E206 Final Project : Tower of Hanoi

Diana Mar, Shreyasha Paudel

May 8, 2014

Contents

1 Project Statement **1**

2 System Kinematics and Dynamics **1**
2.1 Kinematics . . . 1
2.2 Dynamics . . . 3

3 Feedback Linearization **4**

4 Sliding Control **4**

5 Parameter Adaptive Control **5**

6 Simulink Model **7**

7 Results **9**
7.1 Feedback Linearization . . . 9
7.2 Sliding Control . . . 11
7.3 Parameter Adaptive Control . . . 14

8 Discussion **17**
8.1 Feedback Linearization . . . 17
8.2 Sliding Control . . . 17
8.3 Parameter Adaptive Control . . . 17

9 Conclusion **18**

List of Figures

1 Sequence of moves implemented for end effector grips . . . 1
2 Robot arm model . . . 2
3 Robot arm model showing masses and torques . . . 3
4 Simulink Model of Feedback Linearization Control . . . 7
5 Simulink Model of the overall sliding controller . . . 7
6 Simulink Model of implementation of sliding control . . . 8
7 Simulink Model of overall parameter adaptive controller . . . 8
8 Simulink Model of implementation of parameter adaptive control . . . 9
9 Output plot showing trajectory of end effector in operational space when subjected to just feedback linearized control with nominal parameter values . . . 10
10 Output plot showing torque output from feedback linearization control with nominal parameter values . . . 10
11 Output plot showing trajectory of end effector in operational space when subjected to sliding control with nominal parameter values . . . 12
12 Control Torque versus time outputted by the sliding controller with nominal parameter values 12
13 Plots showing desired and actual θ and end effector position when the arm is subjected to sliding control with nominal parameter values . . . 13
14 Output plot showing trajectory of end effector in operational space when subjected to parameter adaptive control with nominal parameter values . . . 14
15 Control Torque versus time outputted by the parameter adaptive controller with nominal parameter values . . . 15
16 Plots showing desired and actual θ and end effector position when the arm is subjected to parameter adaptive control with nominal parameter values . . . 16
1 Screenshot of GUI Layout . . . 19
2 Simulink Model of robot Arm . . . 27

3 Simulink Model of trajectory generator 27

List of Tables

1 Effects of Parameter Uncertainty in Feedback Linearization Simulation 11
2 Effects of U in Sliding control for perfect estimation 13
3 Effects of parameter uncertainty in Sliding control with U = 5 14
4 Effects of parameter uncertainty in parameter adaptive control with $K_\pi = 100I$ 17

1 Project Statement

In this project the team simulated the control of the end effector of a two link planar robot arm to carry out a subset of sequence of moves for a Tower of Hanoi game. The desired trajectory is the sequence of motions necessary to move one disk between an two rods. The starting position, and the sequence of moves are shown below:

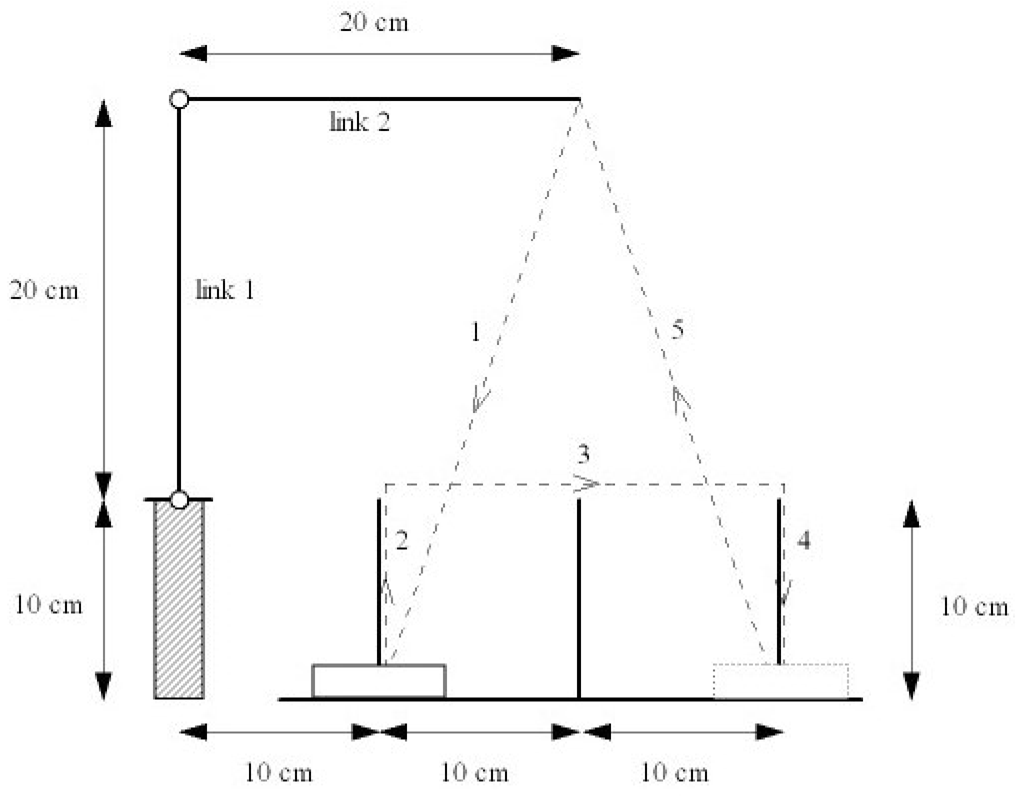


Figure 1: Sequence of moves implemented for end effector grips

Both sliding and parameter adaptive control were implemented and a MATLAB GUI was designed to plot the desired and actual trajectories, applied torques with user estimated system paramters. This paper describes the design process and the results observed in the simulation.

2 System Kinematics and Dynamics

2.1 Kinematics

The kinematics of the two-link robot arm can be modeled as:

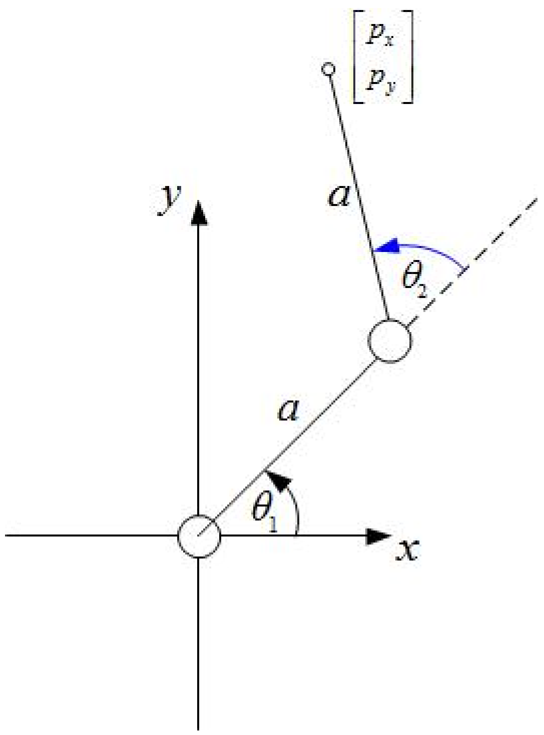


Figure 2: Robot arm model

where a is link length.

The kinematics relates angles θ_1 and $theta_2$ to the position of the end effector $\begin{bmatrix} p_x \\ p_y \end{bmatrix}$. This is given by the relation

$$\begin{aligned} p_x &= acos(\theta_1) + acos(\theta_1 + \theta_2) && (1) \\ p_y &= asin(\theta_1) + asin(\theta_1 + \theta_2); && (2) \end{aligned}$$

Conversely, the angles can be found from the position as shown:

$$\begin{aligned} \alpha &= tan^{-1}(\frac{p_x}{p_y}) \\ \beta &= cos^{-1}(\frac{p_x^2 + p_y^2}{2a\sqrt{(}p_x^2 + p_y^2)}) \\ \theta_1 &= \alpha + \beta && (3) \end{aligned}$$

$$\begin{aligned} cos\theta_2 &= \frac{p_x^2 + p_y^2 - a^2 - a^2}{2a^2} \\ sin\theta_2 &= -\sqrt{(}1 - cos^2\theta_2) \\ \theta_2 &= tan^{-1}(\frac{sin\theta_2}{cos\theta_2}) && (4) \end{aligned}$$

Finally, system kinematics is used to define a straightline trajectory that moves the end-effector to desired positions. The path variable was a quintic polynomial shown below.

$$s(t) = a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \tag{5}$$
$$s(0) = 0$$
$$s(t_f) = \left|\left| \begin{bmatrix} p_{xf} \\ p_{yf} \end{bmatrix} - \begin{bmatrix} p_{xi} \\ p_{yi} \end{bmatrix} \right|\right|$$

where, t_f is the time required to complete the move.

