Jiaming Zhang

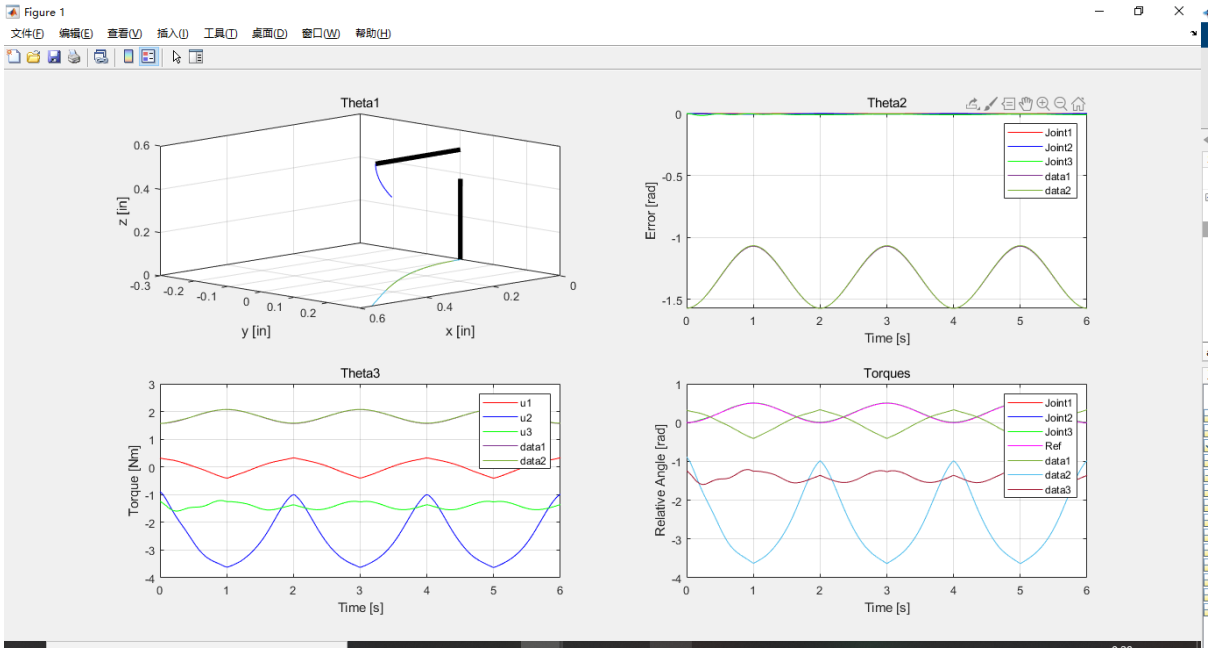
1. Inverse Dynamics Control Algorithm

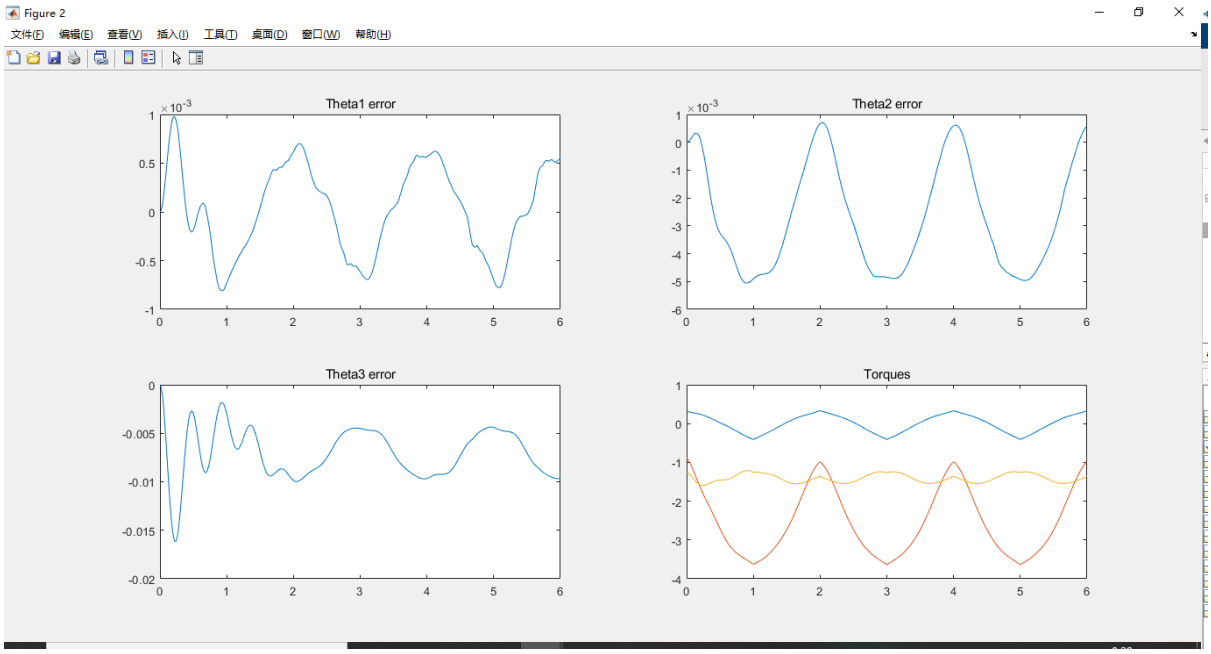
On the whole, we want to control the robot to move at the desired joint angle, and to achieve that goal, we need to get the torque that each joint needs to apply. For the real robot lab, we just need to give the torque to each joint and then it will move as required, and for the simulation, we need to create an model (actually already given to us) to represent the actual robot, and give the torque as input, and we will get the angle of all joints. This also can be called the inner loop. In order to get the torque, we need to use inner control loop. Because in the simulation we can easily get the value of angle and angular velocity, but if we approximate the angular acceleration. The error will be very large. So, we use PD gains and the desired angle, desired angular velocity and desired angular acceleration to calculate the acceleration.

2. Inverse Dynamics Control Code

```
Kp = zeros(3,3);
Kp(1,1) = 200;
Kp(2,2) = 200;
Kp(3,3) = 200;

Kd = zeros(3,3);
Kd(1,1) = 2;
Kd(2,2) = 2;
Kd(3,3) = 3;

a_theta = Kp*(qd - q) + Kd * (dqd - dq) + ddqd;

u = DforController*a_theta + CforController*dq + GforController;
```

