Lab 3: Inverse Kinematics

Pre Lab Due Date: 10/19/2021 @ 11:59pm

Code + Report Due Date: 10/26/2021 @ 11:59pm

This assignment consists of both a written Prelab (due Oct 19 at 11:59pm) and the main portion of the lab (code and report). You will submit all parts of the assignment through Gradescope. Late submissions will be accepted per the standard late submission policy, but they will be penalized by 25% for each partial or full day late. After the late deadline, no further assignments may be submitted; post a private message on Piazza *before the submission deadline* to request an extension if you need one due to a special situation such as illness. The Prelab is worth 5 points and must be completed and submitted individually, and the report and code are each worth 20 points. You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. **If you make use of outside sources, you are expected to cite them in your report.** When you get stuck, post a question on Piazza or go to office hours!

Note: the main lab (like all past and future labs) is a partner assignment! You will complete and submit all Prelabs individually, but submitting the main lab individually is not allowed.

1 Prelab

The goal of the prelab is to connect ideas you are learning in class with the actual tasks you will work on in the projects. These assignments are "pen and paper" assignments in that you should not need to write any code, but if you would like to use digital tools to help explain your answers or improve your own understanding of the problems you are welcome to do so.

1. Local Minima of a Function $f(x)$

   (a) Consider the scalar-valued, single variable function $f(x) = x^4 - 6x^2$. Using techniques from elementary calculus, find all local minima of this function. Show your work, and report the minimizers $x^\star$ which achieve the minimal values $f(x^\star)$.

   (b) As part of you previous answer, you likely generated *another* simple polynomial equation in the form $g(x) = 0$ whose roots are candidate minimizers, and then used algebraic manipulations (e.g. factorization) to solve for these roots.

   Suppose now that $g(x)$ was instead a very complicated function. Even if this $g(x)$ could be approximated by a polynomial, it has been shown that explicit solutions for the roots of general polynomials of order 5 or greater do not exist. Given a starting point x_0, describe a procedure, *e.g.*, algorithm or numerical approach, to find an approximation for a nearby local minimum of $f(x)$ without explicitly solving for the roots of a function $g(x)$.

   Hint: consider phrasing your procedure as an algorithm which, given a current guess x_i, computes the next best guess x_{i+1}.

   (c) Explicitly state some conditions you have chosen, under which the procedure you described in your previous step should *terminate*, i.e. declare either that the current best guess is sufficiently accurate or that you should "give up".

2. Solutions to the Linear System $Ax = b$

   (a) Suppose your classmate has told you that a particular value x_0 solves the equation, i.e. $Ax_0 = b$. They ask you if there exist any ***other*** choices for x which also solve the equation. How would you determine the answer to their question? If such other solutions exist, how would you write a general solution which describes all valid solutions for x?

   Hint: recall the notion of the *nullspace* of a matrix.

   (b) Given two vectors $a, b \in \mathbb{R}^n$, how would you *project* the vector b onto a? That is, give an explicit formula for a vector c which is (anti)parallel to a and minimizes $||c - b||$, the norm of the error between the original and projected vectors.

Submit the Prelab on Gradescope. **You must complete and submit your prelab assignment individually**, and your writeup should be clear and concise.

2 Simulation Setup

To update your own private fork of the `meam520_labs` repo for Lab3 please do the following:

```
$ cd ~/meam520_ws/src/meam520_labs/
$ git pull upstream master
```

This is going to look at the public TA repository and grab the code stubs and launch files that we've added since the last lab. Occasionally when running this command, you might encounter a merge conflict. We don't expect that to occur, but if it does, don't hesitate to reach out to the TAs for help.

3 Inverse Kinematics

In this lab, you will implement an inverse kinematics solver for the Panda arm which can determine a set of joint angles which place the end effector at a desired position and orientation in full 3D space.

3.1 Exact Solutions

For many robot arms, even in 3D, it is relatively easy to compute exact (or *analytical*) inverse kinematics solutions like you did for the planar RRR arm in Lab 1. As you discussed in the analysis section for that lab, this is particularly easy when the robot posseses a *spherical wrist*, so that the rotation and translation inverse kinematics can be decoupled. It is also most straightforward when the *task space* (e.g. the position and orientation space $SE(3) = \mathbb{R}^3 \times SO(3)$) has the same dimension as the *configuration space* (i.e. the joint angle space), since in that case, almost all end effector poses will have a finite number of inverse kinematics solutions.

The Panda does not have a spherical wrist, and also has 7 degrees of freedom. We could consider temporarily fixing the last joint in the arm and solving the IK problem for the remaining six joints, so that all solutions could be found by varying the fixed value of that last joint. The offsets between the axes of the arm and the absence of kinematic decoupling mean that geometric, intuitive IK solutions are hard to come by, even though research has shown that all solutions for the inverse kinematics of 6-DOF arms can be computed through complex algebraic and trigonometric manipulations.