2.2 Dynamics

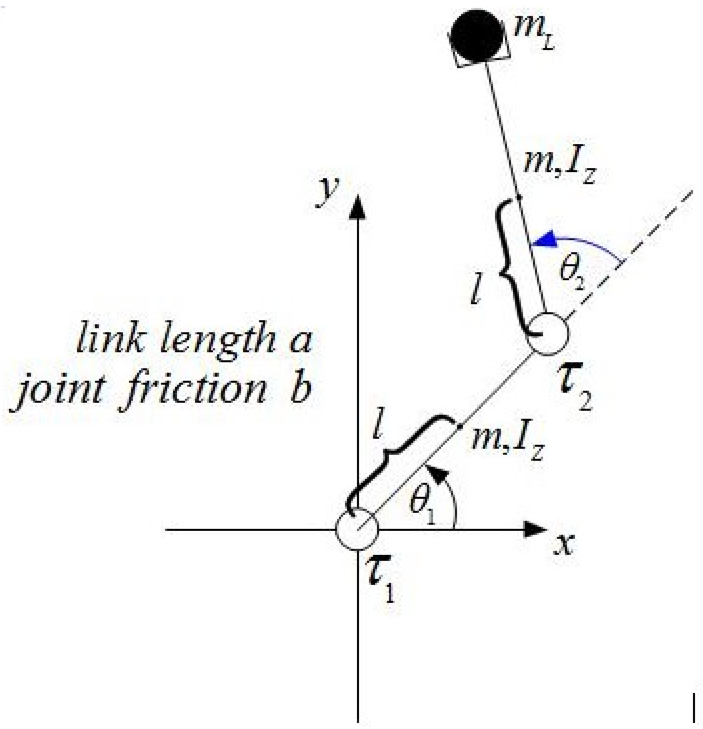


Figure 3: Robot arm model showing masses and torques

The torque applied to a two degree of freedom robot arm is described by:

$$\tau = M(\theta)\ddot{\theta} + b(\theta, \dot{\theta})\dot{\theta} + B\dot{\theta} + g(\theta) \tag{6}$$

where,

$$M(\theta) = \begin{bmatrix} m(a^2 + 2l^2 + 2alc_2) + 2I_z + 2m_La^2(1 + c_2) & ml(l + ac_2) + I_z + m_La^2(1 + c_2) \\ ml(l + ac_2) + I_z + m_La^2(1 + c_2) & ml^2 + I_z + m_La^2 \end{bmatrix}$$

$$b(\theta, \dot{\theta}) = \begin{bmatrix} -2(mal + m_La^2)s_2\dot{\theta}_2 & -(mal + m_La^2)s_2\dot{\theta}_2 \\ (mal + m_La^2)s_2\dot{\theta}_1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix}$$

$$g(\theta) = \begin{bmatrix} (m(a + l) + m_La)gc_1 + (ml + m_La)gc_{12} \\ (ml + m_La)gc_{12} \end{bmatrix}$$

The variables in the above matrix definition are the system parameters given by

$$\begin{aligned}
\text{Link Mass, } m &= m_1 = m_2 = 0.25kg \\
\text{Load mass, } m_L &= 0.1kg \\
\text{Link Length, } a &= 0.2m \\
\text{Link Inertia, } I_z &= 0.01kg/m^2 \\
\text{Center of Mass, } l &= \frac{a}{2} \\
\text{Joint friction, } b &= 0.01Nms \\
c_1 &= cos(\theta_1) \\
c_2 &= cos(\theta_2) \\
c_{12} &= cos(\theta_1 + \theta_2) \\
s_1 &= sin(\theta_1) \\
s_2 &= sin(\theta_2) \\
s_{12} &= sin(\theta_1 + \theta_2)
\end{aligned}$$

3 Feedback Linearization

Feedback Linearization uses a non-linear control law that cancels the nonlinearity in the system. The general control law is

$$u = \frac{1}{c}(v - f) \tag{7}$$

The dynamics of two-link robotic arm mnaipulator presented in Equation 6 can be feedback linearized as shown. First define the error state variable as

$$x = \begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} \theta_r - \theta \\ \dot{\theta}_r - \dot{\theta} \end{bmatrix}$$

Then, the manipulator dynamics can be rewritten as

$$\tau = M(\theta)(\ddot{\theta} - \ddot{e}) + c(\theta, \dot{\theta}) \tag{8}$$

And the non-linear control law becomes

$$u = M(\theta)(\ddot{\theta}_r + v) + c(\theta, \ddot{\theta}) \tag{9}$$

where,

$$v = Kx = K_p e + K_d \dot{e}$$

and $K = [K_p]K_d]$ is the linear feedback gain matrix. K_p and K_d are chosen to provide a desired overshoot and settling time. For desired second order response,

$$K_p = \omega_n^2 I$$

$$K_d = 2\zeta\omega_n I$$

We implemented a feedback linearization control with $\zeta = 1$ and $\omega_n = \frac{4.6}{t_s}$ where t_s is the settling time input by the user. The Simulink model for the feedback linearization block is in Appendix.

4 Sliding Control

Sliding control forces the tracking error of the system onto a switching surface that changes the direction of the control signal when the state trajectory crosses it. The error will then be driven to zero as the system follows the linear dynamics of the switching surface. The method and derivation of the control law is described in this section.

Using estimated parameters of the system, the altered control law becomes:

$$u = \hat{M}(q)(\ddot{q}_r + v) + \hat{c}(q, \dot{q}) \tag{10}$$

Subsitituting u for τ in the robot arm dynamics then gives:

$$\ddot{e} = -v + \eta(q, \dot{q}, q_r, \dot{q}_r, \ddot{q}_r) \tag{11}$$

where $\eta(q, \dot{q}, q_r, \dot{q}_r, \ddot{q}_r) = (I - M^{-1}\hat{M})(\ddot{q}_r + v) + M^{-1}(c - \hat{c})$, the parameter "uncertainty".

The input to the feedback linearized system is then defined:

$$v = Kx + \omega = K_p e + K_d \dot{e} + \omega \tag{12}$$

where ω is the sliding control vector that controls the parameter uncertainty.
Now, the system in state space form is:

$$\dot{x} = Fx + G(\eta - \omega) \tag{13}$$

where $F = \begin{bmatrix} 0 & I \\ -K_p & -K_d \end{bmatrix}; G = \begin{bmatrix} 0 \\ I \end{bmatrix}$
Then we choose the quadratic Lyapunov function $V = x^T P x$ and show that

$$\dot{V} = -x^T x + 2G^T P x(\eta - \omega) \tag{14}$$

where P is a positive definite symmetric matrix that satisfies the Lyapunov equation

$$F^T P + PF = -I \tag{15}$$

and can be found by using Matlab's lyap() function.The switching surface is then chosen as: $s = G^T P x$.
Now, the sliding control vector is defined to be:

$$\omega = U sgn(s)$$

where U is the sliding control gain vector.
$\dot{V} \leq 0$ for Lyapunov stability occurs when $U \geq |\eta|_{max}$. For this project, several values of U were chosen and a $U = 5$ produced the smallest errors with least chattering. Table 2 in Section 7.2 shows the error results associated with different values of U.

5 Parameter Adaptive Control

Parameter adaptive control involves estimating the plant model in real time based on the state feedback signals and a parameter update law. We start with the robot dynamics from feedback linearization:

$$\tau = M(q)\ddot{q} + b(q, \dot{q})\dot{q} + g(q)$$

Then we define an "augmented" reference velocity vector

$$\dot{q}_{ra} = \dot{q}_r + K_d^{-1} K_p e \tag{16}$$

and an augmented state vector

$$\sigma = \dot{e} + Kd^{-1} K_p e \tag{17}$$

The control law for parameter adaptive control becomes:

$$u = \hat{M}(q)\ddot{q}_{ra} + \hat{b}(q, \dot{q})\dot{q}_{ra} + g(q) + v = Y(q, \dot{q}, \dot{q}_{ra}, \ddot{q}_{ra})\hat{\pi} + v \tag{18}$$

where $\hat{\pi} = \begin{bmatrix} \hat{I_z} \\ \hat{m} \\ \hat{m_L} \\ \hat{b} \end{bmatrix}$ is a vector of estimated system parameters provided by the user and $Y(q, \dot{q}, \dot{q}_{ra}, \ddot{q}_{ra})$ is a matrix of configuration dependent joint variables:

$$Y = \begin{bmatrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \end{bmatrix}$$

where

$$\begin{aligned}
y_{11} &= 2\ddot{\theta}_{ra1} + \ddot{\theta}_{ra2} \\
y_{12} &= (a^2 + 2l^2 + 2alc_2)\ddot{\theta}_{ra1} + l(l + ac_2)\ddot{\theta}_{ra2} - als_2(2\dot{\theta}_{ra1} + \dot{\theta}_{ra2})\dot{\theta}_2 + g[(a + l)c_1 + lc_{12}] \\
y_{13} &= a^2(1 + c_2)(2\ddot{\theta}_{ra1} + \ddot{\theta}_{ra2}) - a^2 s_2(2\dot{\theta}_{ra1} + \dot{\theta}_{ra2})\dot{\theta}_2 + ga[c_1 + c_{12}] \\
y_{14} &= \dot{\theta}_{ra1} \\
y_{21} &= \ddot{\theta}_{ra1} + \ddot{\theta}_{ra2} \\
y_{22} &= l(l + ac_2)\ddot{\theta}_{ra1} + l^2\ddot{\theta}_{ra2} + als_2\dot{\theta}_1\dot{\theta}_{ra1} + glc_{12} \\
y_{23} &= a^2(1 + c_2)\ddot{\theta}_{ra1} + a^2\ddot{\theta}_{ra2} + a^2 s_2\dot{\theta}_1\dot{\theta}_{ra1} + gac_{12} \\
y_{24} &= \dot{\theta}_{ra2}
\end{aligned} \tag{19}$$

Substituting the outer loop tracking control $v = K_p e + K_d \dot{e}$ gives a nonlinear first order system description

$$M(q)\dot{\sigma} + [b(q, \dot{q}) + K_d]\sigma = -Y\tilde{\pi}$$

where $\tilde{\pi} = \hat{\pi} - \pi$ is the vector of system parameter errors.
Now we can choose a Lyapunov function

$$V = \frac{1}{2}\sigma^T P\sigma + \frac{1}{2}\tilde{\pi}^T K_\pi \tilde{\pi} \tag{20}$$

and show that

$$\dot{V} = -\sigma^T K_d\sigma + \tilde{\pi}^T(K_\pi\dot{\tilde{\pi}} - Y^T\sigma) \tag{21}$$

For global asymptotic stability, $\dot{V} \leq 0$ when $K_\pi\dot{\tilde{\pi}} - Y^T\sigma = 0$ or

$$\dot{\tilde{\pi}} = \dot{\hat{\pi}} = K_\pi^{-1}Y^T\sigma \tag{22}$$

This is the adaptive parameter update equation. K_π is a diagonal adaptive gain matrix weighted according to the desired relative update speeds for each parameter (I_z, m, m_L, b). For this project, $K_\pi = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}$.

