Sliding Control of a Robot Arm for the Tower of Hanoi Game

Gourav Khadge and Leif Park Jordan

5/8/2015

1. **Introduction and Project Statement**

The Tower of Hanoi game consists of three rods and several disks of decreasing radius. The disks can be stacked on top of one another, but larger disks cannot be placed on top of smaller disks. The goal of the game is to move the stack of disks from the leftmost rod to the rightmost rod. This report will describe an approach used to control a robot arm capable of playing the Tower of Hanoi game. The control system must allow the arm to reach down, pick up a disk, pull it off the rod, and place it on a second rod. The arm must begin and end each move at rest with zero acceleration, it must be able to move the disks in less than 5 seconds, it must stay within 1 mm of a straight line when raising and lowering the disk, and it must be able to adapt to unknown parameters. Specifically, the controller’s estimates of the system’s link inertias, joint/actuator friction coefficients, and end effector load masses can vary from their nominal values. Finally, the maximum torque on each robot arm joint is 3 N-m. The performance of the controller will be verified by programming it to move the robot arm in the pattern shown in Figure 1.

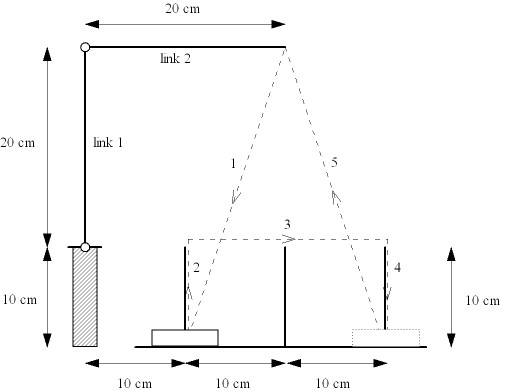


Figure 1: Target path for Hanoi robot arm.

1. **Design**

The design of the control system consisted of three components – the trajectory generator, the feedback linearized control, and the sliding control. These subsystems were developed one at a time and integrated into the overall controller.

The trajectory generator uses an idealized model of the robot arm to generate a trajectory that results in the arm having zero velocity and acceleration at each “corner” of its motion. This is accomplished by using a 5th order polynomial for each point in the movement:

Where *t* is the target time and *tf* is move time. The lower-order terms are zero because of the requirement that the arm reaches its destination with zero velocity or acceleration. The actual position of the end effector at time *t* is given by

Where *p0* is the initial position for the move and ­*pf* is the final position for the move. The code for this

Once the target trajectory has been generated, the feedback controller is used to ensure that the end effector follows it. The feedback controller is an optimal PD controller designed for the linearized version of the robot arm. We define the error vectors

So

We then use the following nonlinear control law:

Which results in the linearized system:

We can then use these error vectors as state variables in our linearized system:

Where KP and KD are the proportional and derivative feedback gains, respectively. These gains were chosen using MATLAB’s lqr function, with , , and the weighting matrices Q and R chosen to heavily penalize position and velocity errors (making Q a diagonal matrix with large weights in each diagonal entry).

In order to improve the system’s response to unknown parameters, we added a sliding control input. We define the new input to the feedback linearized system as

Where w is the sliding control vector, K is the matrix of proportional and derivative gains determined in the feedback linearization step, and x is the error vector. Choosing a quadratic Lyapunov function

We want P to be a positive definite symmetric matrix that satisfies the Lyapunov equation

Where

Using MATLAB’s lyap function, we can find the value of P for any given KP and KD. We then choose the switching surface to be

Where

To find the sliding control gain vector, we choose U such that

Where . This guarantees global asymptotic stability.

This analysis results in the sliding control block diagram shown in Figure 2.

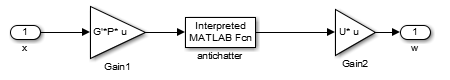


Figure 2: Sliding Control block diagram

The antichatter block eliminates chatter by dividing *s* by its magnitude only when the magnitude of *s* is greater than a parameter we call epsilon.

When fully assembled, the system block diagram appears as shown in Figure 3.

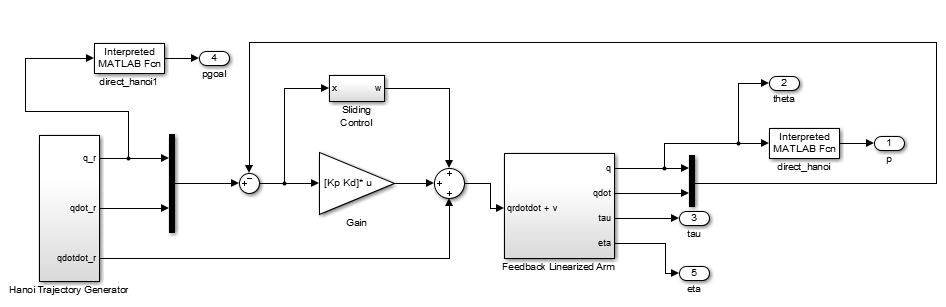


Figure 3: Full system block diagram.