3.2 Numerical Optimization

Instead, in this lab we will compute inverse kinematics solutions using optimization: namely, making an *initial guess* (even a *very* poor one) and then adjusting our guess in a direction which will reduce the error between the desired output and the output of the current guess. This is a *gradient descent* method, since we are (roughly speaking) following the gradient of an error term (also called a *cost function*) to find a point which is a local minimum, i.e. continuing to move in any direction would only increase the cost or error. When our best guess lies within a certain numerical tolerance of a minimum, we terminate the optimization.

We will also need a way to choose a particular solution out of the infinite inverse kinematics solutions which exist for the 7-DOF arm. We will consider an additional cost to optimize that encourages solutions which are far away from the joint limits of the robot.

4 Implementation

You will need to implement your solution in `solveIK.py`. Open the `lib` folder using your preferred text editor, for example:

```
atom ~/meam520_ws/src/meam520_labs/lib/
```

You will implement all missing functionality in the `IK` class in `solveIK.py`. You should examine and finish implementing the functions in the below order.

IMPORTANT NOTE: this lab will make use of your `calculateFK`, `calcJacobian`, and `IK_velocity` implementations from previous labs. If you did not receive full credit for your code on Gradescope, it's vital that you work with the teaching team to correct any issues so they do not propagate to this lab. Please come to office hours or make a detailed post on Piazza, and we will be more than happy to help you sort out any issues!

4.1 Helper Functions

These are small geometric subroutines which will be used by your higher level optimizer functions.

`IK.displacement_and_axis(...)`

This function will compute two quantities:

1. `displacement`: The displacement vector from the origin of the current frame to the target frame.
2. `axis`: A vector pointing along the axis of rotation from the current frame to the target frame, with magnitude $\sin\theta$, where θ is the angle of rotation from the current frame to the target frame.

While the displacement vector is straightforward to compute, the axis vector is more involved. You may find it useful to know that given a single rotation R, the corresponding axis of rotation with magnitude $\sin\theta$ can be found by first computing the skew symmetric part of R as

$$S = \frac{1}{2}(R - R^T)$$

and then extracting the coefficients of the corresponding vector under the skew operation. Explicitly, since S is skew symmetric, $S = [a]_\times$ for some a, where $a \times b = [a]_\times b$ and $\times$ is the cross product in $\mathbb{R}^3$. Since

$$[a]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

one can easily extract the vector coefficients, such that a points in the direction of the axis of rotation and has magnitude $\sin\theta$. What remains is to ensure you are considering the *relative rotation between the target and current frames*, but to ultimately **express the axis relative to the world frame** to comply with the function's defined behavior.

Hint: pay close attention to your signs and coordinate frames in this function!

`IK.distance_and_angle(...)`

This function will compute two quantities:

1. `distance`: The distance (in meters) from the origin of the current frame to the target frame.
2. `angle`: The magnitude of the angle of rotation between the current frame and the target frame.

You may find it useful to know that the magnitude of the angle of rotation[1] of a rotation matrix R can be computed using the trace, as

$$|\theta| = \arccos\left(\frac{tr(R) - 1}{2}\right) \tag{1}$$

Note: the Python `acos` function will throw an error if passed a value outside the range $[-1, 1]$. Since your numerical computations may have some noise, you may wish to constrain its inputs to lie within this range.

`IK.is_valid_solution(...)`

This function will be used to check whether the joint configuration your optimizer spits out is actually a valid solution to the inverse kinematics problem. Specifically, this function should return `True` if and only if all the following conditions hold:

1. The joint angles are within the joint limits
2. The distance between the achieved and target end effector positions is less than `linear_tol`
3. The magnitude of the angle between the achieved and target end effector orientations is less than `angular_tol`

Remember: the `IK` class stores the joint limits as class variables and the tolerances as instance variables.

4.2 Task Functions

In our IK solver, we consider two *tasks* for the robot, which will later be composed hierarchically into one policy. Each of the below task functions computes a desired joint velocity which will achieve that task **when considered independently**, i.e. in the absence of the other task.

[1] You might wonder what specifically we mean by the angle of rotation of a rotation in 3D, since angles are planar objects. Fortunately, any rotation matrix R can be decomposed into two quantities: an axis vector (not necessarily x, y, or z) and an angle of rotation around that axis, which together produce that rotation matrix. This is known as the *axis-angle* representation of rotations.

`IK.end_effector_task(...)`

This function computes a joint velocity which will achieve the *primary task* of the IK solver, namely to reduce the error between the current and target end effector poses.

