Adaptive Control - Assignment 2

Self Tuning Regulators

Student: Murtaza Asaadi, mergenelos@gmail.com
Lecturer: Prof. Baqeri, peyman.bk@gmail.com

May, 2025

Contents

List of Figures

List of Program Codes

1 Pole placement

The Matlab implementation of the original system's transfer function is given in Code 1. Its step response, shown in Figure 1, indicates that the original system is unstable.

```
s = tf('s');
G_Original = ((5*((0.7*s)+1)*(s+0.8))/((((3*s)+1)^2)*((2*s)-1)));
if drawPlot
        figure;
        step(G_Original, 'b');
        legend('Step response');
        title('Step Response of Original System');
        fontsize( 24 ,"points");
end
```

Code 1: Original system

To enable the calculation of the system's settling time, the unstable pole was mirrored and the system was discretized, as implemented in the code shown in Code 2. The step response of the resulting, modified system is presented in Figure 2.

```
%% Discrete transfer function of stable transfer function
G_discrete = c2d(G, sampleTimeIntervals, 'zoh');
% Compare step responses
if drawPlot
        figure;
        step(G, 'b');
        hold on;
        step(G_discrete, 'r');
        legend('Continuous', 'Discrete');
        title('Step Response Comparison');
        fontsize( 24 ,"points");
end
```

Code 2: Discrete stable system

Additionally, we generated the pulse train required for subsequent sections. We then evaluated various pulse periods and widths to determine the required configuration based on the resulting settling time. The code for generating these pulses is depicted in Code 3, while Fig-