The feedback linearized arm block uses the estimated model of the arm to generate the control signal for the actual arm, which is simulated using the nominal parameter values. This block’s diagram is shown in Figure 4.

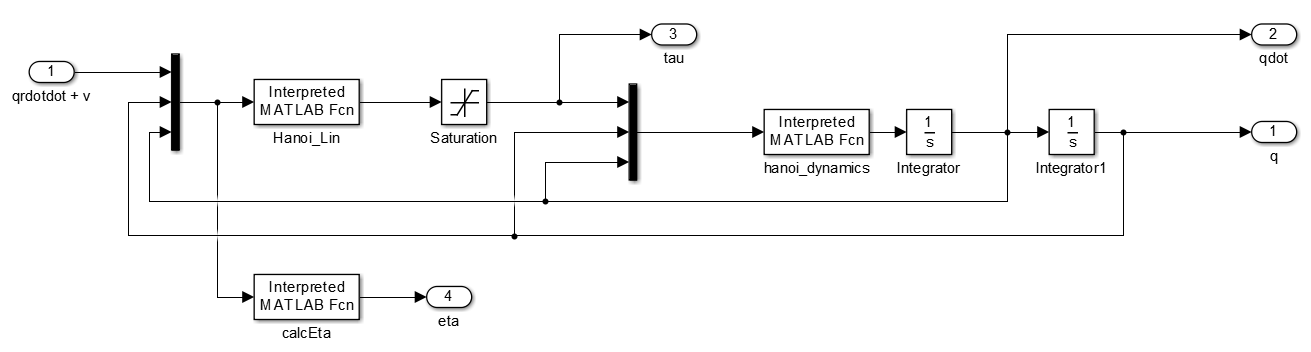


Figure 4: Feedback linearlized arm. Hanoi\_Lin simulates the arm with the estimated parameters, and Hanoi\_dynamics simulates the arm with the nominal parameters. CalcEta calculates the eta parameter.

1. **Simulation Results**

The simulation results are shown in this section. With our optimal controller and sliding controller, we produced the plots in the three worst case scenarios of bad estimates in load mass, joint friction coefficient, and link inertia respectively in , , and . In each case, the maximum deviation during moves 2 and 4 was less than the required 1 mm. For each simulation, the sliding control gain and epsilon for antichattering was tuned properly, according to the approximate maximum value of η.