3. Inverse Dynamics Control Plot



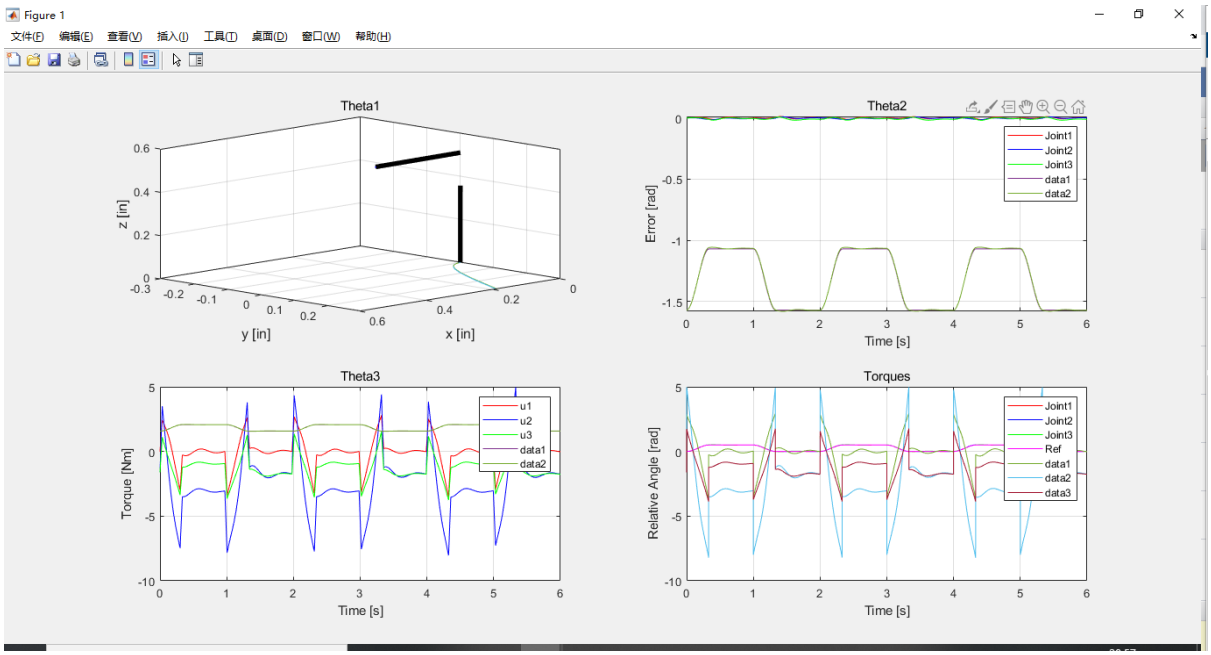


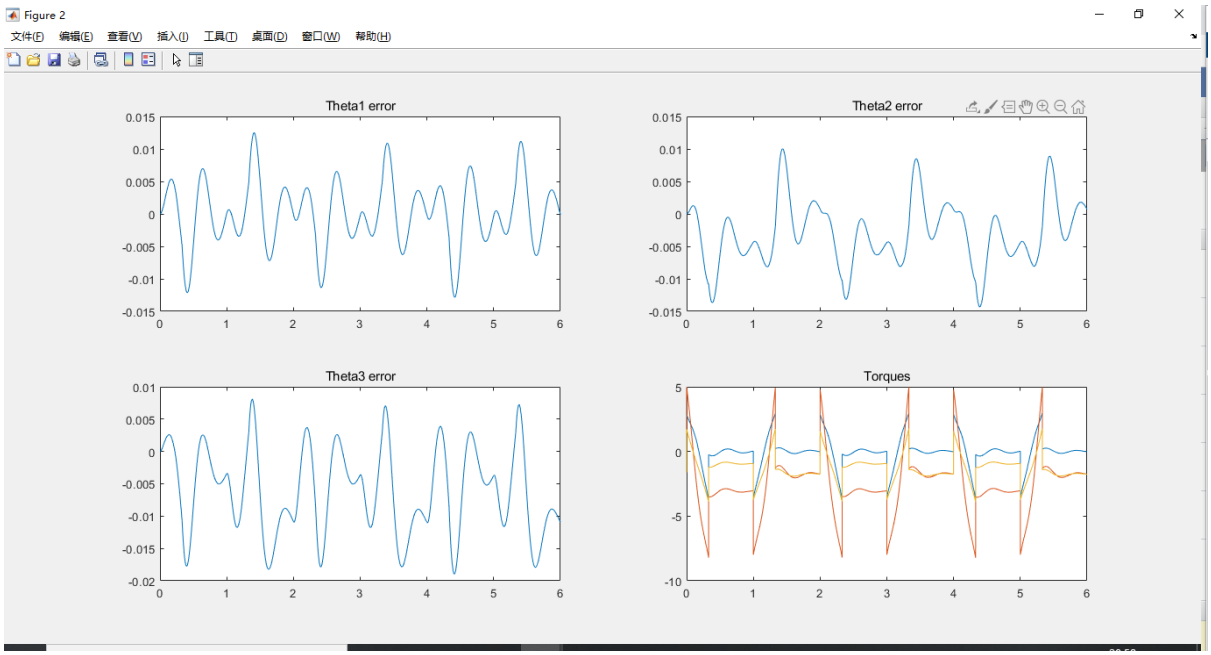
4. Fast trajectory generation

```
    elseif p.flag_ctrl == 4 || p.flag_ctrl == 5 || p.flag_ctr7l == 6 || p.flag_ctrl == 7

        % TODO: Create your fast trajectory here
        t_mod = mod(t,2);

        if ((t_mod > 0) && (t_mod <= 0.33))
            qd = [-27*t_mod^3 + 13.5*t_mod^2;
                  -27*t_mod^3 + 13.5*t_mod^2 - pi/2;
                  -27*t_mod^3 + 13.5*t_mod^2 + pi/2];
            dqd = [-81*t_mod^2 + 27*t_mod;
                   -81*t_mod^2 + 27*t_mod;
                   -81*t_mod^2 + 27*t_mod];
            ddqd = [-162*t_mod + 27;
                    -162*t_mod + 27;
                    -162*t_mod + 27];

        elseif ((t_mod > 0.33) && (t_mod < 1))
            qd = [0.5; 0.5 - pi/2; 0.5 + pi/2];
            dqd = [0.0; 0.0; 0.0];
            ddqd = [0.0; 0.0; 0.0];

        elseif ((t_mod >= 1) && (t_mod <= 1.33))
            qd = [27*t_mod^3 - 94.5*t_mod^2 + 108*t_mod - 40;
                  27*t_mod^3 - 94.5*t_mod^2 + 108*t_mod - 40 - pi/2;
                  27*t_mod^3 - 94.5*t_mod^2 + 108*t_mod - 40 + pi/2];
            dqd = [81*t_mod^2 - 189*t_mod + 108;
                   81*t_mod^2 - 189*t_mod + 108;
                   81*t_mod^2 - 189*t_mod + 108];
            ddqd = [162*t_mod - 189;
                    162*t_mod - 189;
                    162*t_mod - 189];
        else
            qd = [0.0; -0.5*pi; 0.5*pi];
            dqd = [0.0; 0.0; 0.0];
            ddqd = [0.0; 0.0; 0.0];
        end
%         disp(t_mod);
%         disp(qd);
%         disp(dqd);
%         disp(ddqd);
        traj_d = [qd; dqd; ddqd];
    end
end
```

