Introduction to Robotics: Homework3【Bonus 10】

Homework3 is different from the previous two homework. **Homework3 will be used as a bonus. You can choose whether to complete this part.**

In the following exercises, you will implement your own **inverse dynamic, time scaling and control** algorithm step by step and test these algorithm in simulation using a **Franka Panda 7DoF** arm.

Exercise0: Overview

Just like homework1&2, 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 previous homework(homework1&2) to `core.py` of homework3.**

2. `robot_sim.py`: build a simulation environment. **This time you need to fill in some of functions here**. The position you need to fill in has been marked in this file.
3. `demo3.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.**

- A short overview:

  In this homework, you will directly control the joint torque and apply this torque to achieve your goal. Assume the start pose and target pose are already given, you need to implement how to interpolate between the start and target pose(time scaling), how to calculate desired torque(inverse dynamics) and how to control each actuator(controller).

Exercise 1: Inverse Dynamics【60% score】

In this part, you need to implement the inverse dynamics algorithm.

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. `adV = ad(V)` : Calculate the 6x6 matrix $[ad_{\mathcal{V}}]$ of the given 6-vector.
   - input: V: A 6-vector (e.g., a twist).
   - return: adV: The corresponding 6 × 6 matrix $[ad_{\mathcal{V}}]$.
2. `taulist = inverse_dynamics(thetalist,dthetalist,ddthetalist, g,Ftip,Mlist,Glist,Slist)` : Computes inverse dynamics in the space frame for an open chain robot. **This function uses forward-backward Newton-Euler iterations.**
   - input1: thetalist: n-vector of joint variables θ.

input2: dthetalist: n-vector of joint velocities $\dot{\theta}$.

input3: ddthetalist: n-vector of joint accelerations $\ddot{\theta}$.

input4: g: Gravity vector g.

input5: Ftip: Wrench F_{tip} applied by the end-effector expressed in frame {n + 1}.

input6: Mlist: List of link frames {i} relative to {i − 1} at the home position.

input7: Glist: Spatial inertia matrices $\mathcal{G}_i$ of the links.

input8: Slist: Screw axes S_i of the joints in a space frame.

- return: The n-vector τ of required joint forces/torques.

Exercise 2: Time Scaling【10% score】

NOTE1: write your code in the function: `cubic_time_scaling_theta2()` of `robot_env.py`. **DO NOT MODIFY THE FUNCTION NAME!!!**

1. `desired_position, desired_velocity, desired_acceleration = cubic_time_scaling_theta2(Tf, dt, theta_start, theta_end):` perform cubic time scaling.
   - input1: Tf: total time

     input2: dt: each time step

     input3: theta_start: start joint values

     input4: theta_end: final joint values (same length with theta_start)
   - return1,2,3: desired_position, desired_velocity, desired_acceleration

NOTE1: you can see how to use this function in `joint_PID_control()` & `computedTorqueControl2()`

NOTE2: you can refer to `getTrajectory()` to implement your function

Exercise 3: Control【30% score】

1. Feedback control -- PID control

   You need to fill in the functions template `joint_PID_control()` in `robot_env.py`. The location you need to fill in has been marked.

2. Compute torque control -- Feedback + Feed forward

   You need to fill in the functions template `computedTorqueControl2()` in `robot_env.py`. The location you need to fill in has been marked.

   NOTE1: already argument joint position when calling this functions. The reason for doing this is to make the inverse kinematics algorithm works when the URDF of the robot contains *fixed joint*.

   NOTE2: Mlist, Glist and Slist has been calculated, you can use them directly. BUT! Each row of `Slist` is the joint screw axes in the space frame when the robot is at the home position.

Exercise 4: Play with the demo

Congratulations! You have completed your coding part. Just like homework1&2, it's time to play with a robot arm! (in simulation)

In this demo, you are given the target position and try to move the end-effector to the target position. Frist, using IK algorithm to compute the corresponding target joint angles. Second, using time scaling to generate desired position, desired velocity and desired acceleration for each joint. And finally, control each joints to reach the target position.

1. Set `CONTROLLER` variable in `demo3.py` as: `"PID"`, then run the code in `demo.py`, especially running function: `control_test(robot, physicsClientId)`

   Now, you are using a PID controller

2. Set `CONTROLLER` variable as: `"COMPUTE_TORQUE"`, then run the code in `demo.py`, especially running function: `control_test(robot, physicsClientId)`

   Now, you are using a compute torque control methods(feedforward + feedback).

   If you implement it correctly, you will find something like this:

```
#Trajectory points: 481
Desired joint angles: [ 0.03981615 -0.21037288  0.53014952 -1.7841317   0.10584991  1.60179554
  1.3424176   0.          0.          0.          0.        ]
[[ 9.99993410e-01 -8.36335633e-04  3.53286961e-03  4.00474757e-01]
 [-8.16937898e-04 -9.99984604e-01 -5.48852491e-03  2.99126238e-01]
 [ 3.53740547e-03  5.48560260e-03 -9.99978697e-01  5.50096214e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

you should find that M is close to your desired pose
```

This is the end of homework3~

Additional Note

1. **How to submit your homework**

   Compress all your code `core.py`, `robot_sim.py` ,`demo3.py` and folder `/robot_model` . Then, rename this zip file as as your student ID and submit it 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 previous homework(homework1&2) to `core.py` of homework3, 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).