6 Simulink Model

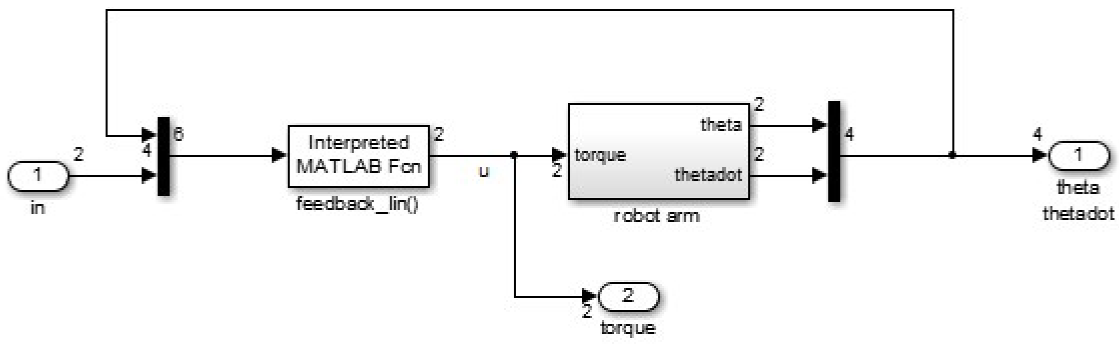


Figure 4: Simulink Model of Feedback Linearization Control

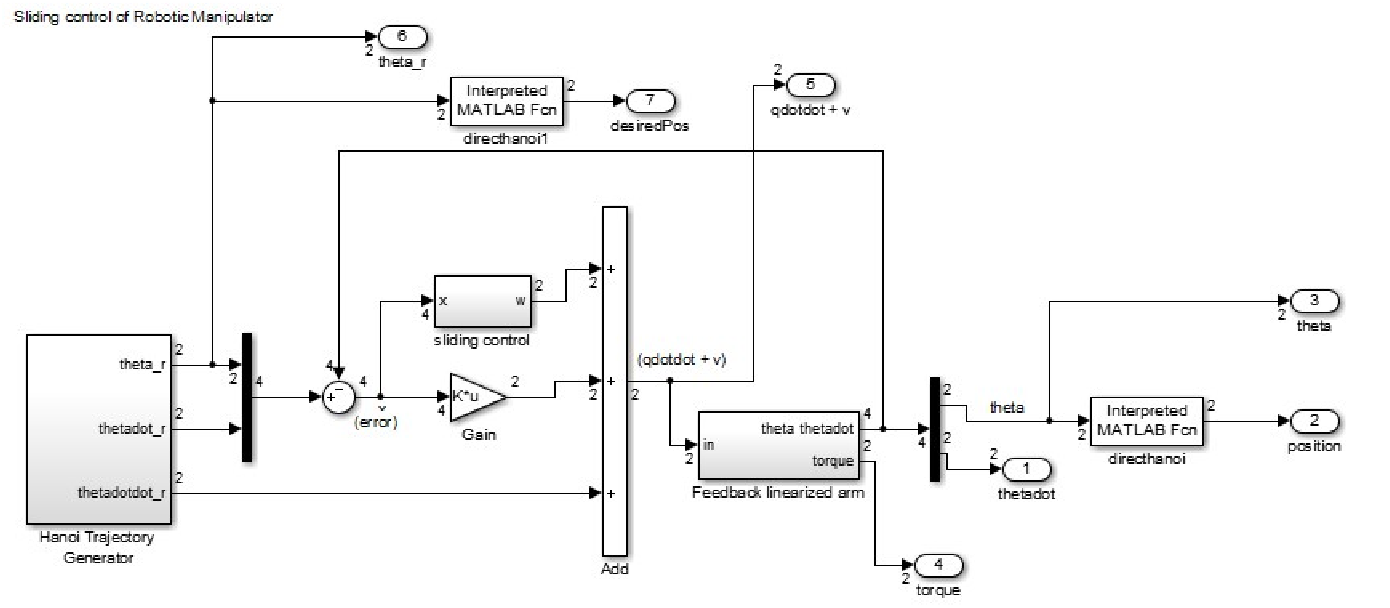


Figure 5: Simulink Model of the overall sliding controller

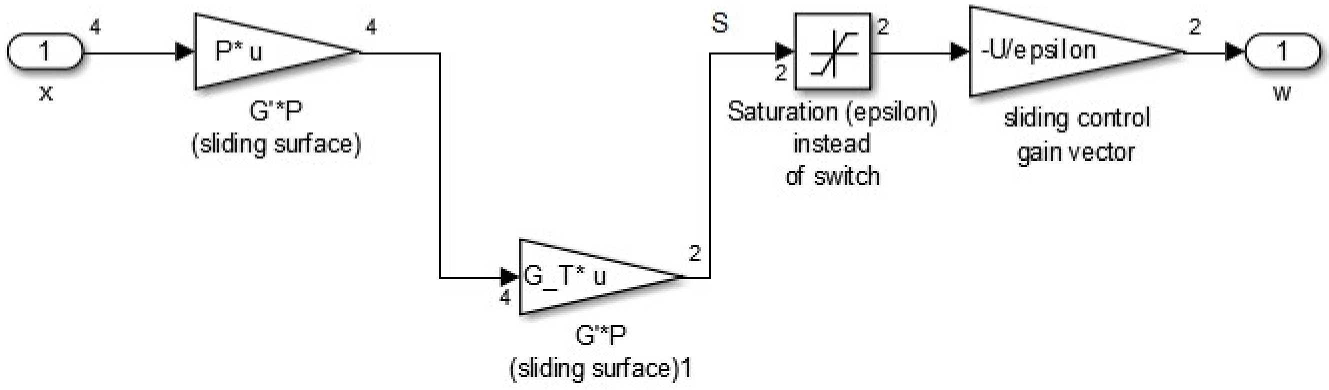


Figure 6: Simulink Model of implementation of sliding control

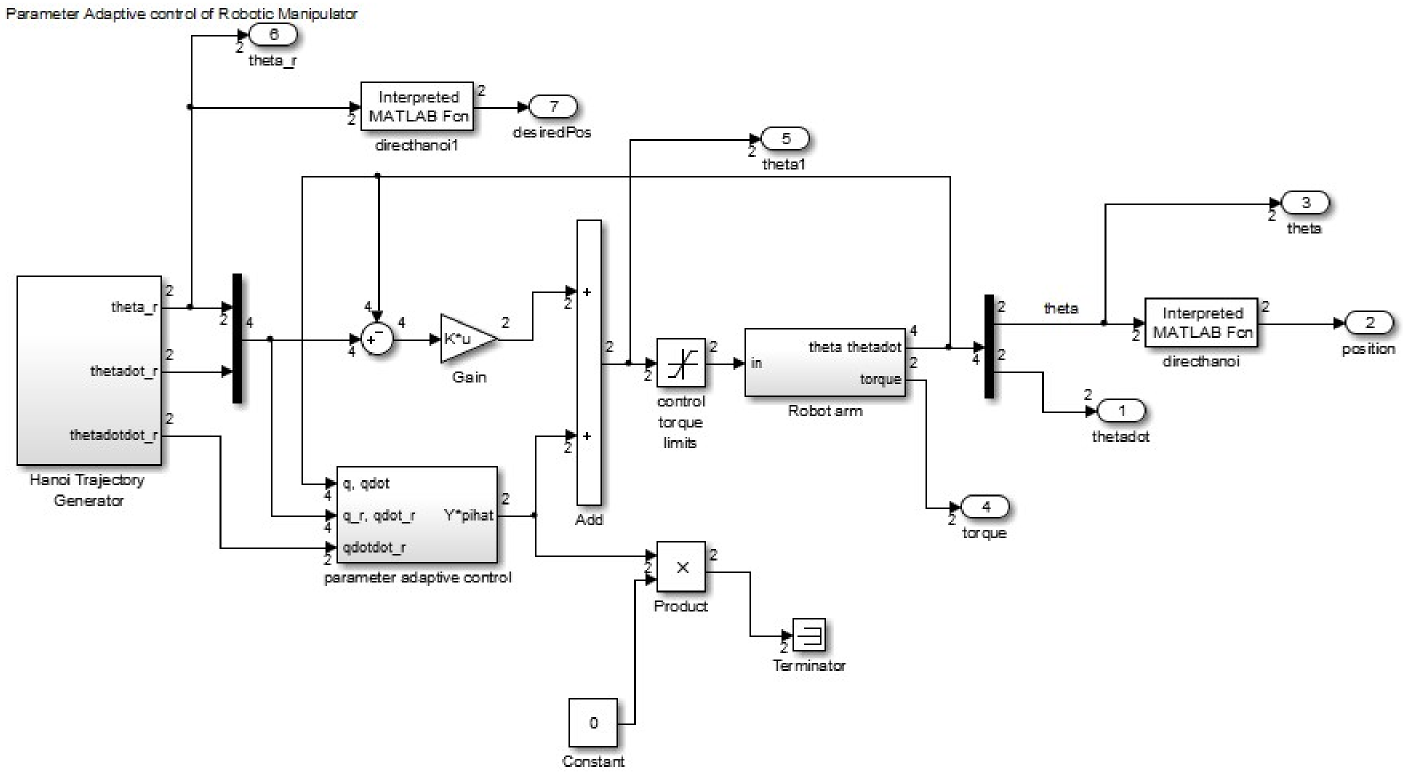


Figure 7: Simulink Model of overall parameter adaptive controller

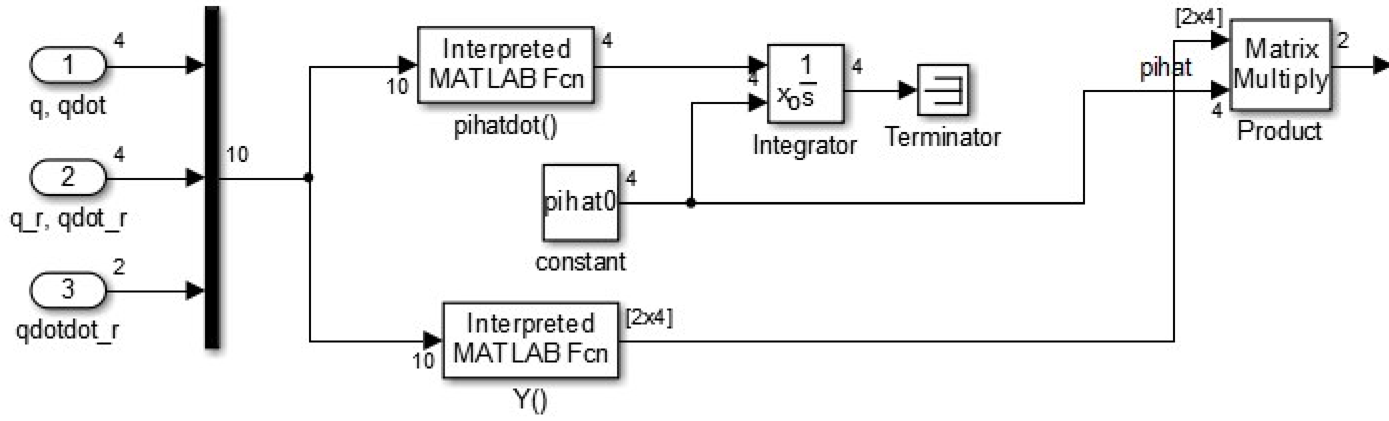


Figure 8: Simulink Model of implementation of parameter adaptive control

7 Results

7.1 Feedback Linearization

A simple feedback linearization controller was implemented by setting U = 0 in the sliding control simulink model. The plots in Figure 9 and 10 show the output from the controller when the parameter estimates equaled nominal values. The move time was set to be 1s and the settling time was set to be 0.1 s in these simulations. As expected, the controller is almost perfect in this case and the trajectory is almost the same as desired straightline trajectory.

Table 1 presents the effect of changing parameter estimate on the controller performance. It is evident that the errors remain fairly small even in this case. The performance is discussed in more detail in Section 8.

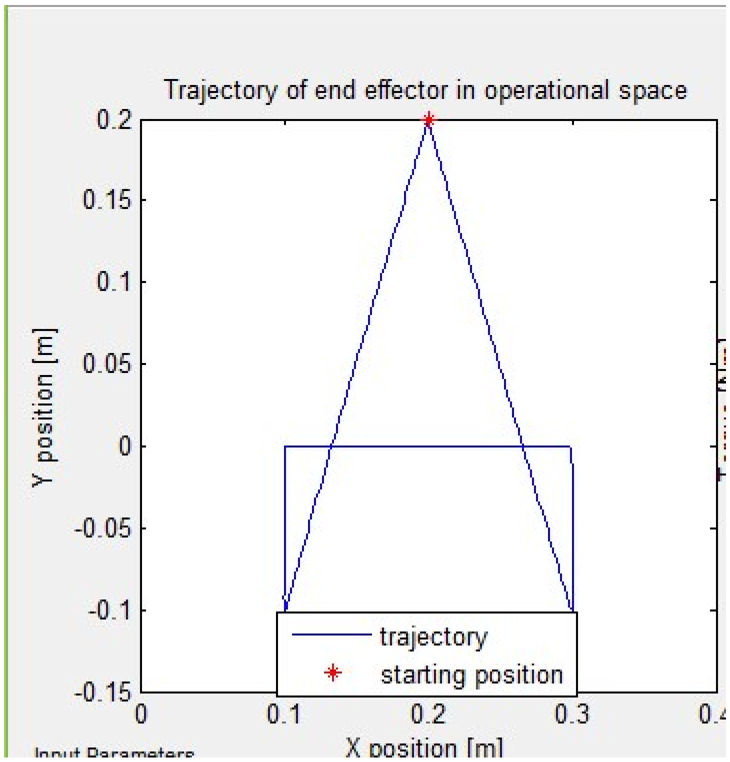


Figure 9: Output plot showing trajectory of end effector in operational space when subjected to just feedback linearized control with nominal parameter values

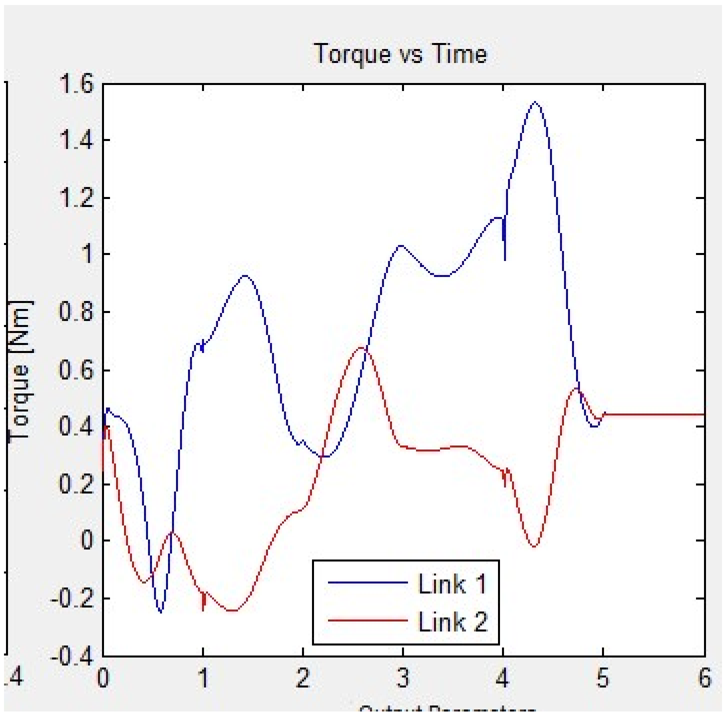


Figure 10: Output plot showing torque output from feedback linearization control with nominal parameter values

Table 1: Effects of Parameter Uncertainty in Feedback Linearization Simulation

