Introduction to Robotics: 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 **forward kinematics(FK)** 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_sim.py`: build a simulation environment. **This file does not need to be modified**, but you can call some functions provided for you.
3. `demo1.py`: this file provides some demonstrations to show whether your algorithms work well in the simulation by calling functions in `robot_sim.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: Rigid-Body Motions 【60% score】

You need to implement some basics functions of rigid-body motions.

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. `invR=rot_inv(R)`: Inverts a rotation matrix, use the property of rotation matrix.
   - input: R: Rotation matrix R
   - return: The inverse of R
2. `so3mat=vec_to_so3(omg)`: Converts a 3-vector to an so(3) representation.
   - input: omg: A 3-vector
   - return: The corresponding 3×3 skew-symmetric matrix in so(3).
3. `omg=so3_to_vec(so3mat)`: Converts an so(3) representation to a 3-vector.
   - input: so3mat: A 3×3 skew-symmetric matrix (an element of so(3)).
   - return: The corresponding 3-vector.
4. `[omghat,theta]=axis_ang3(expc3)`: Converts a 3-vector of exponential coordinates for rotation into axis-angle form
   - input: expc3: A 3-vector of exponential coordinates for rotation $\hat{\omega}\theta$

- return1: omghat: The corresponding unit rotation axis $\hat{\omega}$

  return2: theta: The corresponding rotation angle θ

5. `R=matrix_exp3(so3mat)`: Computes the matrix exponential of a matrix in so(3)
   - input: so3mat: An so(3) representation of exponential coordinates for rotation, $\hat{\omega}\theta$.
   - return: The $R \in SO(3)$ that is achieved by rotating about $\hat{\omega}$ by θ from an initial orientation $R = I$.
6. `R=matrix_log3(R)`: Computes the matrix logarithm of a rotation matrix
   - input: R: Rotation matrix.
   - return: so3mat: The corresponding so(3) representation of exponential coordinates.
7. `T=rp_to_trans(R,p)`: Converts a rotation matrix and a position vector into homogeneous transformation matrix.
   - input1: R: Rotation matrix.

     input2: p: A 3-vector
   - return: T: The corresponding homogeneous transformation matrix $T \in SE(3)$.
8. `[R,p]=TransToRp(T)`: Converts a homogeneous transformation matrix into a rotation matrix and position vector.
   - input: T: Transformation matrix.
   - return1: R: The corresponding rotation matrix.

     return2: p: The corresponding position.
9. `invT = trans_inv(T)`: Inverts a homogeneous transformation matrix.
   - input: T: Transformation matrix.
   - return: invT: Inverse of T.
10. `se3mat=vec_to_se3(V)`: Converts a spatial velocity vector into a 4x4 matrix in se3.
    - input: V: A 6-vector (e.g. representing a twist).
    - return: se3mat: The corresponding 4 × 4 se(3) matrix.
11. `V = se3_to_vec(se3mat)`: Converts an se3 matrix into a spatial velocity vector.
    - input: se3mat: A 4 × 4 se(3) matrix.
    - return: V: The corresponding 6-vector.
12. `AdT = adjoint(T)`: Computes the adjoint representation of a homogeneous transformation matrix.
    - input: T: Transformation matrix.
    - return: AdT: The corresponding 6 × 6 adjoint representation $[Ad_T]$.
13. `S=screw_to_axis(q, s, h):` Takes a parametric description of a screw axis and converts it to a normalized screw axis.
    - input1: q: A point $q \in R^3$ lying on the screw axis.

      input2: An unit vector $\hat{s} \in R^3$ in the direction of the screw axis.

      input3: The pitch $h \in R$ (linear velocity divided by angular velocity) of the screw axis.
    - return: S: The corresponding normalized screw axis $S = (\omega, v)$.
14. `[S,theta] = axis_ang6(expc6):` Converts a 6-vector of exponential coordinates into screw axis-angle form.
    - input: expc6: A 6-vector of exponential coordinates for rigid-body motion, $S\theta$
    - return1: S: The corresponding normalized screw axis S.

      return2: theta: The distance traveled along/about S.

