Introduction to Robotics: Homework2

The goal of homework2 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

Just like homework1, let's first take a look at what we have now. You should have already installed **pybullet** and **numpy**. You will get:

1. `core.py` : this file is a template **where you need to write your own code**.

   【IMPORTANT】**You NEED to copy all of the implemented functions in `core.py` in last homework(homework1) to `core.py` of homework2.**

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. `demo2.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.**

Exercise 1: Velocity Kinematics 【50% score】

In this part, you need to implement two functions to calculate the Jacobian.

NOTE1: write your code in the template file `core.py`. **DO NOT MODIFY THE FUNCTION NAME!!!** Otherwise the autotest program will not work and you will lose your score :-)

NOTE2: *The input and output of the function are numpy arrays!*

Here is what you need to finish:

1. `Jb = jacobian_body(Blist,thetalist)` : Computes the body Jacobian 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: thetalist: A list of joint coordinate values.
   - return: The corresponding body Jacobian.
2. `Js = jacobian_space(Slist,thetalist)` : Computes the space Jacobian for an open chain robot.
   - input1: Slist: The joint screw axes in the space frame when the manipulator is at the home position.

     input2: thetalist: A list of joint coordinate values.
   - return: The corresponding space Jacobian.

That's the end of exercise1. Less functions need to implement compare to homework1, right? :)

Exercise 2: Inverse Kinematics 【50% score】

Now it is time to use the code in exercise 1 and previous code in homework1 to perform your own **inverse kinematics(IK)** algorithm. Here are the functions you need to implement in the template:

NOTE1: use an iterative Newton-Raphson root-finding method starting from the initial guess thetalist0.

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

1. `thetalist,success = IK_in_body(Blist,M,T,thetalist0,eomg,ev):` Computes inverse kinematics in the body 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.
2. `thetalist,success = IK_in_space(Slist,M,T,thetalist0,eomg,ev)`: Computes inverse kinematics in the space frame for an open chain robot.
    - Equivalent to `IK_in_body`, except the joint screw axes are specified in the space frame.

Exercise 3: Play with the demo

Congratulations! You have completed your coding part. Just like homework1, 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:

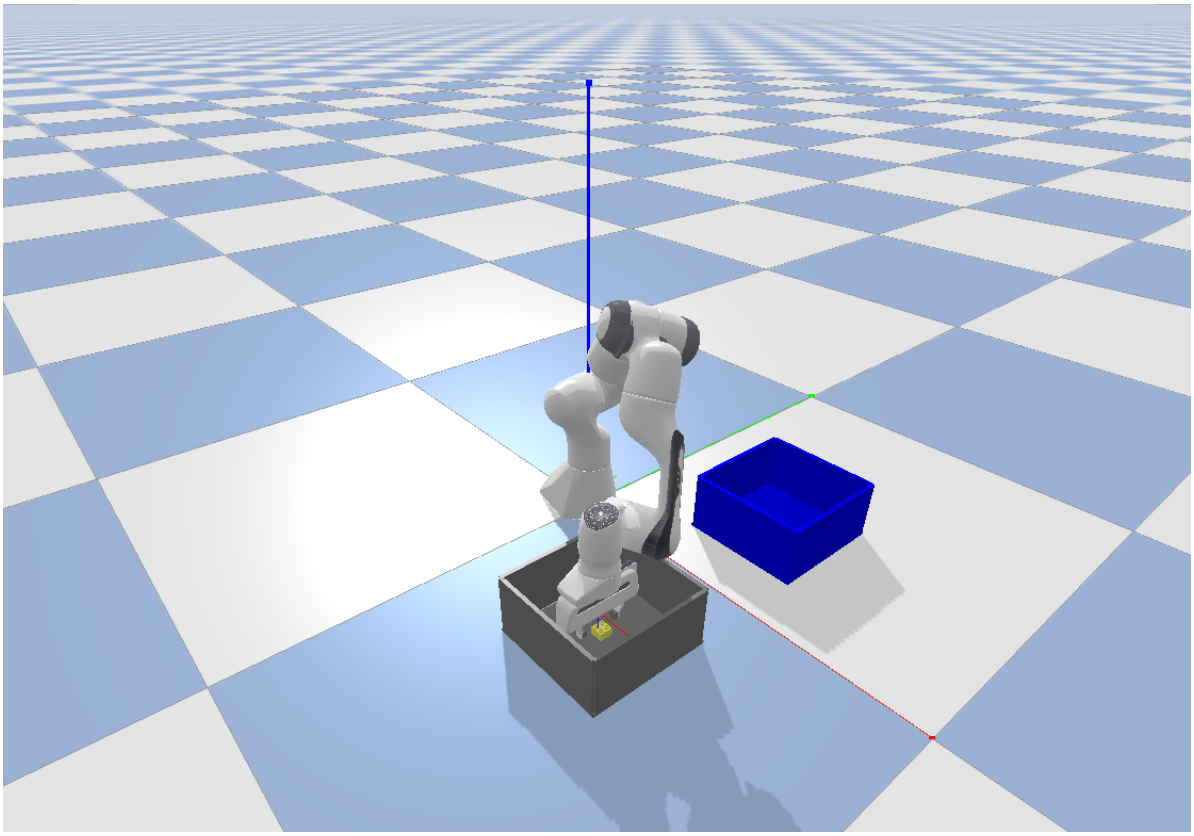
```
---IK testing---
---Robot eff pose---
[[ 1.00000000e+00 -8.00349536e-06 -8.44217564e-07  3.00004244e-01]
 [-8.00347860e-06 -1.00000000e+00  1.98517862e-05  2.99996048e-01]
 [-8.44376447e-07 -1.98517794e-05 -1.00000000e+00  3.99998426e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

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. The way points are already given, 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.

   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 homework2~

Additional Note

1. **How to submit your homework**

   Make a copy of your `core.py` and name the copy as your student ID, such as `2021211276.py`. And submit this copy to *web learning*(网络学堂)

   【WARNING1】Do not modify the function name!!! Otherwise the autotest program will not work and you will lose your score :-)

【WARNING2】You NEED to copy all of the implemented functions in `core.py` in last homework(homework1) to `core.py` of homework2, and then submit the copy.

2. Again, in the template `core.py`, each function provides an input example and a corresponding output, you can use them to check the code.
3. More about pybullet:

   The simulation is base on *pybullet*, you can find more information here: pybullet
4. If you have any problems/bugs on the homework, please contact me via *WeChat group* or email: `zhu-x21@mails.tsinghua.edu.cn` or leave a message on *web learning*(网络学堂).