For these results, Kp and Kd had values of

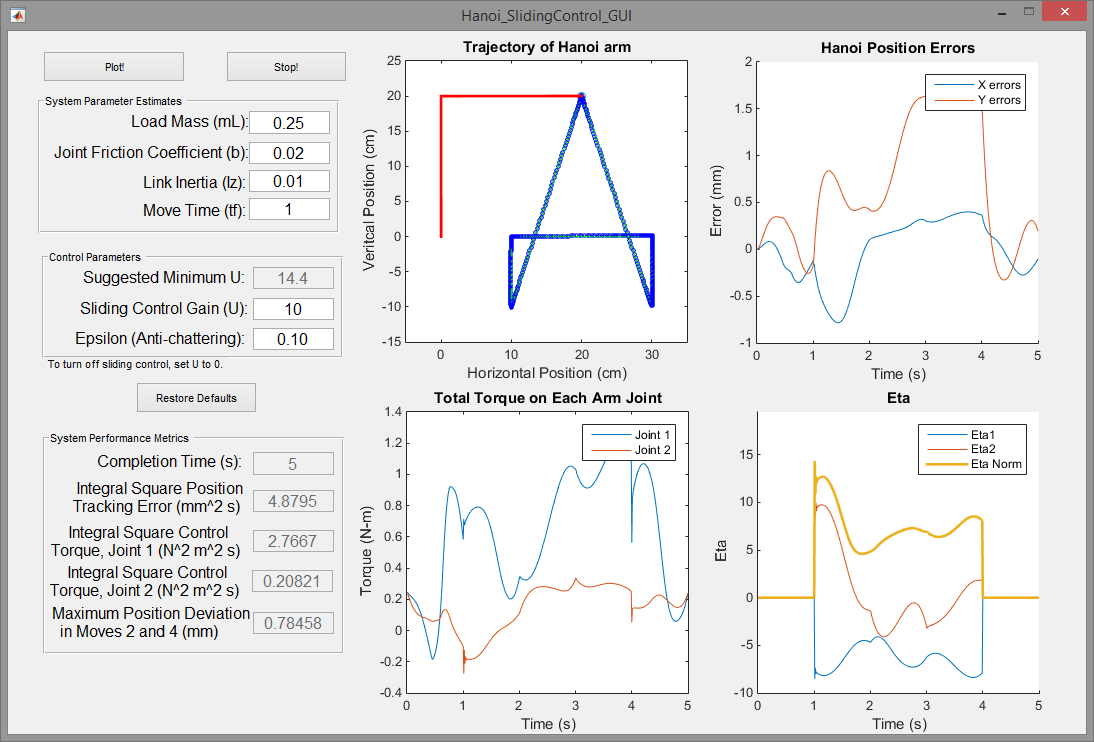


Figure 5: Worst Load Mass estimate

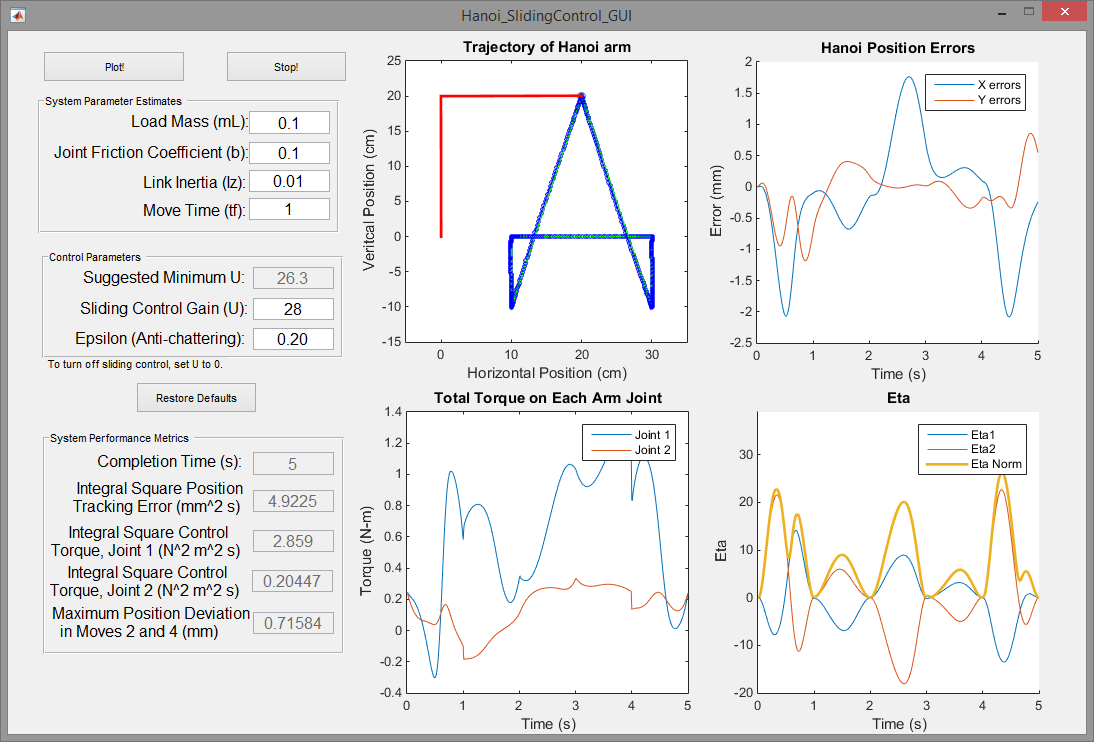


Figure 6: Worst Joint Friction Coefficient estimate

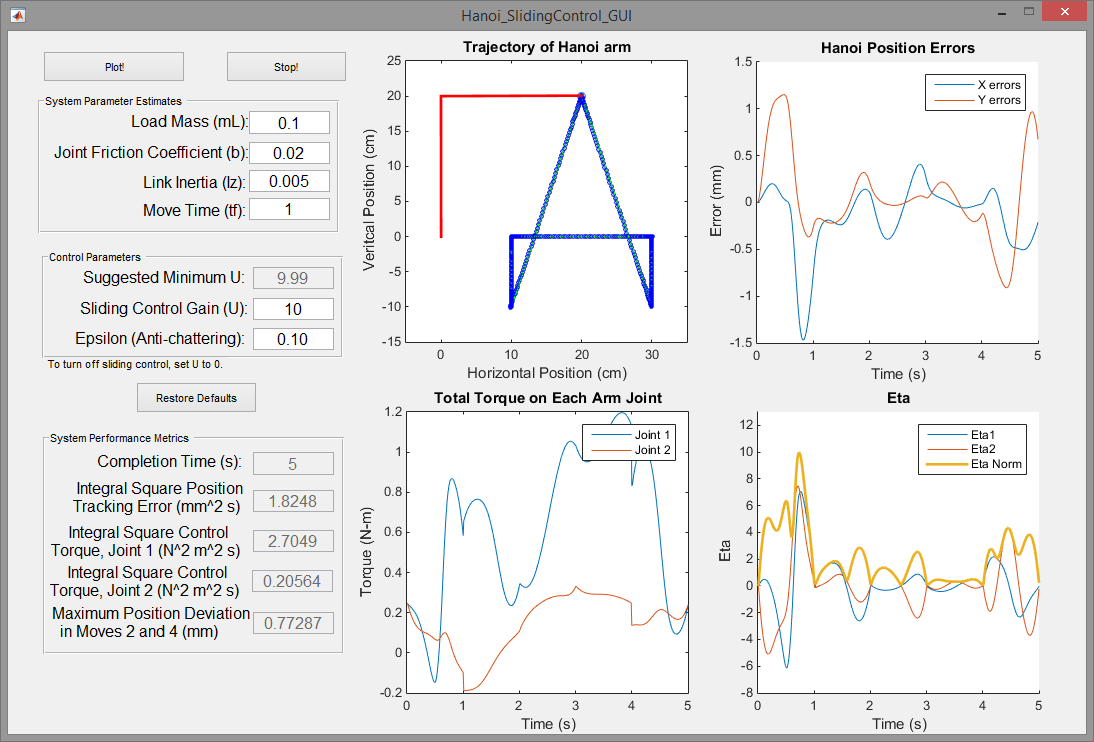


Figure 7: Worst Link Inertia estimate

The effects of sliding control are more visible when Kp and Kd are smaller. The following figures show the