Robust Sliding Mode Control for Robot Manipulators

Modern robotic manipulators frequently encounter various uncertainties and disturbances in their operation:

- Varying payload masses
- Joint friction and damping effects
- Model parameter uncertainties
- External disturbances

While conventional inverse dynamics control performs well with accurate model knowledge, sliding mode control (SMC) provides robust performance in the presence of both parametric uncertainties and bounded disturbances.

The manipulator dynamics with uncertainties can be expressed as:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) + D\dot{q} + F_c(\dot{q}) = \tau$$

where:

- $M(q)$: Uncertain mass matrix
- $C(q,\dot{q})$: Uncertain Coriolis/centrifugal terms
- $g(q)$: Uncertain gravity vector
- D: Unknown diagonal viscous friction matrix
- F_c: Unknown Coulomb friction vector
- τ: Control input

Assignment Tasks

1. **[40 points] Theoretical Analysis**
   - Design a sliding mode controller
   - Present a Lyapunov stability analysis
   - Explain the selection criteria for control gains and sliding surface parameters
2. **[40 points] Implementation and Performance Analysis**
   - Modify the UR5 robot model to include:
     - Additional end-effector mass
     - Joint damping coefficients
     - Coulomb friction
   - Implement both controllers:
     - Basic inverse dynamics controller

    - Designed sliding mode controller
  - Compare their performance through:
    - Tracking errors and control efforts
3. **[20 points] Boundary Layer Implementation**
  - Analyze the chattering phenomenon:
    - Causes and practical implications
    - Boundary layer modification for smoothing
  - Evaluate performance with varying boundary layer thicknesses Φ
  - Analyze the robustness-chattering trade-off

Implementation Guidelines

To modify robot parameters in the simulator:

```
# Set joint damping coefficients
damping = np.array([0.5, 0.5, 0.5, 0.1, 0.1, 0.1])  # Nm/rad/s
sim.set_joint_damping(damping)

# Set joint friction coefficients
friction = np.array([1.5, 0.5, 0.5, 0.1, 0.1, 0.1])  # Nm
sim.set_joint_friction(friction)

# Modify end-effector mass
sim.modify_body_properties("end_effector", mass=4)
```

A template is available here.

For the inverse dynamics controller, use these reference gains:

```
kp, kd = 100, 20
```

These gains, combined with the specified end-effector mass, will clearly demonstrate the effects of parameter uncertainty on control performance.

For dynamics computation with Pinocchio:

```
import pinocchio as pin
# Compute dynamics
pin.computeAllTerms(model, data, q, dq)
M = data.M  # Mass matrix
nle = data.nle  # Nonlinear effects (C*dq + g)
```

Evaluation Criteria

Submissions will be evaluated on:

1. Depth of theoretical understanding
2. Quality of implementation
3. Thoroughness of comparative analysis
4. Practical handling of chattering

Submission Requirements

Your submission should include:

1. Theoretical derivations with stability analysis
2. Controller implementations
3. Comparative performance analysis
4. Performance plots and metrics
5. Video demonstrations
6. Code implementation (e.g., repository link)

References

1. Lecture notes and slides on robust control
2. Mark W. Spong, Seth Hutchinson, and M. Vidyasagar, "Robot Modeling and Control"
3. J.-J. E. Slotine, "Applied Nonlinear Control"