Parameter varied	ISE PosX	ISE PosY	IS Torque1	IS Torque2	Max Dev Move2	Max Dev Move4
None	$2.93 * 10^{-5}$	$4.56 * 10^{-4}$	352.09	69.71	$7.12 * 10^{-4}$	$1.02 * 10^{-4}$
$t_s = 0.1$	$2.93 * 10^{-5}$	$4.56 * 10^{-4}$	352.09	69.71	$7.12 * 10^{-4}$	$1.02 * 10^{-4}$
$t_s = 0.2$	$8.75 * 10^{-4}$	0.01	351.75	69.33	0.003	0.001
$t_s = 0.5$	0.038	0.333	349.00	70.86	0.016	0.009
$m_L = 0.05$	$3.69 * 10^{-5}$	$5.02 * 10^{-4}$	352.32	69.65	$7.12 * 10^{-4}$	$2.12 * 10^{-4}$
$m_L = 0.1$	$2.93 * 10^{-5}$	$4.56 * 10^{-4}$	352.09	69.71	$7.12 * 10^{-4}$	$1.02 * 10^{-4}$
$m_L = 0.195$	$3.99 * 10^{-5}$	$5.47 * 10^{-4}$	351.67	69.74	$7.12 * 10^{-4}$	$1.98 * 10^{-4}$
$b = 0.1$	$2.36 * 10^{-4}$	$4.81 * 10^{-4}$	352.10	69.58	$6.46 * 10^{-4}$	$3.01 * 10^{-4}$
$b = 0.2$	$2.93 * 10^{-5}$	$4.56 * 10^{-4}$	352.09	69.71	$7.12 * 10^{-4}$	$1.02 * 10^{-4}$
$b = 0.4$	$1.42 * 10^{-3}$	$1.09 * 10^{-3}$	352	69.90	$9.65 * 10^{-4}$	$6.71 * 10^{-4}$
$I_z = 0.005$	$2.76 * 10^{-3}$	$2.53 * 10^{-3}$	357.58	69.03	0.001	$8.97 * 10^{-4}$
$I_z = 0.01$	$2.93 * 10^{-5}$	$4.56 * 10^{-4}$	352.09	69.71	$7.12 * 10^{-4}$	$1.02 * 10^{-4}$
$I_z = 0.025$	$5.02 * 10^{-4}$	$3.69 * 10^{-4}$	343.01	77.47	$6.67 * 10^{-4}$	$4.03 * 10^{-4}$

7.2 Sliding Control

The plots in Figure 11, 12, and 13 show the output from the controller when the parameter estimates equaled nominal values. The move time was set to be 1s and the settling time was set to be 0.1 s in these simulations. As expected, the controller is almost perfect in this case and the trajectory is almost the same as desired straightline trajectory.This is especially clear from Figure 13 where the output θ and position conincide perfectly with the desired ones.