effects of mistaken parameter estimates with no sliding control, minimal suggested sliding control gain, and a large sliding control gain with smaller values of Kp and Kd.

The effects of increasing the sliding control gain are noticeable, and reduce the error significantly.

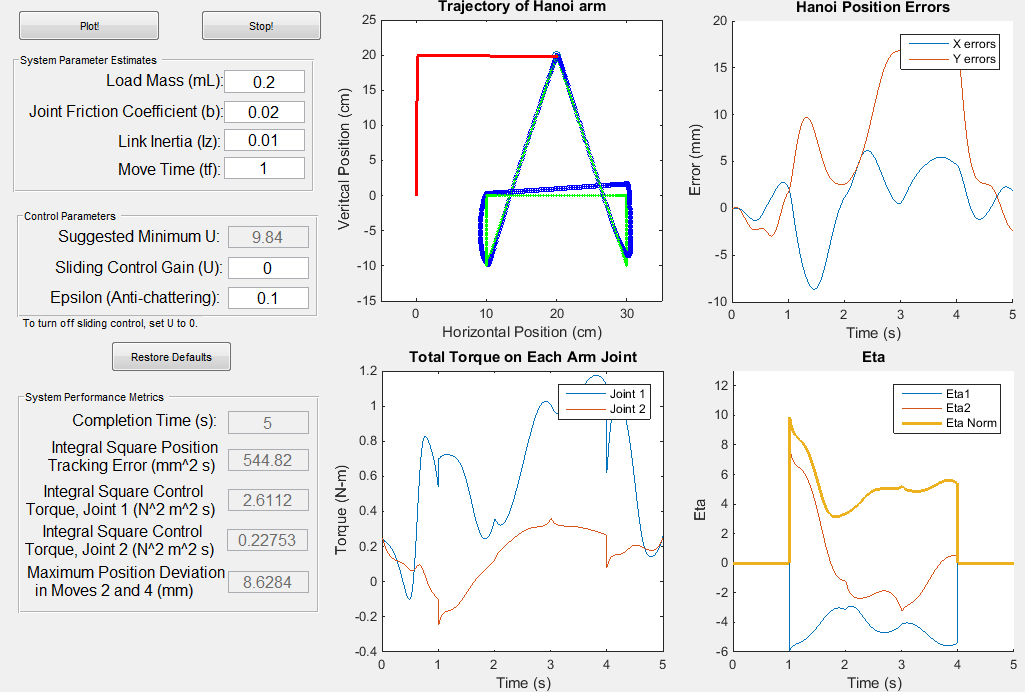


Figure 8: Load Mass is estimated to be 0.2, when it is actually 0.1, without sliding control

