Intelligent Systems and Robotics 2024: Homework1

The goal of homework1 is to practice and implement what you have learned in the recent courses. In the following exercises, you will implement your own **inverse kinematics(IK)** algorithm step by step and test the algorithm in simulation using a **Franka Panda 7DoF** arm.

Exercise 0

Let's take a look at what we have now. You will get:

1. `core.py` : this file is a template **where you need to write your own code**.
2. `robot_env.py`: build a simulation environment. **This file does not need to be modified**, but you can call some functions provided for you.
3. `demo.py`: this file provides some demonstrations to show whether your algorithms work well in the simulation by calling functions in `robot_env.py` and `core.py`.
4. Folder `/robot_model`: contains the URDF files and its meshes. **Do not change anything in this folder.**

Then you need to install the dependencies. Python version 3.7 or 3.8 is recommended. It is recommended to use *conda* and virtual environment to configure the programming environment.

The third-party libraries you need are: *pybullet* and *numpy*. For *numpy* version, I am using version 1.20.1. New versions and some slightly older versions are also available.

For pybullet, just run:

```
pip install pybullet
```

Exercise 1: Inverse Kinematics 【50% score】

Now it is time to perform your own **inverse kinematics(IK)** algorithm. Here the functions you need to implement in the template:

NOTE1: For specific implementation details, you can refer to the inverse kinematics algorithm in **the latest slide of lecture 2**.

NOTE2: the maximum number of iterations has been set as a variable `MAX_NUM_ITER`.

1. `thetalist,success = InverseKinematics_in_space(Blist,M,T,thetalist0,eomg,ev):` Computes inverse kinematics in the space frame for an open chain robot.
    - input1: Blist: The joint screw axes in the end-effector frame when the manipulator is at the home position.

      input2: M: The home configuration of the end-effector.

      input3: T: The desired end-effector configuration T_{sd}.

      input4: thetalist0: An initial guess $\theta_0 \in R^n$ that is “close” to satisfying $T(\theta_0) = T_{sd}$.

      input5: eomg: A small positive tolerance on the end-effector orientation error. The returned joint variables must give an end-effector orientation error less than ϵ_ω.

input6: ev: A small positive tolerance on the end-effector linear position error. The returned joint variables must give an end-effector position error less than ϵ_v.

- return1: thetalist: Joint variables that achieve T within the specified tolerances.

  return2: A logical value(bool) where *TRUE* means that the function found a solution and *FALSE* means that it ran through the set number of maximum iterations without finding a solution within the tolerances ϵ_ω and ϵ_v.

Exercise 2: Play with the demo【50% score】

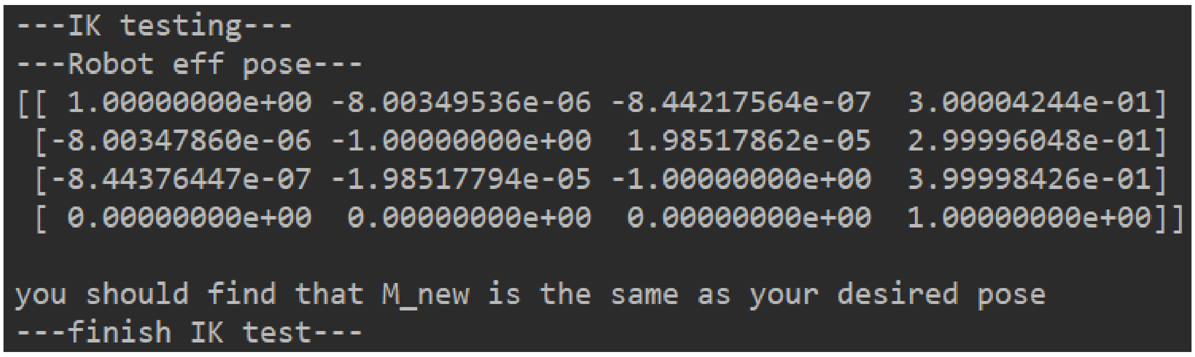
It's time to play with a robot arm and see what your function can do! (in simulation)

1. Set `TESTING` variable in `robot_env.py` as: `"IK"`, then run the code in `demo.py`, especially running function: `IK_test(robot, physicsClientId)`

   In this demo, you will set the target configuration and use the **inverse kinematics** algorithm to solve the corresponding joint angle, then control the joint angle of the robot and check whether the final position of the end effector coincides with the target position.

   Again, you can press the CTRL and left mouse button and drag the mouse to change the view angle. You can also zoom in or out by sliding the mouse wheel.