Table 2 shows the effect of changing U in the controller output. By trial and error, it was found that $U = 5$ gave the optimal performance with minimum chattering and it is used in rest of the simulation. Table 3 presents the effect of changing parameter estimate on the controller performance. It is evident that the errors remain fairly small and the controller is robust. The performance is discussed in more detail in Section 8.

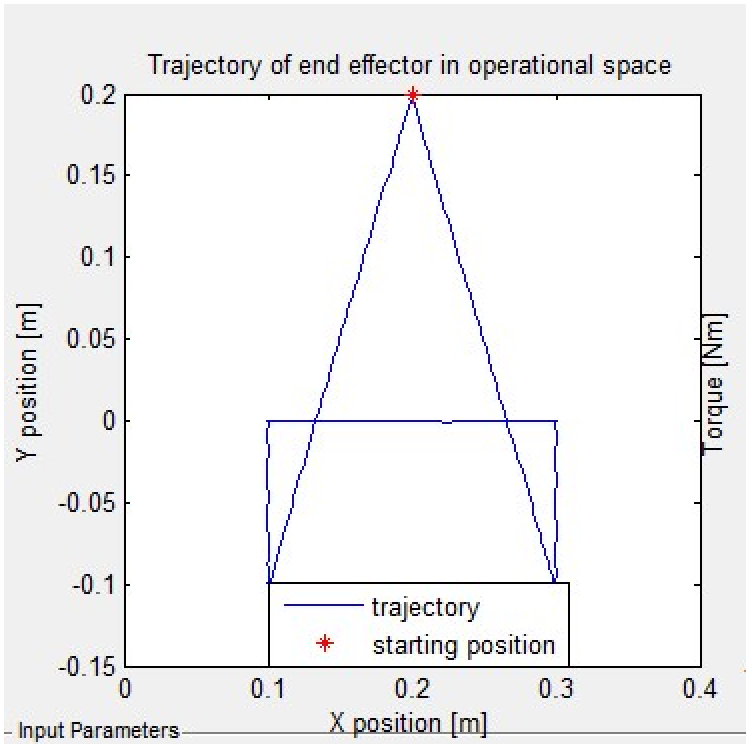


Figure 11: Output plot showing trajectory of end effector in operational space when subjected to sliding control with nominal parameter values

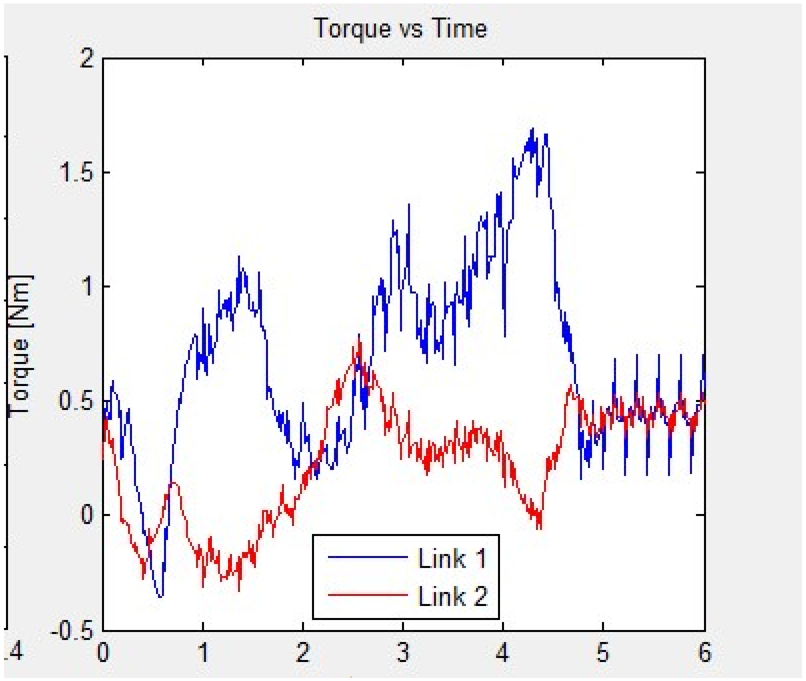


Figure 12: Control Torque versus time outputted by the sliding controller with nominal parameter values

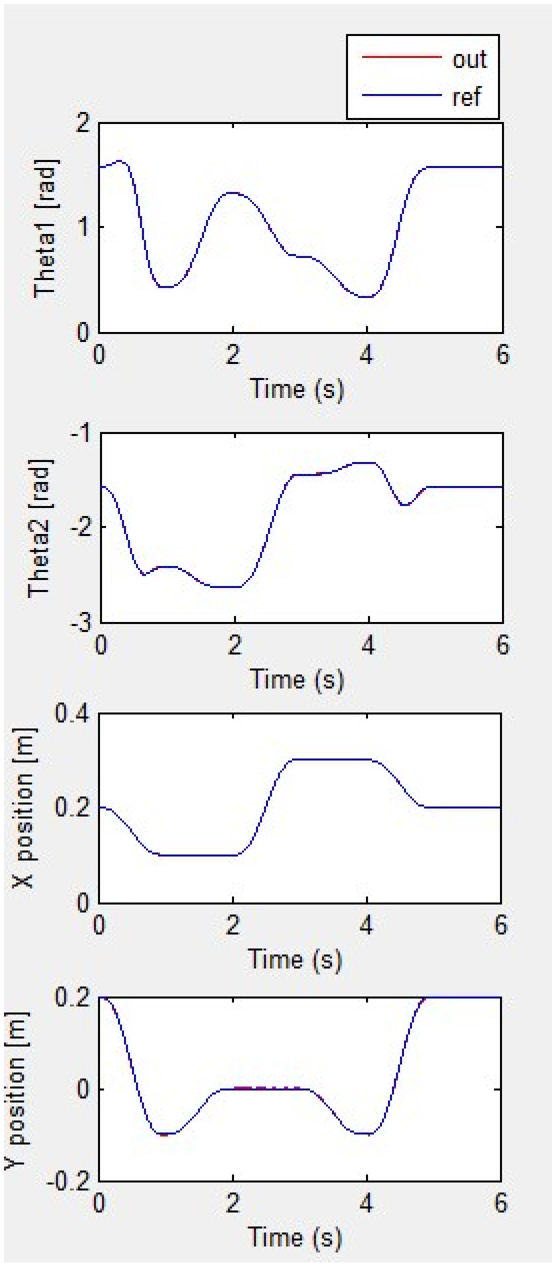


Figure 13: Plots showing desired and actual θ and end effector position when the arm is subjected to sliding control with nominal parameter values

Table 2: Effects of U in Sliding control for perfect estimation

U	ISE PosX	ISE PosY	IS Torque1	IS Torque2	Max Dev Move2	Max Dev Move4
0	$2.93 * 10^{-5}$	$4.56 * 10^{-4}$	352.09	69.71	$7.12 * 10^{-4}$	$1.02 * 10^{-4}$
5	$6.16 * 10^{-5}$	$5.21 * 10^{-4}$	372.38	69.91	$7.1 * 10 * -4$	$3.97 * 10^{-4}$
10	$1.78 * 10^{-4}$	$9.01 * 10^{-4}$	412.23	76.39	0.0012	$9.38 * 10^{-4}$
20	$7.91 * 10^{-4}$	$2.19 * 10^{-3}$	473.91	95.48	0.0026	0.0017

Table 3: Effects of parameter uncertainty in Sliding control with U = 5

Parameter varied	ISE PosX	ISE PosY	IS Torque1	IS Torque2	Max Dev Move2	Max Dev Move4
None	$6.16 * 10^{-5}$	$5.21 * 10^{-4}$	372.38	69.91	$7.1 * 10^{-4}$	$3.97 * 10^{-4}$
$m_l = 0.05$	$7.24 * 10^{-5}$	$5.56 * 10^{-4}$	380.06	69.78	$7.28 * 10^{-4}$	$5.89 * 10^{-4}$
$m_L = 0.1$	$6.16 * 10^{-5}$	$5.21 * 10^{-4}$	372.38	69.91	$7.1 * 10^{-4}$	$3.97 * 10^{-4}$
$m_L = 0.1975$	$7.09 * 10^{-5}$	$5.87 * 10^{-4}$	374.43	71.25	$7.26 * 10^{-4}$	$5.1 * 10^{-4}$
$b = 0.1$	$2.82 * 10^{-4}$	$5.82 * 10^{-4}$	364.48	69.05	$8.65 * 10^{-4}$	$7.65 * 10^{-4}$
$b = 0.2$	$6.16 * 10^{-5}$	$5.21 * 10^{-4}$	372.38	69.91	$7.1 * 10^{-4}$	$3.97 * 10^{-4}$
$b = 0.4$	0.0015	0.0011	381.60	69.20	0.0014	0.001
$I_z = 0.005$	$1.01 * 10^{-4}$	$1.21 * 10^{-4}$	378.95	70.40	$7.3 * 10^{-4}$	$4.63 * 10^{-4}$
$I_z = 0.01$	$6.16 * 10^{-5}$	$5.21 * 10^{-4}$	372.38	69.91	$7.1 * 10^{-4}$	$3.97 * 10^{-4}$
$I_z = 0.025$	$5.56 * 10^{-5}$	$1.99 * 10^{-4}$	381.05	73.82	$4.11 * 10^{-4}$	$3.32 * 10^{-4}$

7.3 Parameter Adaptive Control

The plots in Figure 14, 15, and 16 show the output from the controller when the parameter estimates equaled nominal values. The move time was set to be 1s and the settling time was set to be 0.1 s in these simulations.