15. `T=matrix_exp6(se3mat)`: Computes the matrix exponential of an se3 representation of exponential coordinates.
    - input: se3mat: An se(3) representation of exponential coordinates for rigid-body motion
    - return: T: The $T \in SE(3)$ that is achieved by traveling along/about the screw axis S a distance θ from an initial configuration $T = I$.
16. `se3mat=matrix_log6(T)`: Computes the matrix logarithm of a homogeneous transformation matrix.
    - input: T: Transformation matrix.
    - return: se3mat: The corresponding se(3) representation of exponential coordinates.

That's the end of exercise1. You may find this list is a bit long, but these implementations are quite simple. So, don't worry :)

Exercise 2: Forward Kinematics 【40% score】

Now is time to use the code in exercise 1 to perform your own **forward kinematics(FK)** algorithm. Here are the functions you need to implement in the template:

1. `T=FK_in_body(M, Blist, thetalist):` Computes forward kinematics in the body frame for an open chain robot.
    - input1: M: The home configuration of the end-effector.

      input2: Blist: The joint screw axes in the end-effector frame when the manipulator is at the home position.

      input3: thetalist: A list of joint coordinate values.
    - return: The $T \in SE(3)$ representing the end-effector frame when the joints are at the specified coordinates.
2. `T=FK_in_space(M, Slist, thetalist)`: Computes forward kinematics in the space frame for an open chain robot.
    - input1:M: The home configuration of the end-effector.

      input2: Slist: The joint screw axes in the space frame when the manipulator is at the home position.

      input3: thetalist: A list of joint coordinate values.
    - return: The $T \in SE(3)$ representing the end-effector frame when the joints are at the specified coordinates.

Exercise 3: Play with the demo

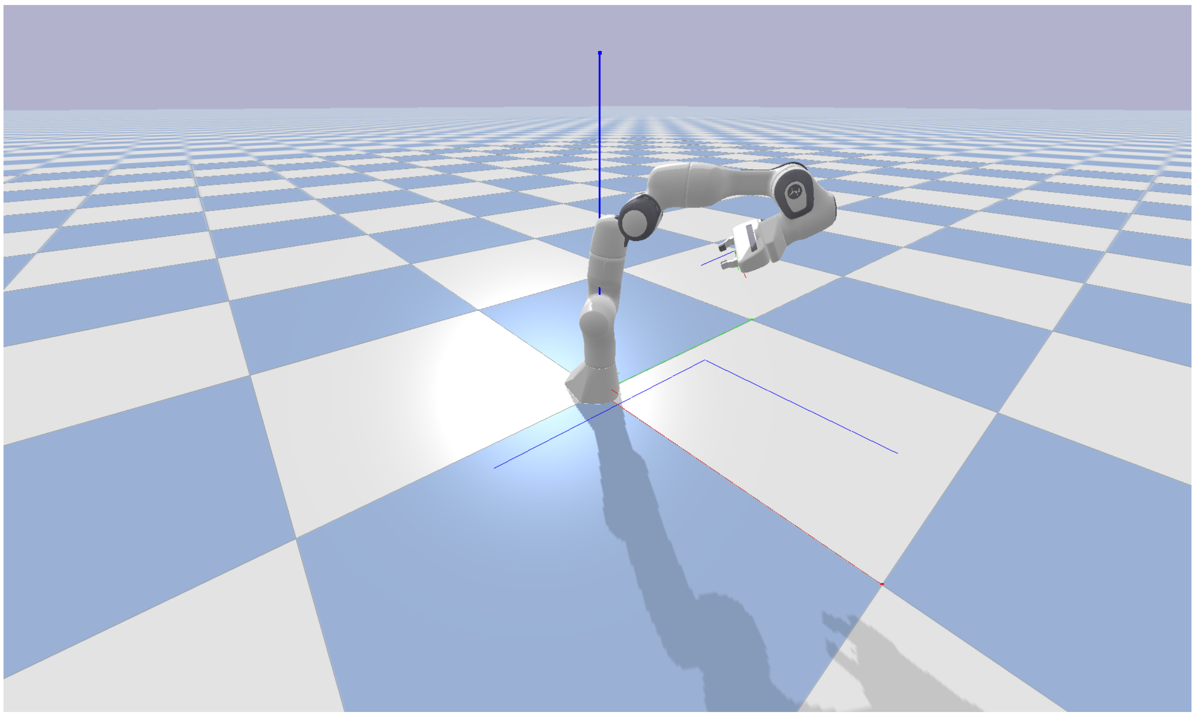
Congratulations! You have completed your coding part. It‘s time to play with a robot arm! (in simulation)