5. Inverse + Fast trajectory

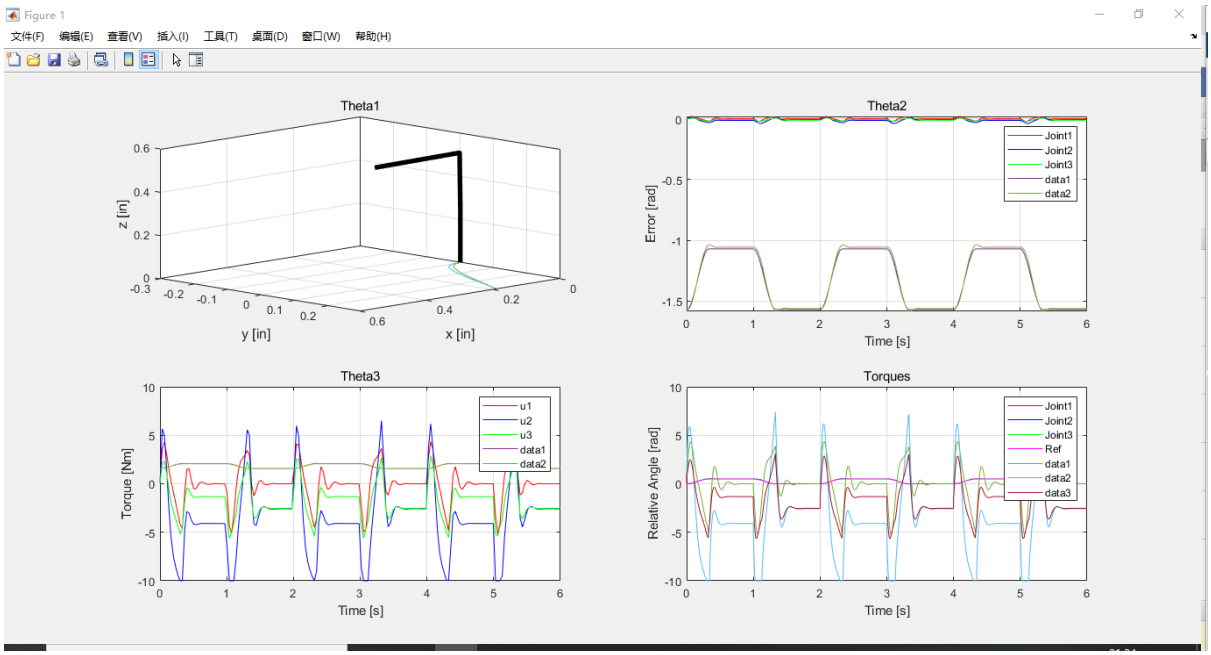
```
Kp = zeros(3,3);
Kp(1,1) = 200;
Kp(2,2) = 200;
Kp(3,3) = 200;

Kd = zeros(3,3);
Kd(1,1) = 5;
Kd(2,2) = 7;
Kd(3,3) = 9;
```