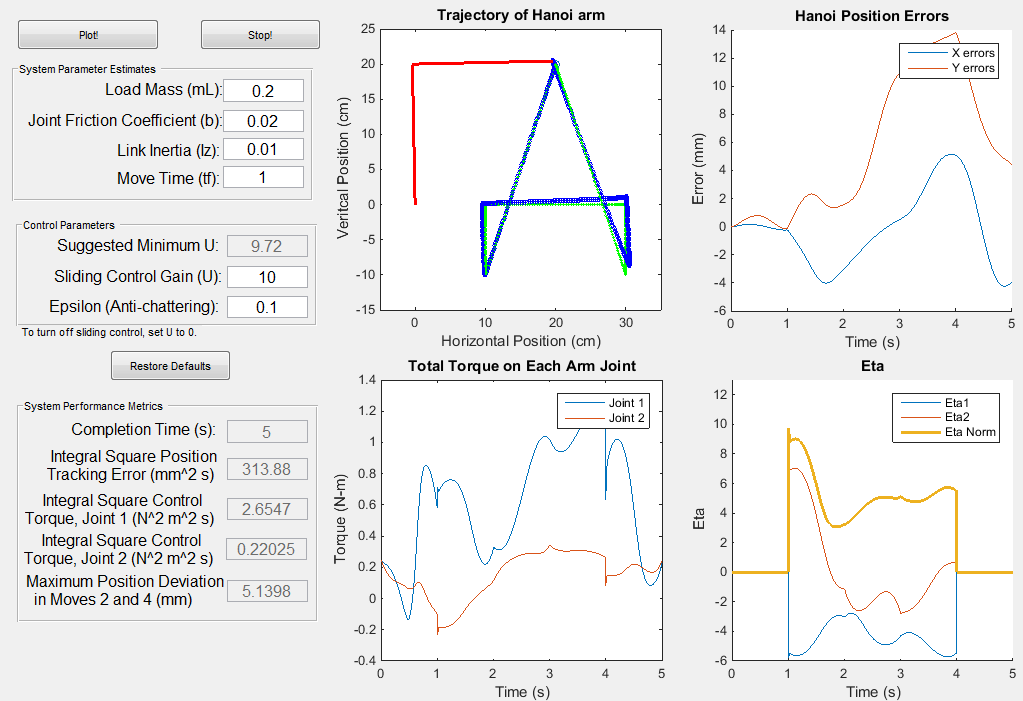


Figure 9: Load Mass is estimated to be 0.2, when it is actually 0.1, with sliding control

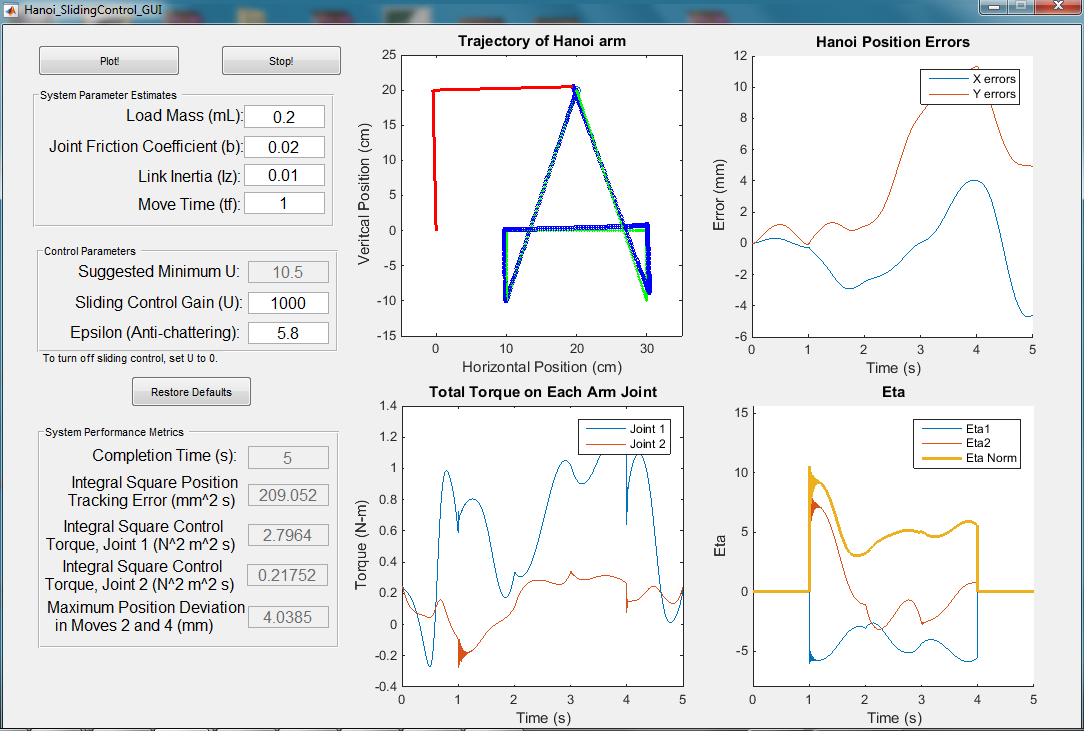


Figure 10: Load Mass is estimated to be 0.2, when it is actually 0.1, large sliding control gain