Consider that if we compute the `displacement` and `axis` of rotation from our current frame to the target frame using `IK.displacement_and_axis(...)`, then these vectors point in the direction our end effector needs to move to close the gap between its current pose and the desired pose. *(Remember that the angular velocity of a body is essentially just the instantaneous axis of rotation of that body, so the same thinking applies!)* Furthermore, because the `displacement` and `axis` both decay to zero magnitude as we get closer and closer to the goal, as we approach the desired end effector pose, the amount we want to move will also go to zero.

Your task is to use the current and target poses to compute a desired joint velocity `dq` such that:

1. the corresponding linear velocity is equal to the `displacement` from the current frame to the target frame
2. the corresponding angular velocity is equal to the `axis` from the current frame to the target frame

Hint: by reusing your Lab 1 and Lab 2 solutions, this should be a three-line function!

`IK.joint_centering_task(...)`

This function has already been fully implemented for you - read through it, but no need to edit it!

This function computes a joint velocity which will achieve the *secondary task* of the IK solver, namely to move the joints towards the center of their range of motion. The joint offsets are scaled so that all errors are weighted proportionate to their total range of motion. This secondary task will help the solver converge to solutions which respect the joint limits of the robot, making the solutions useful in practice. This task will compute joint velocity of all zeros only when the joints lie exactly at the center of their range of motion. Otherwise, the velocity will be proportional to the offset from the center, pointing toward the center of the range.

4.3 Iterative Optimization Function

`IK.inverse(...)`

This function is the heart of the IK solver! It will take a `seed` (or initial guess), and from there try to achieve the two tasks (minimizing pose error and centering the joints).

The scaffolding of this function has already been written for you. At each iteration of the loop, joint velocities which achieve each of the two tasks **in isolation** are computed using your task functions. You are left to implement two aspects of the loop:

1. **Task Prioritization**: Clearly, the robot cannot simultaneously achieve the desired end effector pose (a task with 6 DOF) while also centering all of the joints (a task with 7 DOF) since $6+7 = 13 > 7$, the number of joints! For this reason, we adopt a *task hierarchy*, through which we require that the robot "does its best" to achieve a secondary task (joint centering) but **without degrading its performance on the primary task in any way.** Explicitly, the idea is that the joint velocities `dq` that are ultimately chosen should perfectly achieve the desired end effector velocities. But, since the robot has 7 joints while this task specifies only 6 quantities, it is free to move along this remaining degree of freedom in whatever way will most closely approximate performance of the secondary task.

   Hint: Think back to the Prelab, and what you learned working on inverse velocity kinematics. Suppose we start by just letting `dq = dq_ik`. What additional terms added to `dq` would not affect the end effector velocities? How can we approximate the performance of the 7-DOF secondary task with only one remaining degree of freedom?

2. **Termination Conditions:** The optimization must eventually decide it has found a solution or that it should give up. Your function should stop iterating when either of two conditions is met:

   (a) The `while` loop has performed `max_steps` iterations

   (b) The norm of the chosen step `dq` is smaller than `min_step_size`

   After terminating, the function will check your solution using your implementation of `IK.is_valid_solution` and return your solution, the `success` boolean, and a history of the joint angles at each iteration of the optimizer.