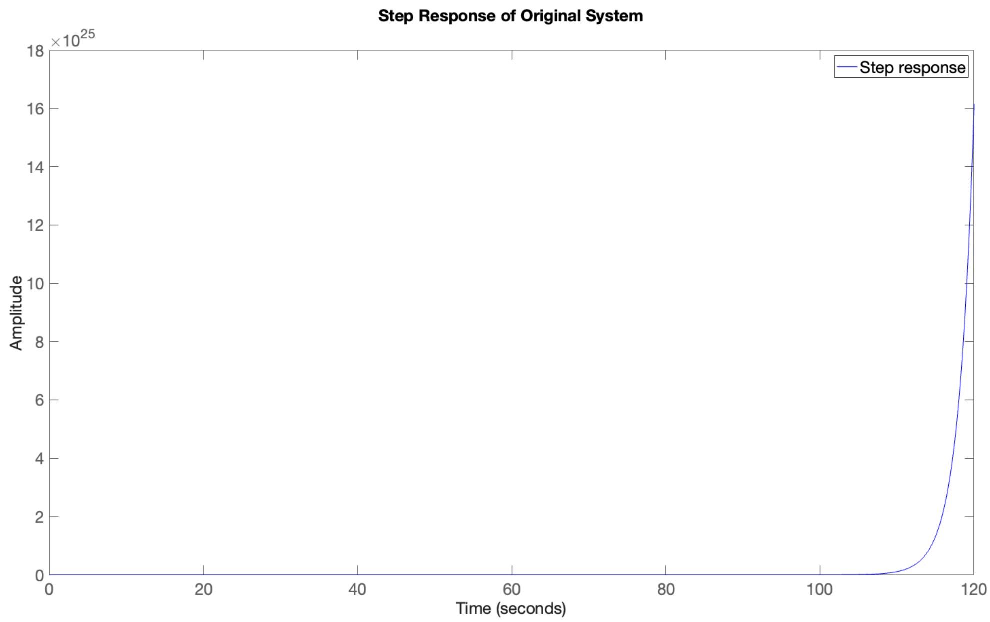

Figure 1: Original system step response

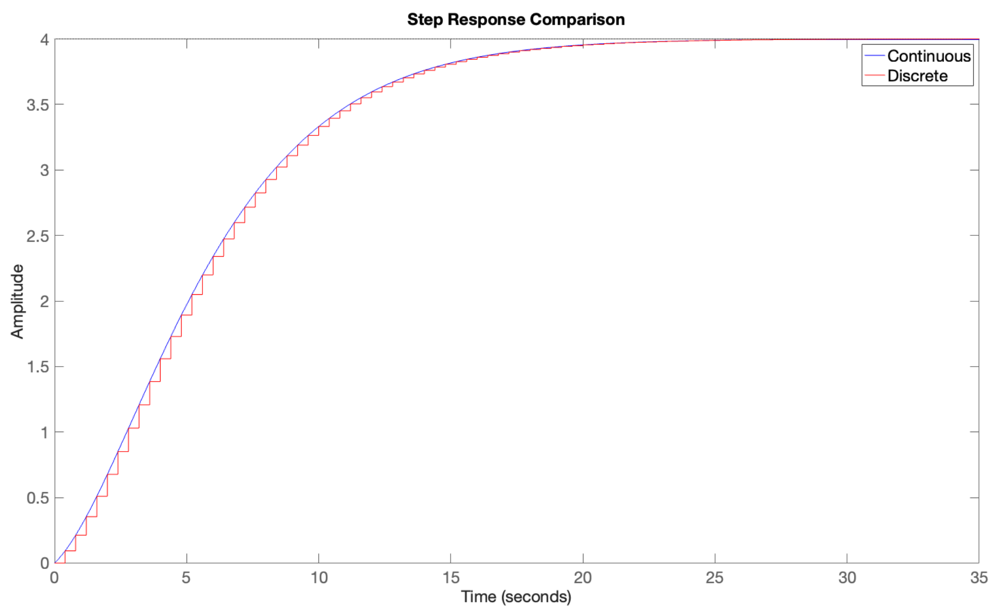

Figure 2: Discrete & contineus stable system step response

ure 3 presents the stable system's output responses to each corresponding pulse input.

```
k = 5; % 4x settling time
Tp_min = k * Ts; % Minimum recommended period between pulses

fprintf('Minimum Recommended Pulse Period (Tp): %.2f seconds\n', Tp_min);

%Simulation Time
Tsim = 20 * Tp_min;
t = 0:sampleTimeIntervals:Tsim;

%Pulse Train 1 (Respecting Settling Time)
u1 = gensig('square' ,Tp_min , Tsim , Gz.Ts);
u2 = gensig('square' ,Tp_min/k , Tsim , Gz.Ts);

y1 = lsim(G_discrete, u1, t);
y2 = lsim(G_discrete, u2, t);

if drawPlot
        figure;
        subplot(2,1,1);
        plot(t, u1, 'r-', t, y1, 'b-');fontsize( 24 ,"points");