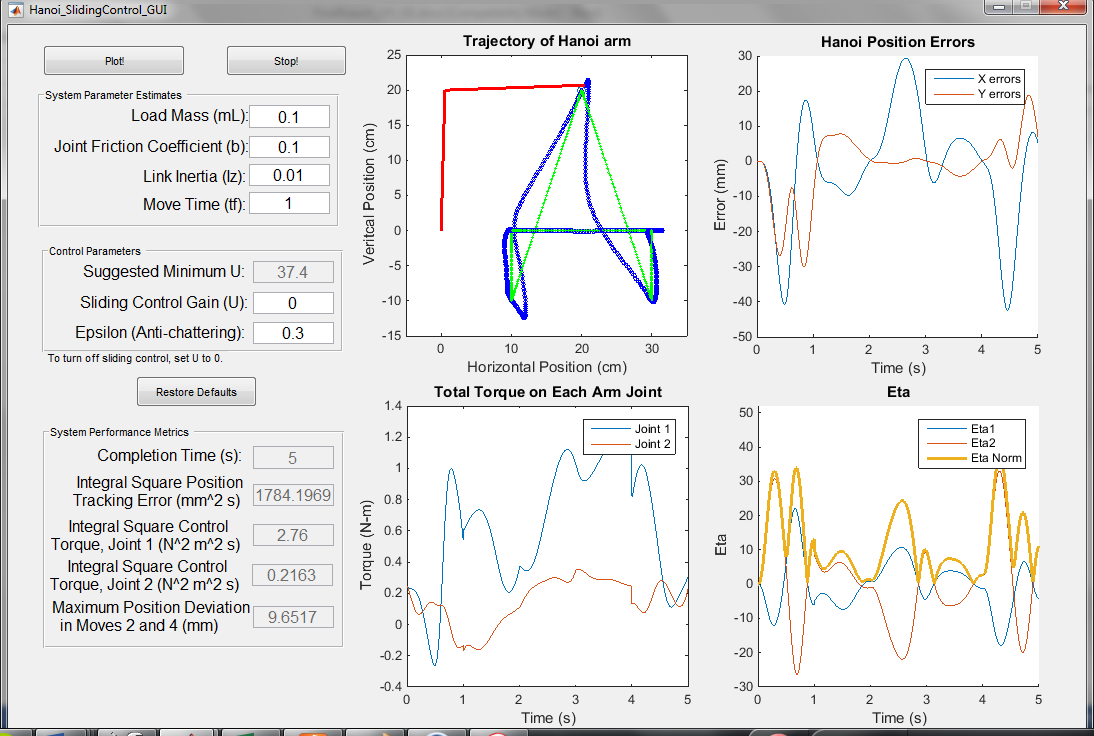


Figure 11: Joint Friction Coefficient is estimated 0.1, but actually 0.02, no sliding control

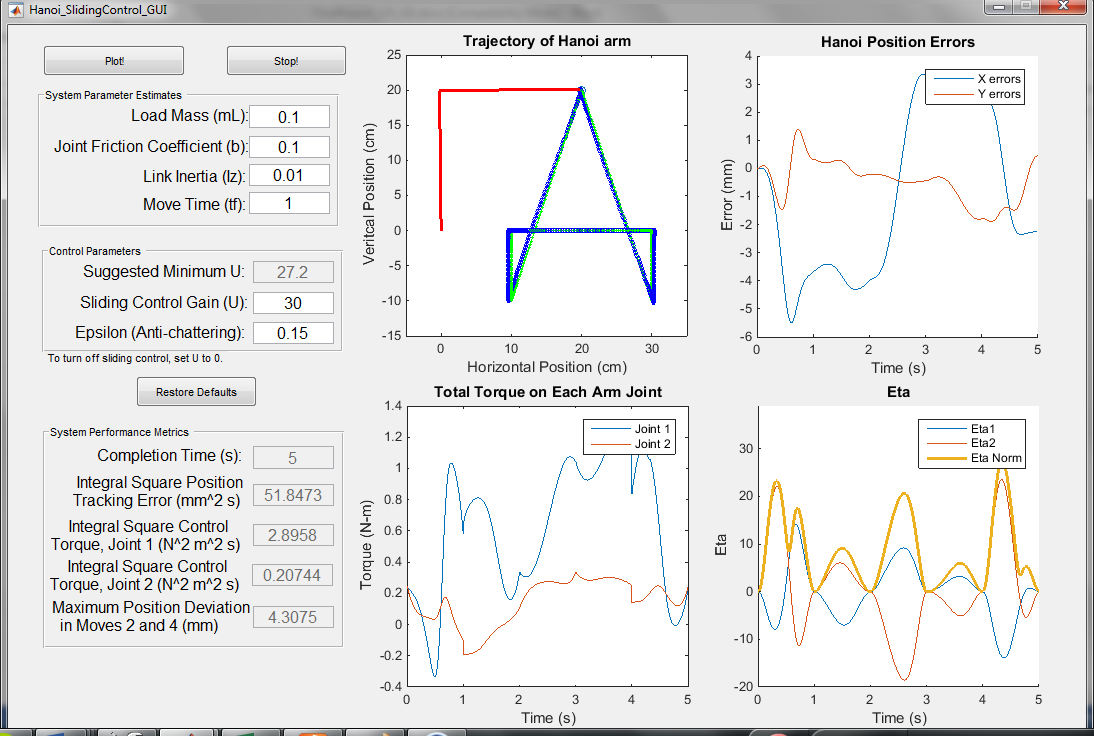


Figure 12: Joint Friction Coefficient is estimated 0.1, but actually 0.02, sliding control

