Task Space Inverse Dynamics

In real-world applications, we are primarily concerned with controlling the end-effector of a manipulator (task-space) rather than individual joint control. In this assignment, you will develop a control algorithm that solves this problem without computing inverse kinematics, instead converting the end-effector pose into a stable differential equation. This approach is known as Task-Space Inverse Dynamics (TSID).

Mathematical Background

Task space tracking can be achieved by modifying the outer-loop control a_q in the conventional inverse-dynamics algorithm (refer to the lecture) while **keeping the inner-loop control unchanged**. Let p represent the end-effector pose in $SE(3)$. Since p is a function of the joint variables $q \in \mathcal{Q}$, we have:

$$\dot{p} = J(q)\dot{q}$$
$$\ddot{p} = J(q)\ddot{q} + \dot{J}(q)\dot{q}$$

By choosing a_q as:

$$a_q = J^{-1}(q)\left\{\begin{bmatrix} a_x \\ a_\omega \end{bmatrix} - \dot{J}(q)\dot{q}\right\}$$

and applying it to the manipulator equations, we obtain the system:

$$\ddot{r} = a_r \in \mathbb{R}^3$$
$$\dot{\omega} = a_\omega \in \mathbb{R}^3$$
$$\dot{R} = S(\omega)R, \quad R \in SO(3), S \in \mathfrak{so}(3)$$

The inner control aims to achieve $r \to r_d, R \to R_d$. This requires understanding the differential kinematics of the $SO(3)$ group. For more details, please refer to the provided documentation.

Note: Implementation of the outer-loop control requires a non-singular Jacobian. Plan your desired points and trajectories to avoid Jacobian singularities. For robots with more or fewer than six joints, the Jacobians will not be square. Alternative approaches using pseudoinverse Jacobians have been developed for such cases.

Assignment:

Design a TSID controller to regulate and track the desired end-effector pose of the UR5/UR10 manipulator:

- **[35 points]** Design the outer loop as described above using the logarithmic map for orientation error. Provide and explain the complete set of equations describing your controller.
- **[40 points]** Implement posture regulation using modeling tools like Pinocchio for computing dynamics, kinematics, and Jacobians. Implement your controller in the MuJoCo simulator and provide plots demonstrating tracking convergence and control signals. Robot descriptions (URDF and XML) are provided. A template with a movable target site is available for your convenience.
- **[25 points]** Demonstrate your algorithm by making the robot follow a task space trajectory (e.g., a circular path while maintaining the end-effector orientation toward the center).

Implementation Notes:

For computing end-effector frame position, velocity, and acceleration using Pinocchio, the following code may be helpful:

```
import pinocchio as pin
# Compute forward kinematics
pin.forwardKinematics(model, data, q, dq, ddq)
# Get end-effector frame ID
ee_frame_id = model.getFrameId("wrist_3_link")
frame = pin.LOCAL
# Calculate kinematics of frames
pin.updateFramePlacement(model, data, ee_frame_id)
# Get velocities and accelerations
twist = pin.getFrameVelocity(model, data, ee_frame_id, frame)
dtwist = pin.getFrameAcceleration(model, data, ee_frame_id, frame)
# Jacobian
J = pin.getFrameJacobian(model, data, ee_frame_id, frame)
# Get the frame pose
ee_pose = data.oMf[ee_frame_id]
ee_position = ee_pose.translation
ee_rotation = ee_pose.rotation
# Transform desired velocity to end-effector frame
desired_twist = ee_pose.actInv(pin.Motion(desired_velocity))
```

Submission Requirements:

Submit a concise report containing all necessary equations, explanations, videos and plots, along with your Python implementation.