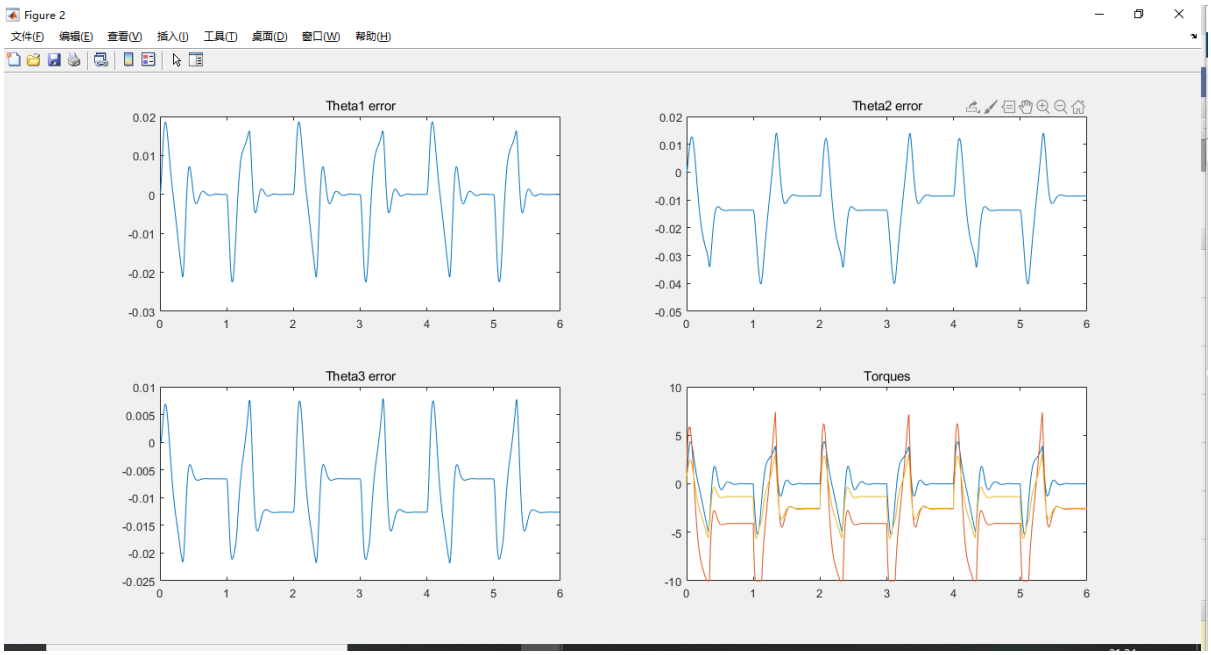




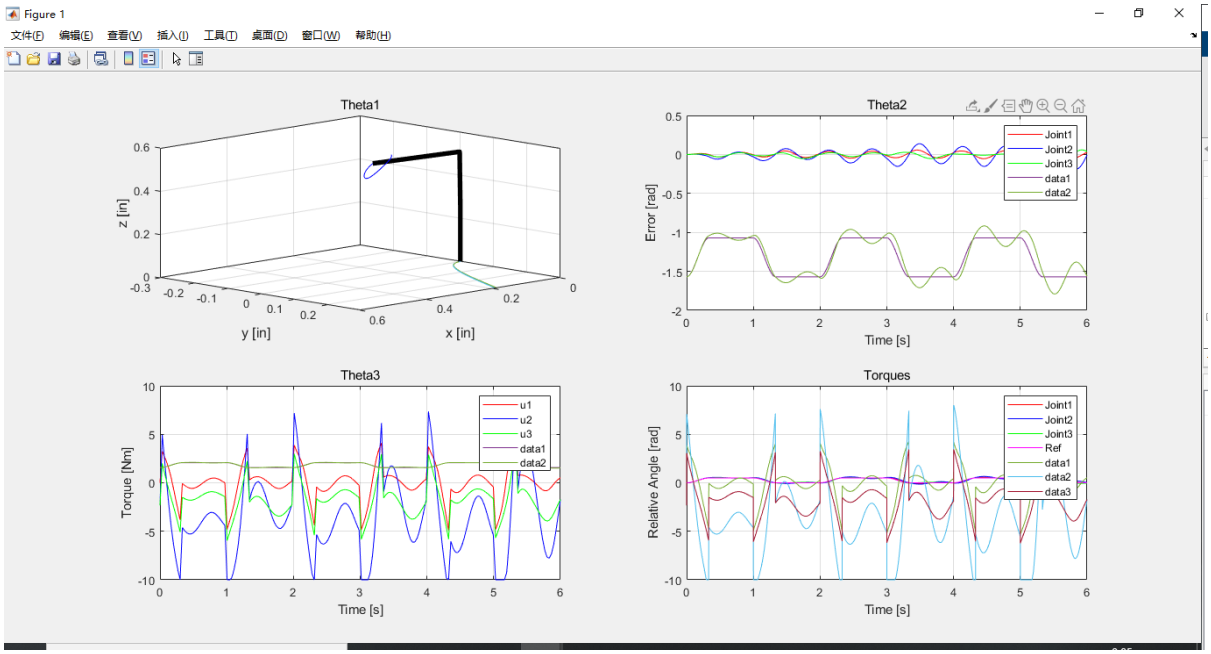
6. PD+Feedforward+Fast Trajectory

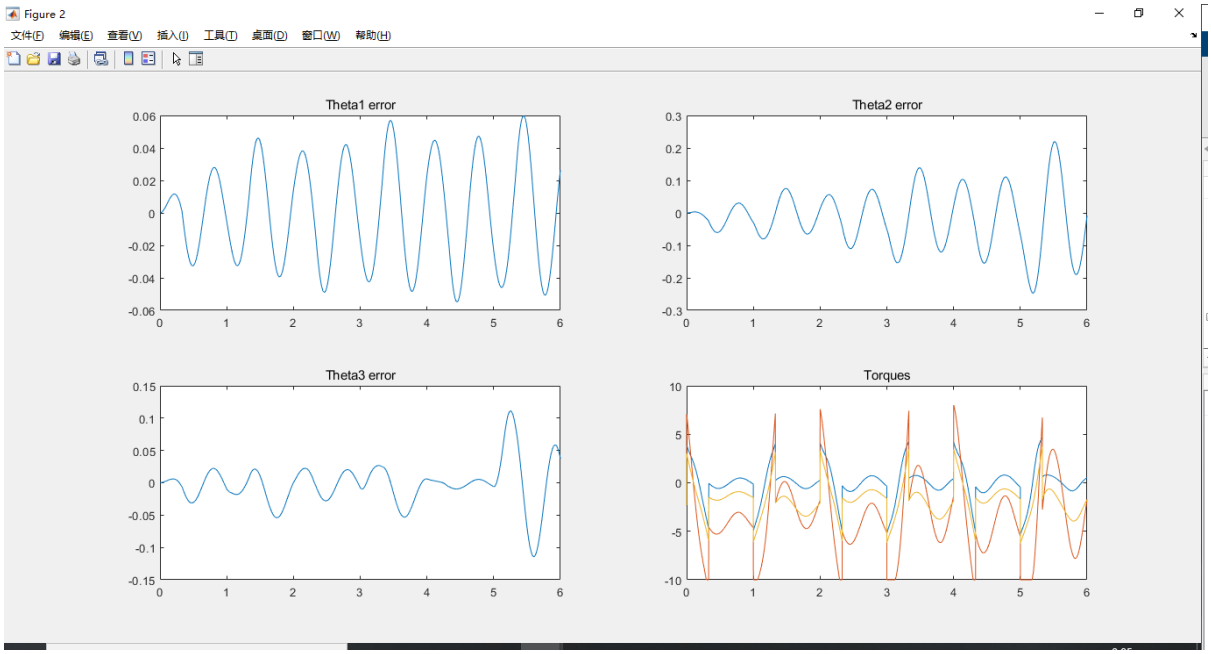
```
Kp = zeros(3,3);
Kp(1,1) = 200;
Kp(2,2) = 300;
Kp(3,3) = 200;

Kd = zeros(3,3);
Kd(1,1) = 4;
Kd(2,2) = 12;
Kd(3,3) = 6;
```