Run the code in `demo.py`, especially using function: `FK_test(robot, physicsClientId)`

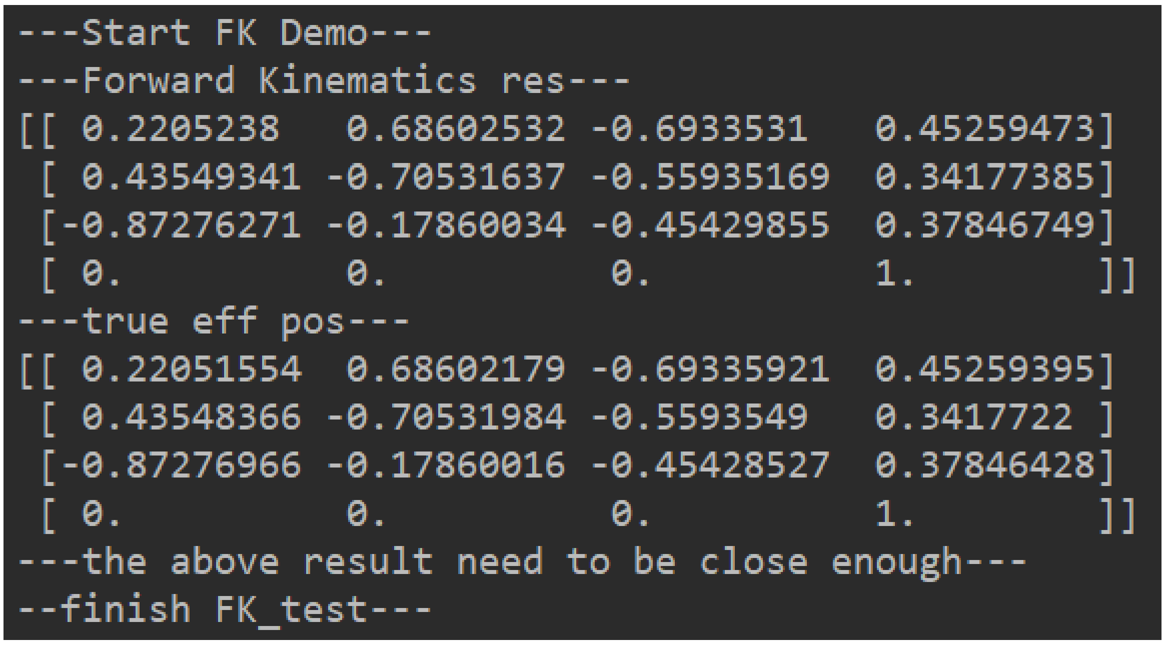
In this demo, you will send commands to set the position of each joint and see whether the real configuration caused by commands is the same as the result of your analytical calculation using your own FK algorithm.

After running the demo, you will see a robot(Franka Panda Arm) like this.

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 FK algorithm is correct, you will find something like this:



```
---Start FK Demo---
---Forward Kinematics res---
[[ 0.2205238   0.68602532 -0.6933531   0.45259473]
 [ 0.43549341 -0.70531637 -0.55935169  0.34177385]
 [-0.87276271 -0.17860034 -0.45429855  0.37846749]
 [ 0.          0.          0.          1.        ]]
---true eff pos---
[[ 0.22051554  0.68602179 -0.69335921  0.45259395]
 [ 0.43548366 -0.70531984 -0.5593549   0.3417722 ]
 [-0.87276966 -0.17860016 -0.45428527  0.37846428]
 [ 0.          0.          0.          1.        ]]
---the above result need to be close enough---
--finish FK_test---
```

This is the end of homework1~

Additional Note: Very IMPORTANT!

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)

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

2. More about autotest: The autotest program will test whether your function output is the same as the correct output given the same input. Each function is tested through 3~10 test cases, the score is the percentage of the number of test cases you pass (just like the OJ system).

3. In the template `core.py`, each function provides an input example and a corresponding output, you can use them to check the code.

4. More about pybullet:

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

5. 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).