Table 1 presents the effect of changing parameter estimate on the controller performance. The performance is discussed in more detail in Section 8.

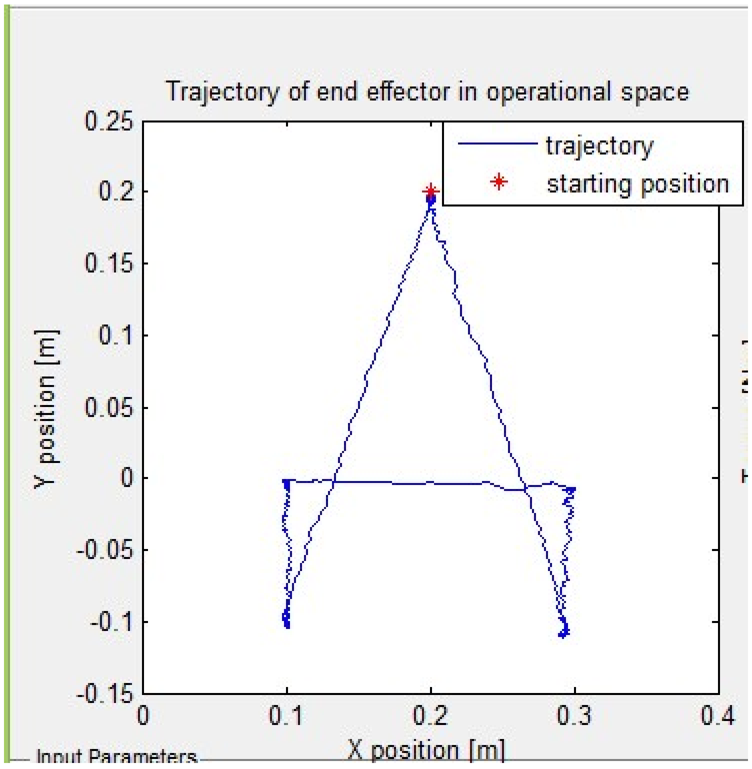


Figure 14: Output plot showing trajectory of end effector in operational space when subjected to parameter adaptive control with nominal parameter values

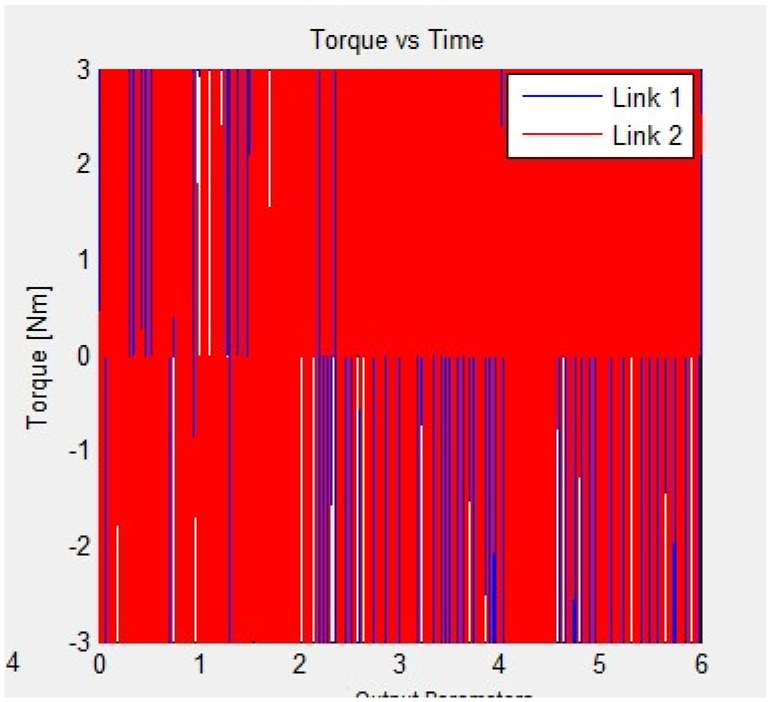


Figure 15: Control Torque versus time outputted by the parameter adaptive controller with nominal parameter values

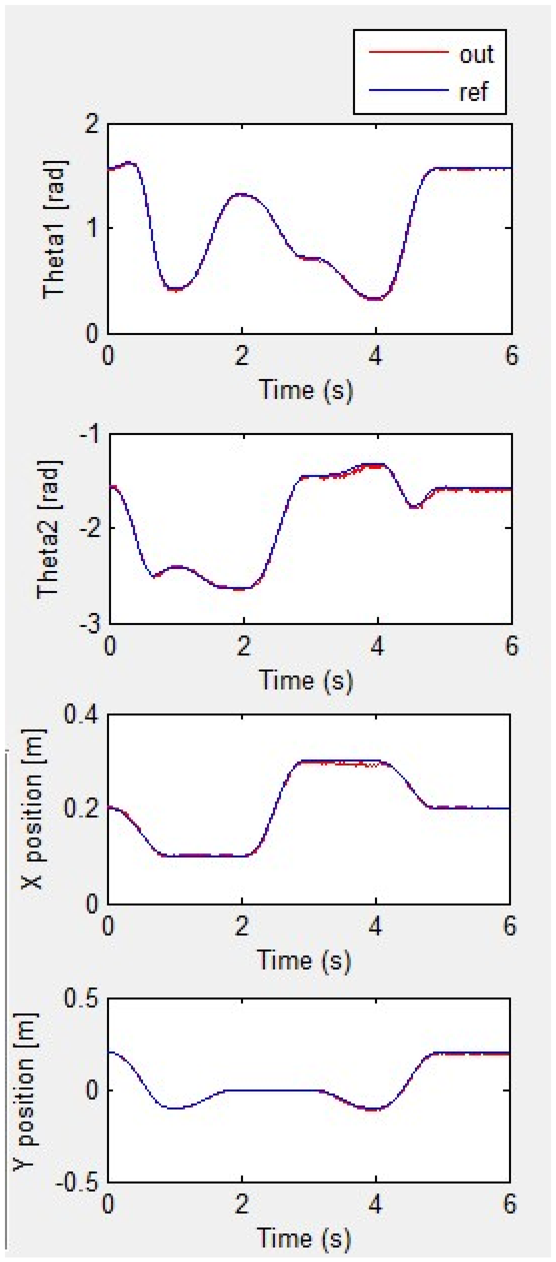


Figure 16: Plots showing desired and actual θ and end effector position when the arm is subjected to parameter adaptive control with nominal parameter values

Table 4: Effects of parameter uncertainty in parameter adaptive control with $K_\pi = 100I$

Parameter varied	ISE PosX	ISE PosY	IS Torque1	IS Torque2	Max Dev Move2	Max Dev Move4
None	0.0071	0.0201	5207.17	5268.78	0.0039	0.0103
$m_L = 0.05$	0.007	0.018	5268.48	5271.69	0.0039	
$m_L = 0.1$	0.0071	0.0201	5207.17	5268.78	0.0039	0.0103
$m_L = 0.2$	0.0066	0.0203	5227.84	5298.2	0.0045	0.0092
$b = 0.1$	0.0075	0.0192	5237.31	5286.35	0.0047	0.0099
$b = 0.2$	0.0071	0.0201	5207.17	5268.78	0.0039	0.0103
$b = 0.4$	0.0080	0.0192	5276.06	5349.33	0.0042	0.0106
$I_z = 0.005$	0.0076	0.0213	5242.76	5290.92	0.0038	0.0093
$I_z = 0.01$	0.0071	0.0201	5207.17	5268.78	0.0039	0.0103
$I_z = 0.025$	0.0092	0.0197	5216.55	5319.4	0.0040	0.0111

8 Discussion

8.1 Feedback Linearization

As seen in Table 1, the nominal parameter values yielded the smallest tracking errors, as expected since knowing the exact dynamics of the system will force the system to behave like a second order system for each move. When the parameters were estimated and varied, we see that a shorter settling time significantly improved the tracking ability (lower ISE's). Varying estimations of the load mass did not affect the system very much. On the other hand, changes in the joint friction estimations significantly affected the tracking performance. As expected intuitively, the higher the friction, the larger the tracking errors and deviations in moves 2 and 4. Lastly, estimations of link inertia did not seem to have a large effect on the robot arm's position ISEs. However, an overestimation of I_z lowered the IS of torque in link 1 and increased the IS of torque in link 2. Overall, the position tracking errors were within less than 1 cm as long as the settling time was less than 0.5 seconds.

8.2 Sliding Control

Firstly, as shown in Table 2, increasing the the sliding control gain vector, U, increased the tracking errors and IS control torques. Like the results from feedback linearization, we see that different estimations of the load mass did not significantly change the tracking performance. However, as with feedback linearization, increasing the joint friction estimations had a large effect on the tracking errors: the ISEs in position increased, the IS of the torque in link 1 increased (although not so much of an effect on link 2), and the maximum deviations in moves 2 and 4 increased. Lastly, we see that changes in link inertia did not influence the position errors, but did change the IS of the torques in links 1 and 2. This time, both under- and over-estimations of the link inertia increased the control torque efforts. This is reflected in Figure 12, where the control torque is more noisy than with feedback linearization. There is also a bit of steady state error as a result of chattering along the switching surface, but that did not seem to deter the robot arm from tracking the trajectory. Overall, even with the parameter estimations, sliding control is comparable to feedback linearization.

