

HW 3: Inverse Kinematics

Due Oct 27th 11:59pm (PST)

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:

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

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:

(a) $\mathbf{q} = [0.0, 0.0, 0.0], \mathbf{L} = [1.0, 1.0, 1.0]$

(b) $\mathbf{q} = [0.3, 0.4, 0.8], \mathbf{L} = [0.8, 0.5, 1.0]$

(c) $\mathbf{q} = [1.0, 0.0, 0.0], \mathbf{L} = [3.0, 1.0, 1.0]$

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)
```

¹<https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html>

objective refers to the objective function that the optimizer attempts to minimize. \mathbf{q}_0 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:

$$\mathbf{p}_d = [0.1, 1.33, 0.0], \mathbf{L} = [0.7, 1.0, 1.0], \mathbf{q}_0 = [0, 0, 1.86], (l_i, u_i) = (-\pi, \pi) \forall i$$

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, and save as `ik-a_solution.png`.

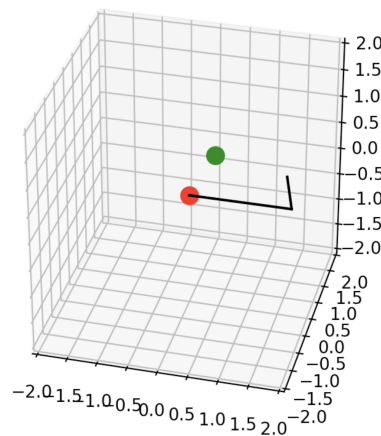


Figure 1: Example visualization. A solution state should result in the arm touching the green sphere.

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 algorithm². 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

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

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`.

5. Try increasing the radius of the obstacle r . What do you observe? Also experiment with different starting configurations \mathbf{q}_0 . Discuss the results in a file named `answers.pdf`.