        title(sprintf('Response with fpulse Allows Settling '));
        xlabel('Time (s)'); ylabel('Amplitude'); legend('Input Pulse', 'System Output'); grid on;
        ylim([-0.1 4.5]);

        subplot(2,1,2);
        plot(t, u2, 'r-', t, y2, 'b-');fontsize( 24 ,"points");
        title(sprintf('Response with fpulse Too Fast'));
        xlabel('Time (s)'); ylabel('Amplitude'); legend('Input Pulse', 'System Output'); grid on;
        ylim([-0.1 4.5]);
end
```

Code 3: Discrete stable system

The code for this section is available at assignment2/ part1 /PP1_0.m.

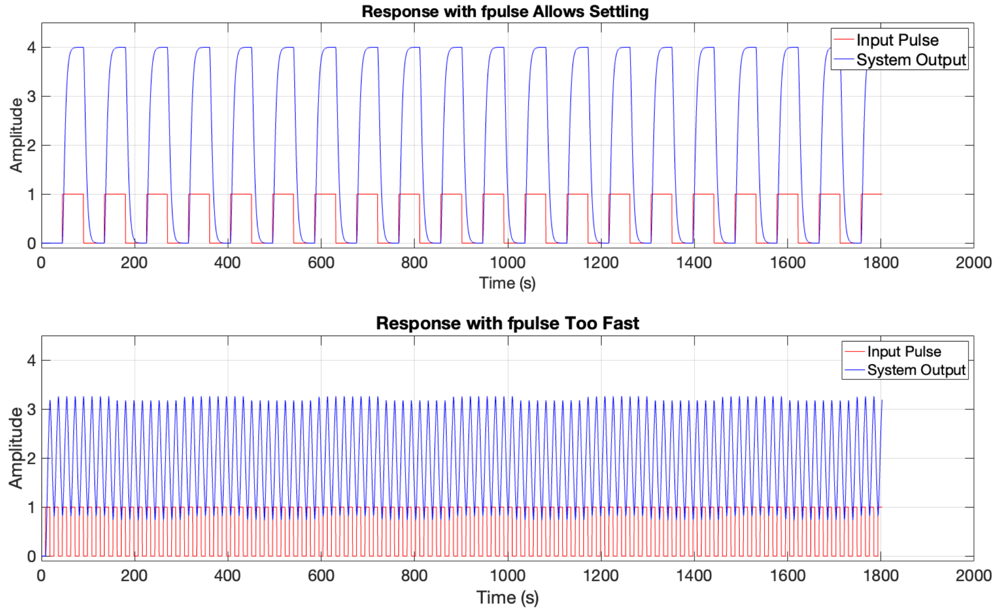

Figure 3: Pulse input and response

1.1 Question 1

Code 4 outlines the steps required to plot the zero and pole locations for both continuous and discrete systems. The resulting plots are shown in Figure 4 for the continuous system and