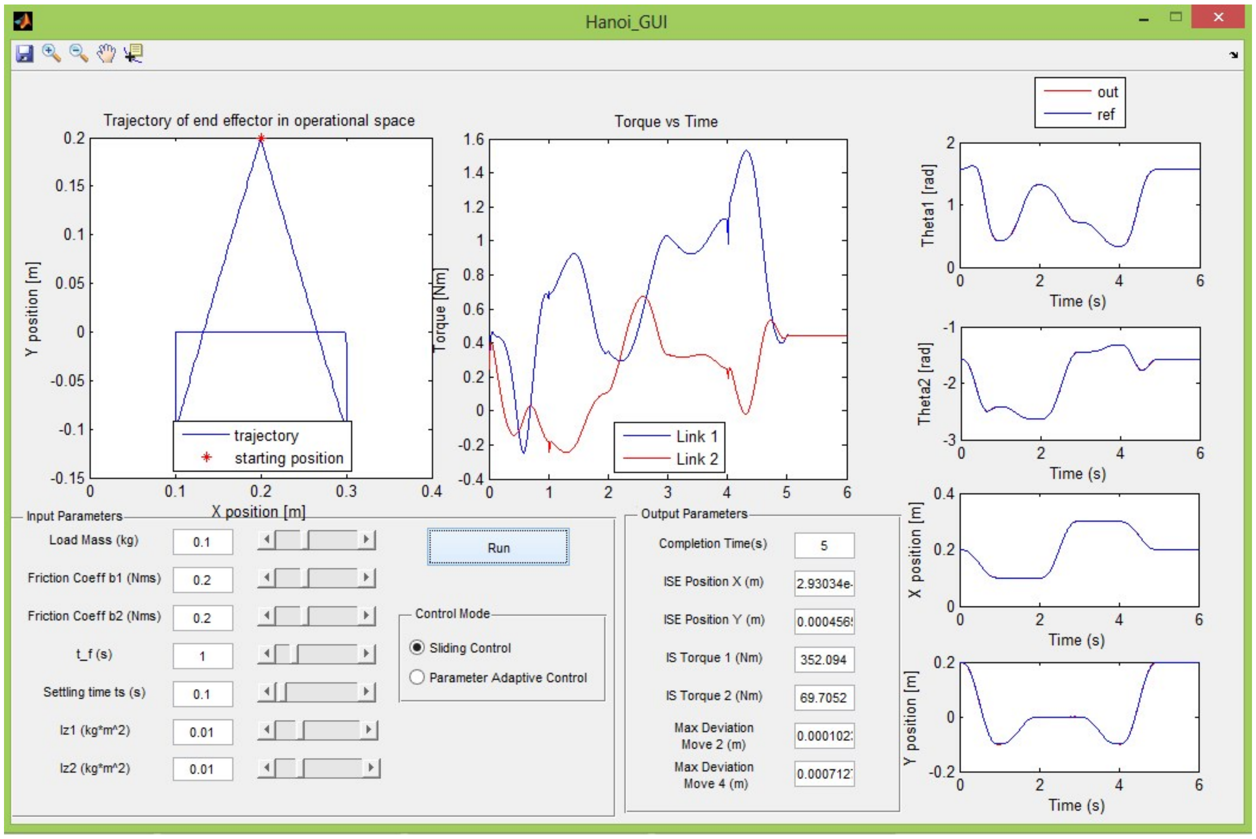
8.3 Parameter Adaptive Control

From Figures 14 and 15 and Table 4, we see that parameter adaptive control's performance did not track as well as feedback linearization and sliding control. The errors and IS of the torques were quite significant, and the control torques were very quickly switching between 3Nm and -3Nm. Additionally, unlike feedback linearization and sliding control where the maximum deviations in moves 2 and 4 were less than 1mm, those of parameter adaptive control were greater than 1mm. We think there was an error in implementing parameter adaptive control, specifically with our function to find Y, but were unable to find the solution.

9 Conclusion

From our results, we have shown that nonlinear control of the robotic arm can be achieved using feedback linearization, sliding control, and, less-so, parameter adaptive control to accommodate parameter estimations. However, when control effort (IS of control torques) is to be minimized while meeting the performance specifications on the speed and position of the end effector, our findings suggest that feedback linearization is the optimal choice of control, followed closely by sliding control (perhaps with a lower control gain vector U to decrease the IS of control torques). Parameter adaptive control was not properly implemented, as it did not result in meeting the performance objectives.

Appendix A: GUI Layout



Input Parameters

Load Mass (kg) 0.1

Friction Coeff b1 (Nms) 0.2

Friction Coeff b2 (Nms) 0.2

t_f (s) 1

Settling time ts (s) 0.1

Iz1 (kg*m^2) 0.01

Iz2 (kg*m^2) 0.01

Control Mode

[✓ Sliding Control] Parameter Adaptive Control

Output Parameters

Completion Time(s) 5

ISE Position X (m) 2.93034e-

ISE Position Y (m) 0.000456

IS Torque 1 (Nm) 352.094

IS Torque 2 (Nm) 69.7052

Max Deviation Move 2 (m) 0.000102

Max Deviation Move 4 (m) 0.000712

Figure 1: Screenshot of GUI Layout

Appendix B: MATLAB Code

HANOI GUI run

```
% --- Executes on button press in Run.
function Run_Callback(hObject, eventdata, handles)
% hObject    handle to Run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global m mL mLuser a Iz1 Iz2 Iz l b1 b2 b g; %system parameters
global tf ts omega_n zeta ; % model parameters
global Kp Kd;

m = 0.25; %link mass
a = 0.2; %link length
l = a/2;
g= 9.81; %gravity

%Get values from the GUI
mL = str2double(get(handles.mL_box, 'String')); %end effector mass
mLuser = mL;
b1 = str2double(get(handles.b1_box, 'String')); %joint/actuator friction
b2 = str2double(get(handles.b2_box, 'String'));
Iz1 = str2double(get(handles.Iz1_box, 'String'));
Iz2 = str2double(get(handles.Iz2_box, 'String'));

Iz = Iz1;
b = b1;

tf = str2double(get(handles.tf_box, 'String')); %move time
ts = str2double(get(handles.ts_box, 'String')); %settling time

%Calculate omega, zeta and gain values
zeta = 1; % No overshoot
omega_n = 4.6/(ts * zeta);

Kp = omega_n^2.*eye(2);
Kd = 2*zeta*omega_n.*eye(2);
K = [Kp Kd];

%Sliding control calculations
F = [zeros(2) eye(2); -Kp -Kd];
G = [zeros(2); eye(2)];
G_T = G';
P = lyap(F,eye(4));
epsilon = 0.1; %play with epsilon value
U = 5;

%Parameter control calculations
pihat0 = [Iz; m; mL; b];

%Assign values to base
assignin('base','K', K);    % Feedback gain
assignin('base','m', m);
assignin('base','a', a);
assignin('base','Iz', Iz);
assignin('base','Iz1', Iz1);
assignin('base','Iz2', Iz2);
assignin('base','l', l);
assignin('base','g', g);
assignin('base','mL', mL);
assignin('base','b1', b1);
assignin('base','b2', b2);
assignin('base','tf', tf);
assignin('base','ts', ts);
assignin('base','epsilon',epsilon);
assignin('base','U',U);
assignin('base','P',P);
assignin('base','G_T',G_T);
assignin('base', 'pihat0', pihat0);

tspan = 6 * tf;
selected = get(get(handles.loopControl, 'SelectedObject'),'Tag');
if (strcmp(selected, 'slidingControl') == 1)
    [tout,~,yout]=sim('sliding_control', tspan);
end


if (strcmp(selected, 'parameterAdaptive') == 1)
    [tout,~,yout]=sim('ParameterAdaptive_control', tspan);
end

assignin('base', 'tout', tout);

%plot Trajectory
axes(handles.trajPlot)
plot(yout(:,3), yout(:,4));
title('Trajectory of end effector in operational space');
xlabel('X position [m]');
ylabel('Y position [m]');
hold on
plot(yout(1,3), yout(1,4), '*r');
legend('trajectory','starting position','location','south')
hold off

%plot Torque
axes(handles.torquePlot)
plot(tout, yout(:,7));
title('Torque vs Time');
xlabel('Time (s)');
ylabel('Torque [Nm]');
hold on
plot(tout, yout(:,8),'r');
legend('Link 1', 'Link 2','location','south');
hold off

%plot Theta1
axes(handles.thetaXPlot)
```

```
plot(tout, yout(:,5),'r');
hold on
plot(tout, yout(:,11));
%title('Theta 1 vs time');
xlabel('Time (s)');
ylabel('Theta1 [rad]');
legend('out', 'ref','location','northeastoutside');
hold off

%plot Theta2
axes(handles.thetaYPlot);
plot(tout, yout(:,6),'r');
hold on
plot(tout, yout(:,12));
%title('Theta2 vs time');
xlabel('Time (s)');
ylabel('Theta2 [rad]');

hold off

%plot PosX
axes(handles.PosTimeXPlot)
plot(tout, yout(:,3),'r');
hold on
plot(tout, yout(:,13));
%title('X position vs time');
xlabel('Time (s)');
ylabel('X position [m]');
hold off

%plot PosY
axes(handles.PosTimeYPlot)
plot(tout, yout(:,4), 'r');
hold on
plot(tout, yout(:,14));
%title('Y position vs time');
xlabel('Time (s)');
ylabel('Y position [m]');
hold off

%Calculate performance measure
completionTime = 5* tf;
set(handles.tCompletion, 'String', completionTime);

diffPosX = yout(:,13)-yout(:,3);
iseX = sum(diffPosX.^2);
set(handles.ISEPosX, 'String', iseX);

diffPosY = yout(:,14)-yout(:,4);
iseY = sum(diffPosY.^2);
set(handles.ISEPosY, 'String', iseY);

istorque1 = sum(yout(:,7).^2);
set(handles.ISTorque1, 'String', istorque1);
```

```
istorque2= sum(yout(:,8).^2);
set(handles.ISTorque2, 'String', istorque2);

%move 2 deviation calc
x = yout(:,3);
move2start = 100*tf+1;
move2stop = 200*tf - 1;
move2Deviation = abs(0.1 - x(move2start:move2stop));
move4start = 300*tf + 1;
move4stop = 400 * tf - 1;
move4Deviation = abs(0.3 - x(move4start:move4stop));
maxdeviation2 = max(move2Deviation);
maxdeviation4 = max(move4Deviation);
set(handles.maxDev2, 'String', maxdeviation2);
set(handles.maxDev4, 'String', maxdeviation4);
 set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

directhanoi.m