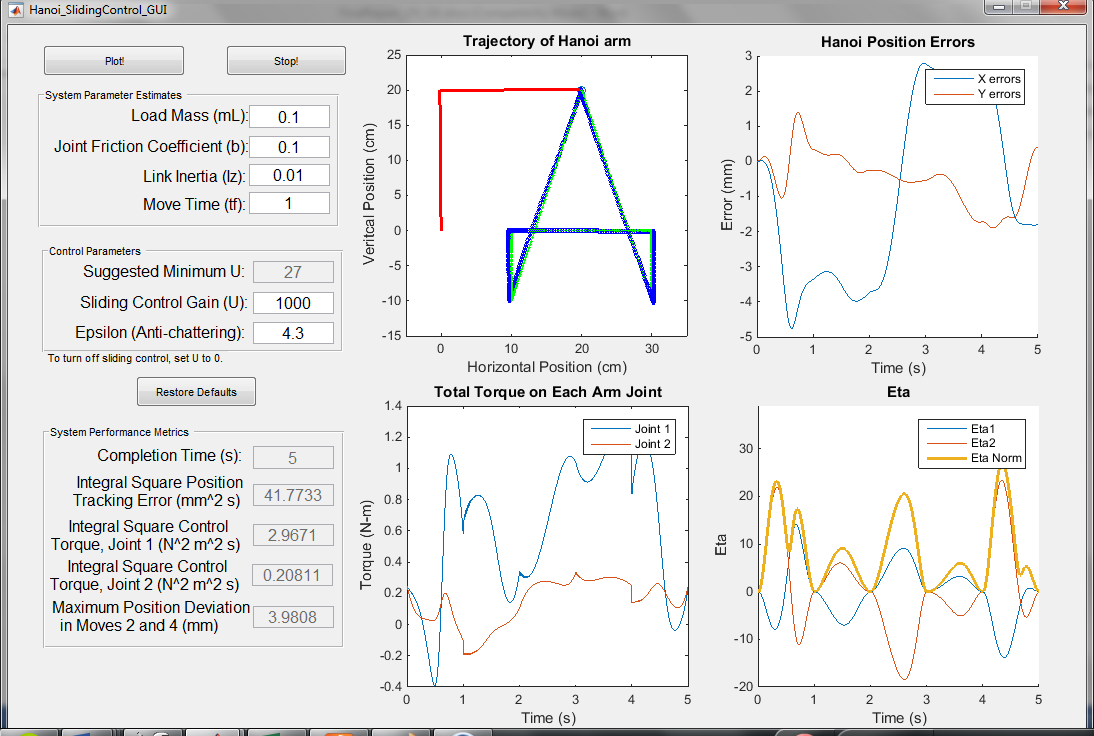


Figure 13: Joint Friction Coefficient is estimated 0.1, but actually 0.02, large sliding control gain

1. **Conclusion**

In the previous lab, we found that open loop control produces poor performance and unstable behavior in our robot manipulator. Even with accurate estimates of the system parameters, the error of the manipulator’s motions grew into instability. In this GUI, linearized feedback optimal control is used to implement feedback-based control, and sliding control is added to give parameter estimate robustness. We have found that feedback has much improved the tracking error of the manipulator. Additionally, with various parameters in uncertain states, the sliding control has been shown to have noticeable effect in improving the system’s performance, making our system more robust and reliable.

**Appendix A: Matlab Pushbutton Callback**

% --- Executes on button press in plotButton.  
function plotButton\_Callback(hObject, eventdata, handles)  
% hObject handle to plotButton (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)

assignin('base','stop',0);  
  
% set values (ask whether they go to traj generator or arm as well)  
global mL;  
global b;  
global tf;  
global Iz;  
global epsilon;  
% epsilon = 1;  
epsilon = str2double(get(handles.epsilonEntry,'String'));  
  
mL = str2double(get(handles.LoadMassEntry,'String'));  
b = str2double(get(handles.frictionEntry,'String'));  
tf = str2double(get(handles.moveTimeEntry,'String'));  
Iz = str2double(get(handles.linkInertiaEntry,'String'));  
U = str2double(get(handles.SC\_GainEntry,'String'));  
assignin('base','U',U);  
G = [zeros(2,2);eye(2)];  
assignin('base','G',G);  
P = eye(4);  
assignin('base','P',P);  
  
% Kp = eye(2)\*447;  
% Kd = eye(2)\*43.5;  
Kp = eye(2)\*1414.2;  
Kd = eye(2)\*61.9;  
% Kp = eye(2)\*100;  
% Kd = eye(2)\*101;  
assignin('base','Kp',Kp);  
assignin('base','Kd',Kd);  
  