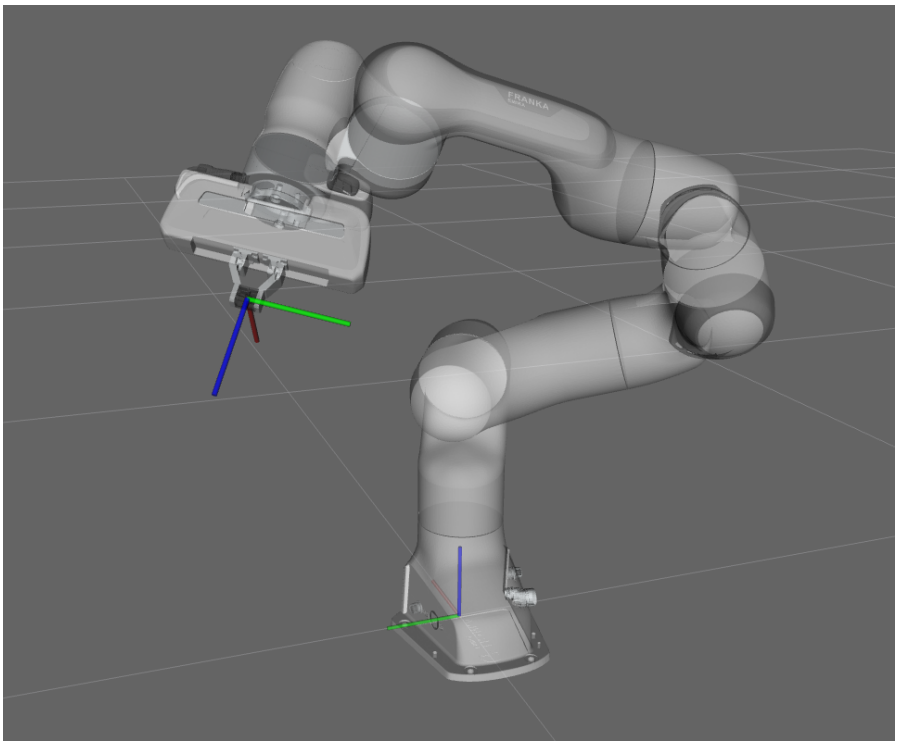

Figure 1: The visualization generated by the `visualize.py` script, which will iterate through a list of target end effector poses and run your code to find an IK solution.

5 Testing

We recommend that you test your solution separately from the simulator first. Examine the bottom of `solveIK.py` to find a simple testing environment. Running this script from the terminal, i.e.,

```
cd ~/meam520_ws/src/meam520_labs/lib/
python solveIK.py
```

will run your solver for a single desired end effector pose. The progression of your optimizer will be printed out after the iteration terminates.

You should perform more extensive testing yourself by adding to the tests in `solveIK.py`. Remember, it should be very easy to certify that a solution is correct by simply passing it back through your forward kinematics implementation! Indeed, your IK solver will need to do this when terminating the optimization to decide whether the current best guess is acceptable as a successful solution.

As usual, we've provided a visualization tool which will help you test your solution. First, launch the simulator with

```
roslaunch meam520_labs lab3.launch
```

and then run the test script via

```
cd ~/meam520_ws/src/meam520_labs/labs/lab3
python visualize.py
```

This script will step through a list of target poses and run your solution against each, going to the joint configuration returned by successful solves.

6 Report

You will submit a written report describing your work in this lab. **The report must be no more than 8 pages.** You are free to use whatever typesetting workflow you are comfortable with, but we highly recommend using LaTeX. Handwritten submissions will not be accepted. If you need to sketch some diagrams by hand, this is okay. Your report should address all of the tasks listed in this section.

6.1 Methods

Describe the approach that your inverse kinematics solver implements. Fully discuss all aspects of the solution, including a description of the components of the `IK` class that you did not implement. For these, a "black box" perspective is sufficient, i.e. describe their functionality from an input-output only perspective.

6.2 Evaluation

You should extensively test your solutions to ensure their correctness. In this section, document your testing process: how can you be confident that your solutions are free of bugs? In addition, include the following:

New Poses

List three new poses on which you tested your solver, including screenshots of the robot executing the corresponding IK solution.

Performance Statistics

Write a loop which runs your solver on at least ten different poses (even more would be better!) *different from those provided in the visualize script*. Report

1. **mean**, **median**, and **maximum** time elapsed (consider using `time.perf_counter`)
2. **mean**, **median**, and **maximum** number of iterations performed
3. **the rate of success** (how often you found a solution which returns `success = True`)

Feel free to include any other statistics you find meaningful.

6.3 Analysis

In this section, you will analyze the performance of your solutions and consider some extensions. Your report should comment on each of the tasks below.

Uniqueness of Solutions

Try testing your IK solver using different seeds i.e. the initial guess provided to `IK.inverse(...)`. Do you always yield the same solution? Justify your findings.

Algorithmic Completeness

While testing, you may have encountered some target end effector poses for which your algorithm fails to find a solution. Does this mean that the robot is unable to achieve that end effector pose? Does it mean you have coded your algorithm incorrectly? Please explain.

Warm Start

You may have noticed that an approach using numerical optimization tends to be much slower than an analytical approach. Suppose you wanted to compute IK solutions for many different poses, but these poses were all clustered rather close together in the reachable workspace with similar orientations (e.g. imagine picking up many objects off a small table near the robot). How might you be able to decrease the total runtime/iterations necessary to compute IK solutions for all these poses? Explain your reasoning.

7 Submission via Gradescope

- **Group Submission:** Each group should only submit once to Gradescope per Gradescope assignment. While going through the submission please ensure that you **add your group members to the submission**. Failure to do so could impact the grades of all parties involved.
- Please submit `solveIK.py`, and any other files needed to run your code to the Lab 3 Code on Gradescope (including those from previous labs which you rely on). Once submitted, we will run automated tests to grade your code. You may submit as many times as you want, but any attempts to try to figure out what test cases we are running will result in your code getting a 0.
- Please submit your written report as a PDF to the Lab3 Report assignment on Gradescope.

IMPORTANT, please submit the correct thing to each assignment. Don't submit the code to the report assignment. Don't submit the report to the code assignment.