```
function p = directhanoi(theta)
    %given angle, calculates position of end effector
    a = 0.2;
    t1 = theta(1);
    t2 = theta(2);

    p1 = a* cos(t1)+ a* cos(t1 + t2);
    p2 = a* sin(t1) + a* sin(t1 + t2);

    p = [p1;p2];
end
```

inversehanoi.m

```
function theta = inversehanoi(px,py)
% calculates joint angle vector position given the end effector position
    a = 0.2; %m
    alpha = atan(py/px);
    beta = acos((px^2 + py^2)/(2*a*sqrt(px^2+py^2)));

    theta1 = alpha + beta;

    ctheta2 = (px^2 + py^2 - a^2 -a^2)/(2*a*a);
    stheta2 = - sqrt(1 - ctheta2^2); % for elbow up

    theta2 = atan2(stheta2, ctheta2);

    theta = [theta1; theta2];
end
```

hanoitraj.m

```
function tau = hanoi_torque( u )
```

```
%u = vector of reference link angular positions, velocities, and
%accelerations. calculates required torque for open loop
theta1_r = u(1);
theta2_r = u(2);
theta1dot_r = u(3);
theta2dot_r = u(4);
theta1dotdot_r = u(5);
theta2dotdot_r = u(6);

theta_r = [theta1_r; theta2_r];
thetadot_r = [theta1dot_r; theta2dot_r];
thetadotdot_r = [theta1dotdot_r; theta2dotdot_r];

global m mL a Iz l b1 b2 g; %system parameters
global tf ts omega_n zeta ; % model parameters

%define matrices
m11 = m*(a^2 + 2*l^2 + 2*a*l*cos(theta2_r)) + 2* Iz + 2*mL*a^2*(1+cos(theta2_r));
m12 = m*l*(l+a*cos(theta2_r)) + Iz + mL*a^2*(1+cos(theta2_r));
m21 = m*l*(l+a*cos(theta2_r))+Iz+mL*a^2*(1+cos(theta2_r));
m22 = m*l^2+Iz+mL*a^2;
M_theta = [m11 m12; m21 m22];

b11 = -2*(m*a*l+mL*a^2)*sin(theta2_r)*theta2dot_r;
b12 = -(m*a*l + mL*a^2)*sin(theta2_r)*theta2dot_r;
b21 = (m*a*l + mL*a^2)*sin(theta2_r)*theta1dot_r;
b22 = 0;
b_theta = [b11 b12; b21 b22];

B = [b1 0;0 b2];

g11 = (m*(a+l) + mL*a)*g*cos(theta1_r) + (m*l + mL*a)*g*cos(theta1_r + theta2_r);
g12 = (m*l + mL*a)*g*cos(theta1_r + theta2_r);
g_theta = [g11; g12];

Tau_r = M_theta*thetadotdot_r + b_theta*thetadot_r + B*thetadot_r + g_theta;
tau = Tau_r;

end
```

Y.m

```
function outY = Y(u)
%function to calculate Y for parameter adaptive control
%   Detailed explanation goes here
global m mL mLuser a Iz1 Iz2 Iz l b1 b2 g;
global Kd Kp

theta = [u(1); u(2)];
thetadot = [u(3); u(4)];
theta_r = [u(5); u(6)];
thetadot_r = [u(7); u(8)];
thetadotdot_r = [u(9); u(10)];
```

```
e = theta_r - theta;
edot = thetadot_r - thetadot;
Kdinv = inv(Kd);
thetadot_ra = thetadot_r + Kdinv * Kp * e;
thetadotdot_ra = thetadotdot_r + Kdinv * Kp * edot;

%Define cosines
c1 = cos(theta(1));
c2 = cos(theta(2));
c12 = cos(theta(1) + theta(2));

%Define sines
s1 = sin(theta(1));
s2 = sin(theta(2));
s12 = sin(theta(1) + theta(2));

y11 = 2*thetadotdot_ra(1) + thetadotdot_ra(2);
y21 = thetadotdot_ra(1) + thetadotdot_ra(2);
y12 = (a^2 + 2*l^2 + 2 * a * l * c2) *thetadotdot_ra(1) + l*(l+ a*c2)*thetadotdot_ra(2) - a*l*s2*(2*the
y22 = l*(l+a*c2)*thetadotdot_ra(1) + l^2 * thetadotdot_ra(2) + a*l*s2* thetadot(1)* thetadot_ra(1) + g*
y13 = a^2*(1+c2)*(2*thetadotdot_ra(1) + thetadotdot_ra(2)) - a^2 * s2* (2*thetadot_ra(1) + thetadot_ra(
y23 = a^2 * (1+c2) * thetadotdot_ra(1) + a^2*thetadotdot_ra(2) + a^2 * s2 * thetadot(1) * thetadot_ra(1
y14 = thetadot_ra(1);
y24 = thetadot_ra(2);

outY = [y11 y12 y13 y14; y21 y22 y23 y24];

end
```

pihatdot.m

```
function output = pihatdot(u )
%Calculate pihatdot for parameter adaptive control
%pihatdot = K_pi*Y'*sigma
global Kp Kd;
theta = [u(1); u(2)];
thetadot = [u(3); u(4)];
theta_r = [u(5); u(6)];
thetadot_r = [u(7); u(8)];
thetadotdot_r = [u(9); u(10)];

%Define K_pi: user defined diagonal adaptive control gain matrix 4x4
k1 = 100;
k2 = 100;
k3 = 100;
k4 = 100;
K_pi = [k1 0 0 0;0 k2 0 0; 0 0 k3 0; 0 0 0 k4];
Kpi_inv = inv(K_pi);

e = theta_r - theta;
edot = thetadot_r - thetadot;
sigma = edot + inv(Kd)* Kp * e;

Ymatrix = Y(u);
```

```
output = Kpi_inv * Ymatrix’ * sigma;
end
```

Appendix C: Other Smulink Model

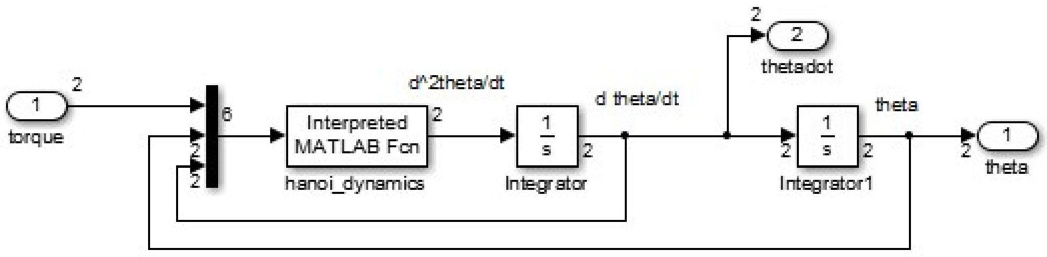


Figure 2: Simulink Model of robot Arm

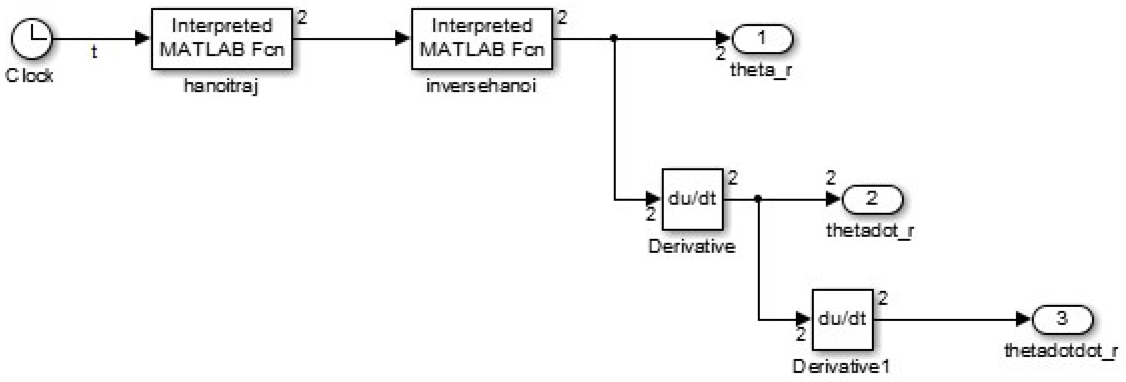


Figure 3: Simulink Model of trajectory generator