   If your IK algorithm is correct, you will find something like this:



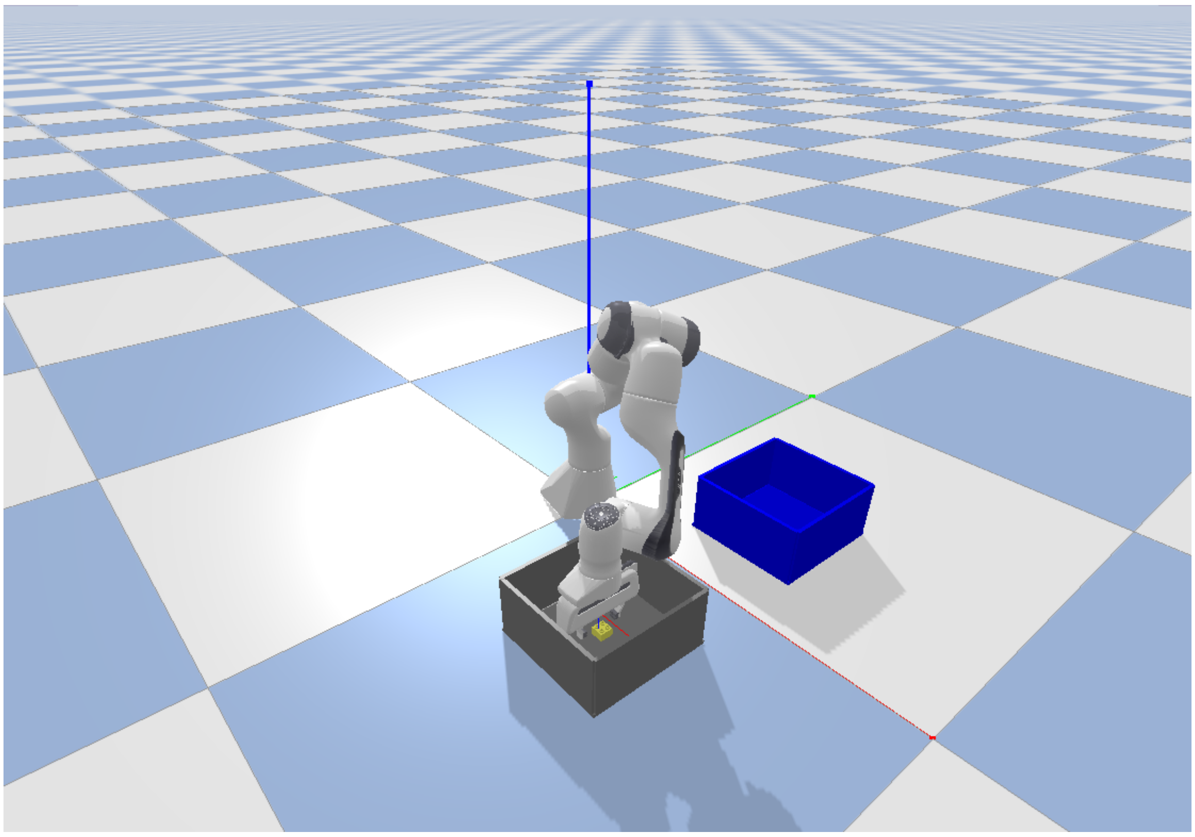
```
---IK testing---
---Robot eff pose---
[[ 1.00000000e+00 -8.00349536e-06 -8.44217564e-07  3.00004244e-01]
 [-8.00347860e-06 -1.00000000e+00  1.98517862e-05  2.99996048e-01]
 [-8.44376447e-07 -1.98517794e-05 -1.00000000e+00  3.99998426e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

you should find that M_new is the same as your desired pose
---finish IK test---
```

2. Set `TESTING` variable as: `"PICK_PLACE"`, then run the code in `demo.py`, especially running function: `pick_and_place_demo(robot, physicsClientId)`

   In this pick and place demo, you will solve a pick and place problem and your goal is to move a yellow LEGO brick from the gray box to the blue box. **You need to define your own waypoints,** you will use your own IK algorithm to solve the corresponding joint angle for each way point and then control the robot to reach that joint angle. You can refer to the `IK_test` to know how to compute IK and send commands to the robotic arm.

   If you implement the IK algorithm correctly, you will see that the robot arm successfully grabs Lego blocks from the gray box to the blue box.



Hope you can have some fun from this pick and place demo XD

This is the end of homework1~

Additional Note on Submission: Very IMPORTANT!

1. **How to submit your homework**

   Record the video of the pick and place demo and name the video as student ID, such `2021211276.mp4`.

   You need to submit your `demo.py` and `core.py`. You don't have to rename these 2 python files.

   Please zip all the files that required to submit and name the zip pack as your student ID, such as `2021211276.zip`. Submit this zip pack to *web learning*(网络学堂).

   ```
   【NOTE】Do not modify the function name!!!
   ```

2. More about pybullet:

   The simulation is base on *pybullet*, you can find more information here: pybullet

3. Problem Shooting:

   If you have any problems/bugs on the homework, please contact me(TA - Xiang Zhu, Chengming Shi) via *WeChat group* or email: `zhuxiang24@mails.tsinghua.edu.cn`, `scm23@mails.tsinghua.edu.cn` or leave a message on *web learning*(网络学堂).