% run sim  
[tout, ~, yout] = sim('FeedbackLinearizedArm.slx', tf\*5);  
assignin('base','yout','yout')  
p = yout(:,1:2);  
theta = yout(:,3:4);  
Tau = yout(:,5:6);  
pgoal = yout(:,7:8);  
eta = yout(:,9:10);  
etaNorm = sqrt(eta(:,1).^2 + eta(:,2).^2);  
maxEta = max(etaNorm);  
assignin('base','eta',eta);  
assignin('base','maxEta',maxEta);  
% assignin('base','p',p);  
% assignin('base','pgoal',pgoal);  
% error = norm(p-pgoal);  
pdiff = p - pgoal;  
error = 10\*sqrt(pdiff(:,1).^2 + pdiff(:,2).^2); % in mm  
% assignin('base','error',error);  
errorsquare = error.^2;  
dt = 0.01;  
sumErrorSquare = sum(errorsquare\*dt);  
  
sumTau1 = sum(Tau(:,1).^2)\*dt;  
sumTau2 = sum(Tau(:,2).^2)\*dt;  
  
moves24 = find((tf<=tout & tout<2\*tf) | (3\*tf<=tout & tout<4\*tf));  
% errors24 = error(moves24);  
errorsX24 = 10\*abs(pdiff(moves24,1)); % in mm  
maxdeviation24 = max(errorsX24);  
  
set(handles.minUDisplay,'String', num2str(maxEta,3));  
  
set(handles.completionTimeDisplay,'String', num2str(tf\*5));  
set(handles.ISerrorDisplay,'String', num2str(sumErrorSquare));  
set(handles.IStorque1Display,'String', num2str(sumTau1));  
set(handles.IStorque2Display,'String', num2str(sumTau2));  
set(handles.deviationDisplay,'String', num2str(maxdeviation24));  
set(handles.plotButton, 'Enable', 'off');  
% plot stuff  
axes(handles.errorAxes)  
cla;  
hold all  
plot(tout,pdiff(:,1).\*10);  
plot(tout,pdiff(:,2).\*10);  
title('Hanoi Position Errors')  
xlabel('Time (s)')  
ylabel('Error (mm)')  
legend('X errors','Y errors');  
  
% axes(handles.angleAxes)  
% plot(tout,theta\*360/(2\*pi));  
% title('Hanoi Arm Angles')  
% xlabel('Time (s)')  
% ylabel('Angle (degrees)')  
% legend('Joint 1 angle', 'Joint 2 angle');  
  
axes(handles.angleAxes)  
cla;  
hold all;  
plot(tout,eta(:,1));  
plot(tout,eta(:,2));  
plot(tout,etaNorm,'LineWidth', 2)  
title('Eta')  
xlabel('Time (s)')  
ylabel('Eta')  
legend('Eta1', 'Eta2','Eta Norm');  
axis auto  
ylims = ylim;  
ylim([ylims(1) ylims(2)\*1.3]);  
  
axes(handles.torqueAxes)  
plot(tout,Tau);  
title('Total Torque on Each Arm Joint')  
xlabel('Time (s)')  
ylabel('Torque (N-m)')  
legend('Joint 1', 'Joint 2');  
  
axes(handles.trajAxes);  
cla;  
a = 20;  
n = 1;  
plot(p(1,1),p(1,2),'o-');  
% plot(pgoal(1,1),pgoal(1,2),'o-','Color', 'Green');  
link1 = line([0 a\*cos(theta(n,1))], [0 a\*sin(theta(n,1))],'LineWidth',2, 'Color','Red');  
link2 = line([a\*cos(theta(n,1)) a\*cos(theta(n,1)) + a\*cos(theta(n,1)+theta(n,2))], ...  
 [a\*sin(theta(n,1)) a\*sin(theta(n,1)) + a\*sin(theta(n,1)+theta(n,2))],'LineWidth',2,'Color','Red');  
xlim(gca,[-40 40])  
ylim(gca,[-40 40])  
title('Robot Manipulator')  
xlabel('X position (cm)')  
ylabel('Y position (cm)')  
drawnow  
  
skip = 1;  
  
tic  
t = toc;  
for i = [2:skip:length(p(:,1)),length(p(:,1))]  
% tic;  
 t = t+0.01;  
 delete(link1)  
 delete(link2)  
% cla(handles.axes\_SimPlot);  
 hold all;  
 plot(p(i-skip:i,1),p(i-skip:i,2),'o-', 'Color', 'Blue', 'MarkerSize',3);  
 plot(pgoal(i-skip:i,1),pgoal(i-skip:i,2),'o-', 'Color', 'Green', 'MarkerSize',1);  
 link1 = line([0 a\*cos(theta(i,1))], [0 a\*sin(theta(i,1))],'LineWidth',2, 'Color','Red');  
 link2 = line([a\*cos(theta(i,1)) a\*cos(theta(i,1)) + a\*cos(theta(i,1)+theta(i,2))], ...  
 [a\*sin(theta(i,1)) a\*sin(theta(i,1)) + a\*sin(theta(i,1)+theta(i,2))],'LineWidth',2,'Color','Red');  
  
  
% plot(p(1:i,1), p(1:i,2));  
 axis([-5 35 -15 25]);  
 title('Trajectory of Hanoi arm')  
 xlabel('Horizontal Position (cm)');  
 ylabel('Veritcal Position (cm)');  
 drawnow;  
  
 stop = evalin('base','stop');  
 if (stop == 1)  
 set(handles.plotButton, 'Enable', 'on');  
 return;  
 end  
 while (toc < t)  
 %do nothing  
 end  
end  
set(handles.plotButton, 'Enable', 'on');