf{y}, \mathbf{z}] = [3, 0, 0]$

   (b) $\mathbf{q} = [0.3, 0.4, 0.8], \mathbf{L} = [0.8, 0.5, 1.0]$
   $[\mathbf{x}, \mathbf{y}, \mathbf{z}] = [1.21742749, 1.55602, 0]$

   (c) $\mathbf{q} = [1.0, 0.0, 0.0], \mathbf{L} = [3.0, 1.0, 1.0]$
   $[\mathbf{x}, \mathbf{y}, \mathbf{z}] = [2.70151153, 4.20735492, 0]$

   Hint: Express the pose of the first link relative to the reference frame using a rigid transform with an elementary rotation about the z-axis, then the pose of the second link relative to the first link and so on up to the end-effector.

2. For the inverse kinematics computation, you will use the scipy.optimize package.[1] We will numerically compute a solution through the following function call:

   ```
   solution = minimize(objective, q0, method='SLSQP', bounds=bnds)
   ```

   `objective` refers to the objective function that the optimizer attempts to minimize. `q0` is the initial vector of values and `bounds` specify the range of allowable values. **The scipy.minimize function has problems handling type promotion, so make sure all your numerical values are of type np.float64**.

   We specify the following parameters for the IK problem:

   $$\mathbf{p_d} = [0.1, 1.33, 0.0], \mathbf{L} = [0.7, 1.0, 1.0], \mathbf{q_0} = [0, 0, 1.86], (l_i, u_i) = (-\pi, \pi)\ \forall i$$

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`

We provide the file **ik-a.py**, which includes also code for plotting. The initial configuration of the robot should show up as follows, with the base of robot in red and the desired end-effector position in green. To compute the inverse kinematics, fill in the `objective` function. Visualize the result with the given plotting function. Save your figure as `ik-a_solution.png` and add it to the PDF.

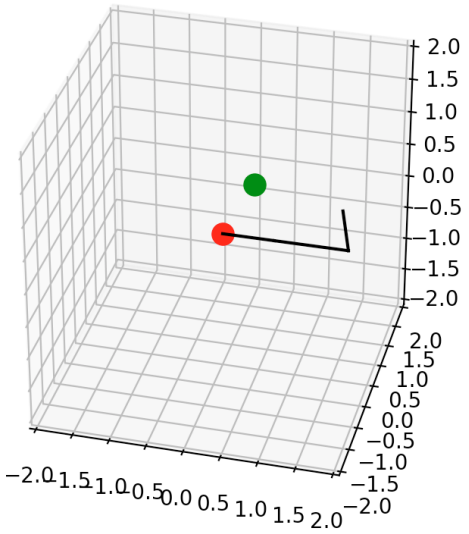

Figure 1: Example visualization. A solution state should result in the arm touching the green sphere.

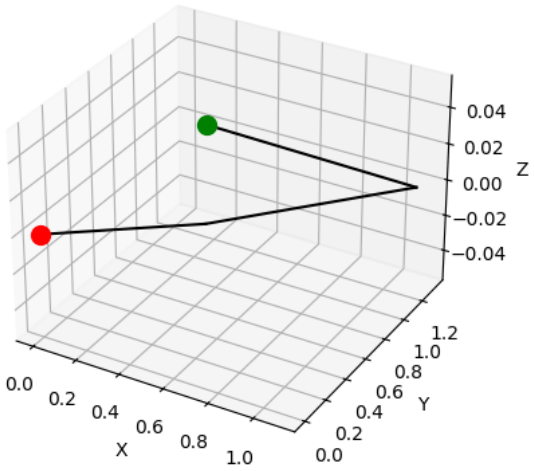

Figure 2: Result of inverse kinematics

3. You will now add an obstacle that the robot should avoid. You will approximate the obstacle with a sphere, with circle $\mathbf{c} = (c_x, c_y, c_z)$ and radius r. To detect whether the robot collides with the obstacle, the easiest way is to implement a line-sphere intersection algorithm[2]. If there is no real solution, then there is no intersection between a line and the sphere. In the provided file **collision.py**, fill in the function `line_sphere_intersection`. It should implement the algorithm and return the value under the square-root (the discriminant).

4. To ensure that the robot does not collide with the obstacle, we will add three obstacle collision constraints, one for each robot link. The constraints are provided as input to the optimization algorithm, as shown in the provided file **ik-b.py**. Fill in the code for the constraints, using the collision detection algorithm from the previous exercise. The parameters for the collision sphere are as follows:

$$\mathbf{c} = (0.6, 0.5, 0), r = 0.2$$

Note that if the constraints are satisfied, they should return a *non-negative* value. Using the following parameters for the obstacle, solve and visualize the solution. Save your visualization as `ik-b_solution.png` and add it to the PDF.

[2] `https://en.wikipedia.org/wiki/Line-sphere_intersection`

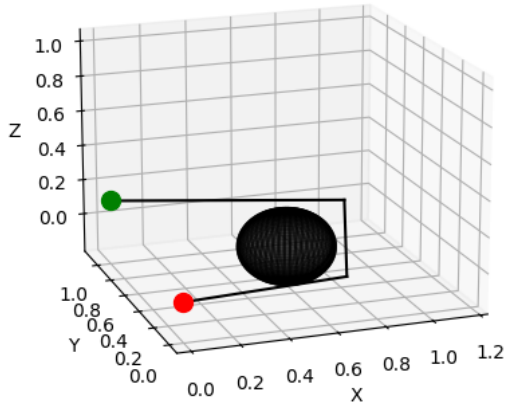

Figure 3: Inverse kinematics with collision constraints

5. Try increasing the radius of the obstacle r. What do you observe? Also experiment with different starting configurations $\mathbf{q_0}$. Add a figure to the PDF for at least one increased obstacle radius, and add another figure for at least one different starting configuration. Discuss the results in your PDF answers.

   Slowly increasing the radius, at a certain point, a solution does not exist for the given parameters (in the figure below, using a radius of, $r = 0.3$ results in an invalid solution).

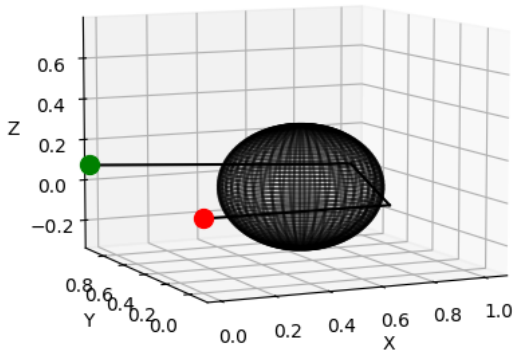

Figure 4: No solution exists with larger radius, $r = 0.3$

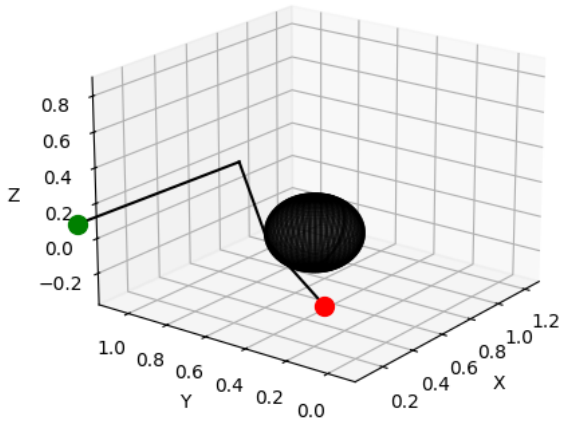

Figure 5: Solution with different starting configuration ($q_0 = [1, 0.3, 1.86]$)