Figure 5 for the discrete system.

```
fprintf('Roots of continues transfer function:\n');
continuesRoots = roots(G_Original.den{1});
display(continuesRoots);
figure;
pzplot(G_Original);
ee = findobj(gca,'type','line');
for i = 1:length(ee)
        set(ee(i),'markersize',24);
        set(ee(i), 'linewidth',2) ;
end
xlim([-2 1]);title('Pole-Zero map of continues system');fontsize( 24 ,"points");

fprintf('Roots of discrete transfer function:\n');
discreteRoots = roots(Gz.den{1});
display(discreteRoots);
figure;
pzplot(Gz);
ee = findobj(gca,'type','line');
for i = 1:length(ee)
        set(ee(i),'markersize',24);
        set(ee(i), 'linewidth',2) ;
end
title('Pole-Zero map of discrete system');fontsize( 24 ,"points");
```

Code 4: Zeros & poles

As shown in Figure 5, the system has a pole located outside the unit circle, indicating that it is unstable. To design our reference model, we will relocate this unstable pole inside the unit circle and determine B_m such that the step response approaches a steady-state value of 1.

The code for this section is available at assignment2/ part1 /PP1_1.m.

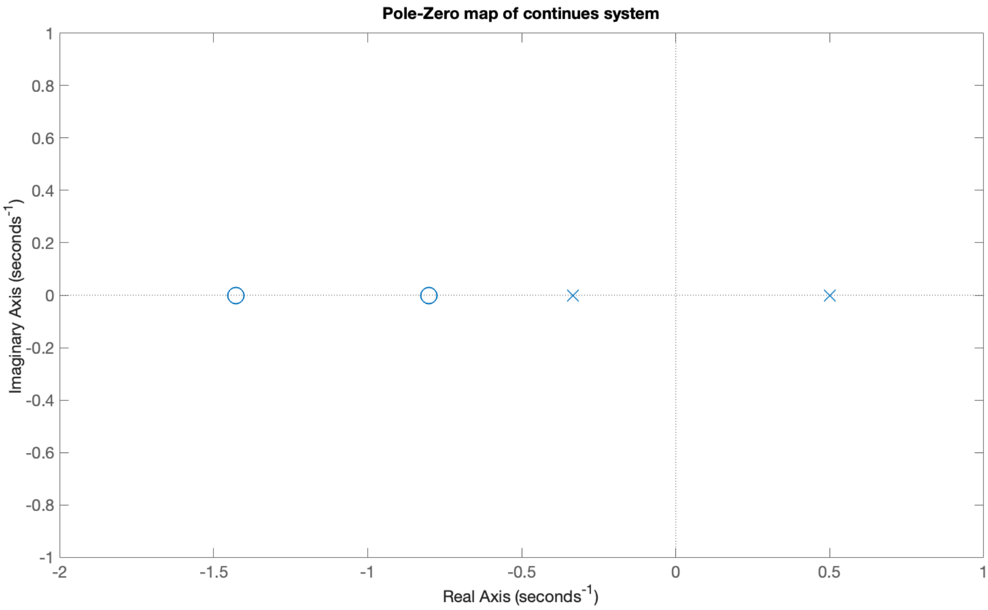

Figure 4: Continues system poles & zeros

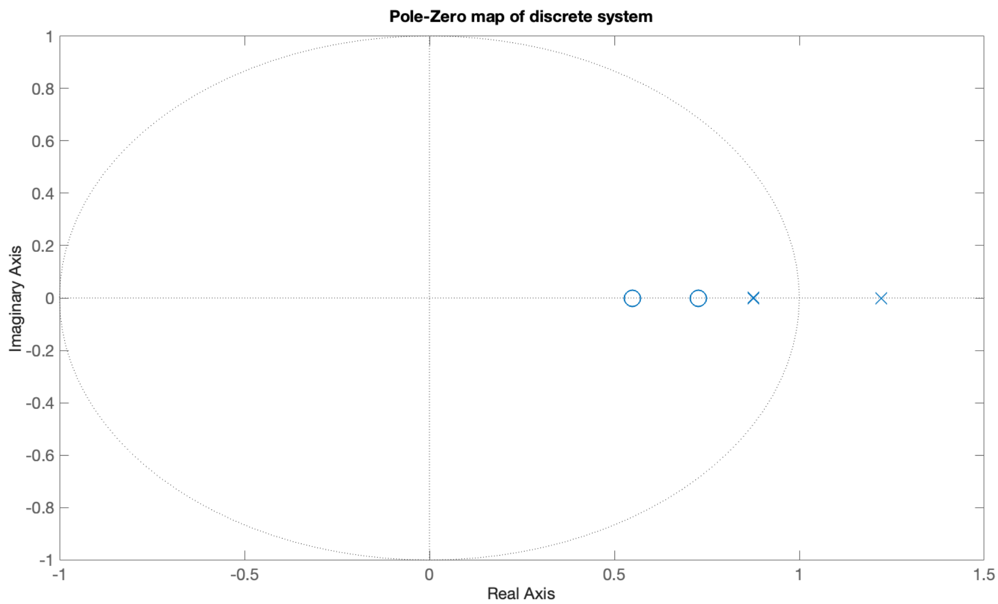

Figure 5: Discrete system poles & zeros

1.2 Question 2

In this section, we place our desired poles as shown in Code 5. Our located poles does not cancell out the system zeros. Figure 6 shows the step response of the closed loop system. Using the pulse input generated in last section, the system output and control signals are plotted at Figure 7.

```
% Convert to state-space representation
[A, B, C, D] = ssdata(Gz);

% Check controllability
Co = ctrb(A, B);
if rank(Co) == size(A, 1)
disp('System is controllable');
else
error('System is not controllable');
end

% Choose desired poles to improve settling time
desired_poles = [0.4, 0.5, 0.8];

K = place(A, B, desired_poles);

sys_cl_no_kr = ss(A - B*K, B, C, D,sampleTimeIntervals);
kr = 1 / dcgain(sys_cl_no_kr);

% Create closed-loop system with precompensator
B_cl = B * kr;
sys_cl = ss(A - B*K, B_cl, C, D,sampleTimeIntervals);
```

Code 5: Pole placement without cancellation

The code for this section is available at assignment2/part1/PP1_2.m.