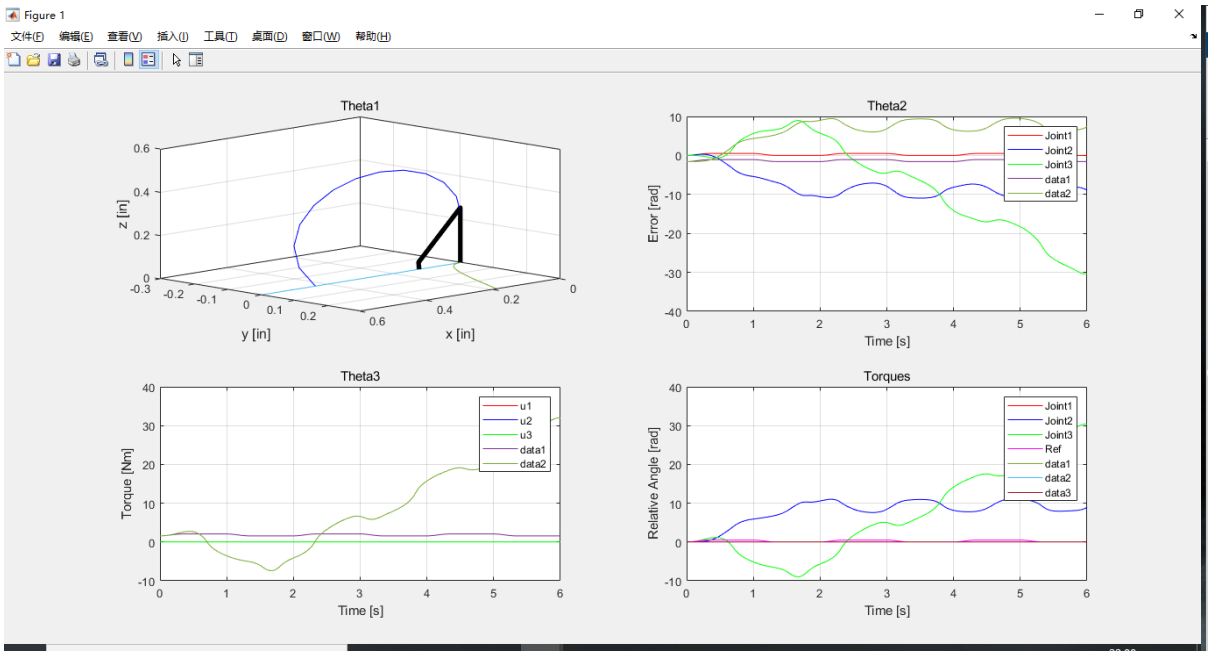


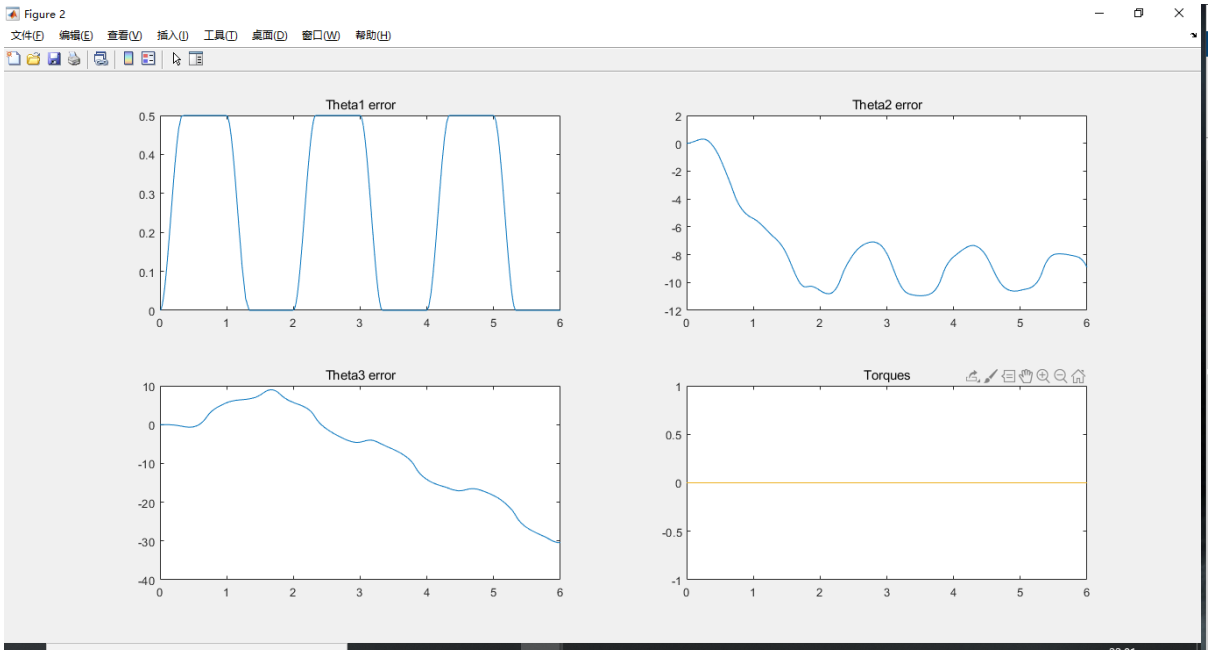
7. Inverse + Fast trajectory + Mass





8. PD+Feedforward + Fast trajectory + Mass





9. Comparison of two control method

   For a specific system, both PD+feedback control and Inverse control can have good control performance. However, if the model has some change both of the two control methods will have worse performance. But the inverse control performs much better than PD+feedback control. The inverse control still stable but has more overshoot and longer adjust time. But the PD+feedback control is unstable. In others words, inverse control has better robust character.