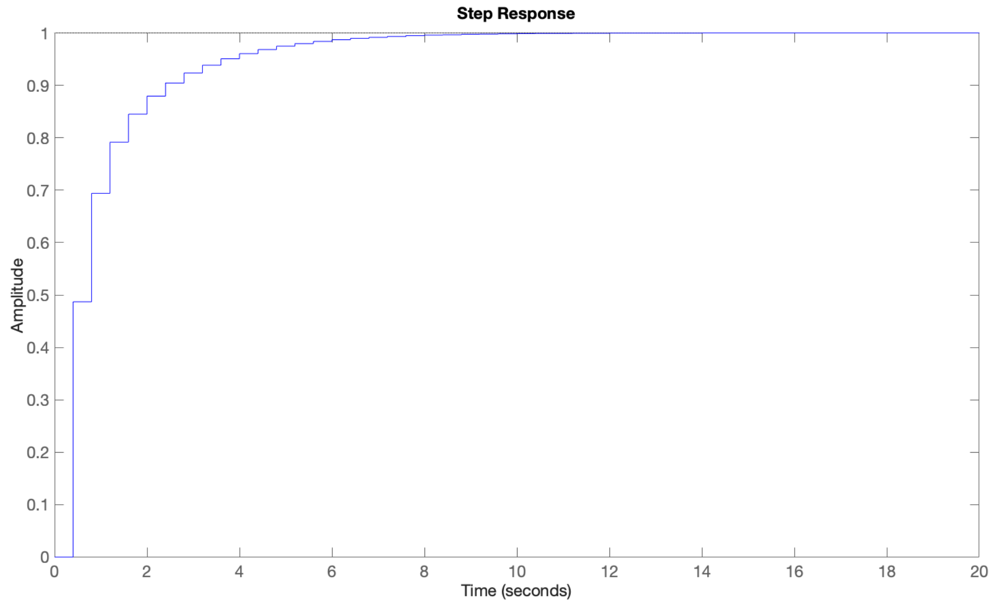

Figure 6: Step response of closed system without cancellation

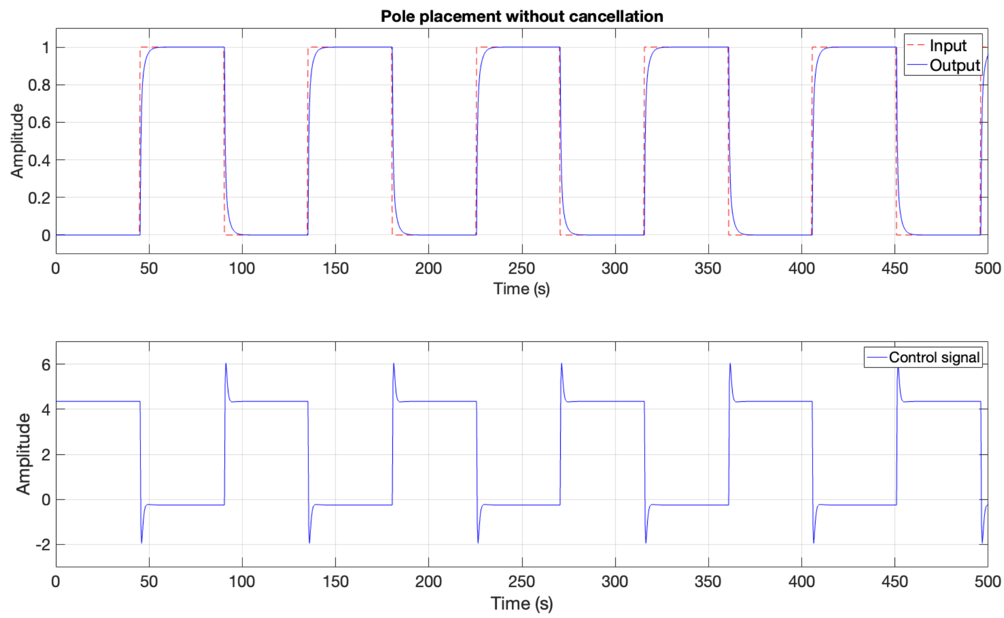

Figure 7: Pulse response of closed system without cancellation

1.3 Question 3

In this section, we place the desired poles as demonstrated in Code 6. The step response of the resulting closed-loop system is shown in Figure 8. Using the pulse input generated in the previous section, the system output and control signals are plotted in Figure 9. As shown, the desired pole locations effectively cancel the zeros of the original discrete system.

```
desired_poles = [roots(Gz.num{1}); 0.8];  % Cancel zeros with poles and add fast pole

% Compute state feedback gain
K = place(A, B, desired_poles);

% Create closed-loop system with precompensator
sys_ol = ss(A, B, C, D,sampleTimeIntervals);
sys_cl = ss(A-B*K, B, C, D,sampleTimeIntervals);
sys_cl = minreal(sys_cl);
% Adjust DC gain for unity steady-state
kr = 1/dcgain(sys_cl);
sys_cl = ss(A-B*K, B*kr, C, D,sampleTimeIntervals);
```

Code 6: Pole placement with cancellation

The code for this section is available at assignment2/part1/PP1_3.m.

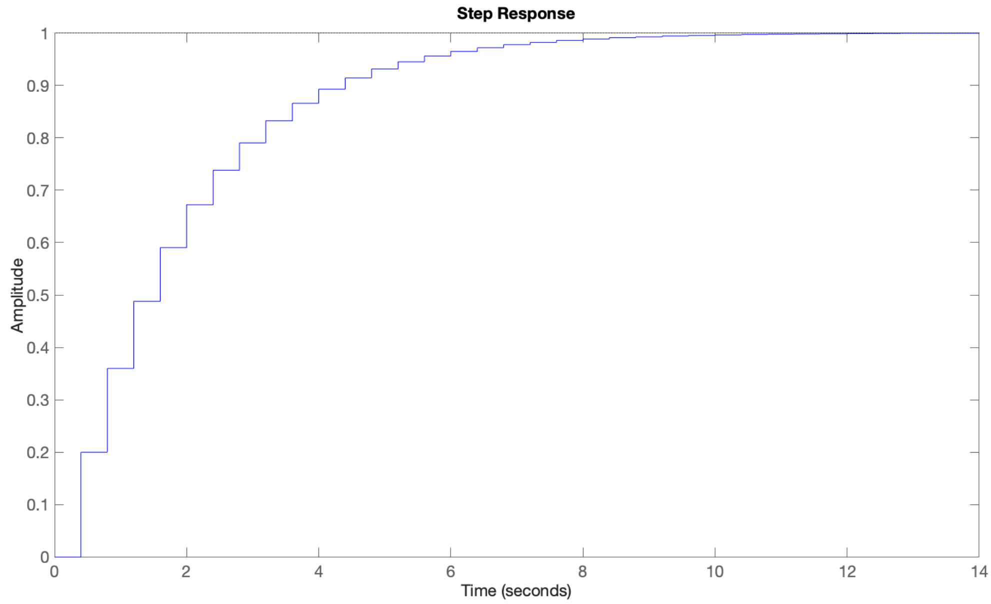

Figure 8: Step response of closed system with cancellation

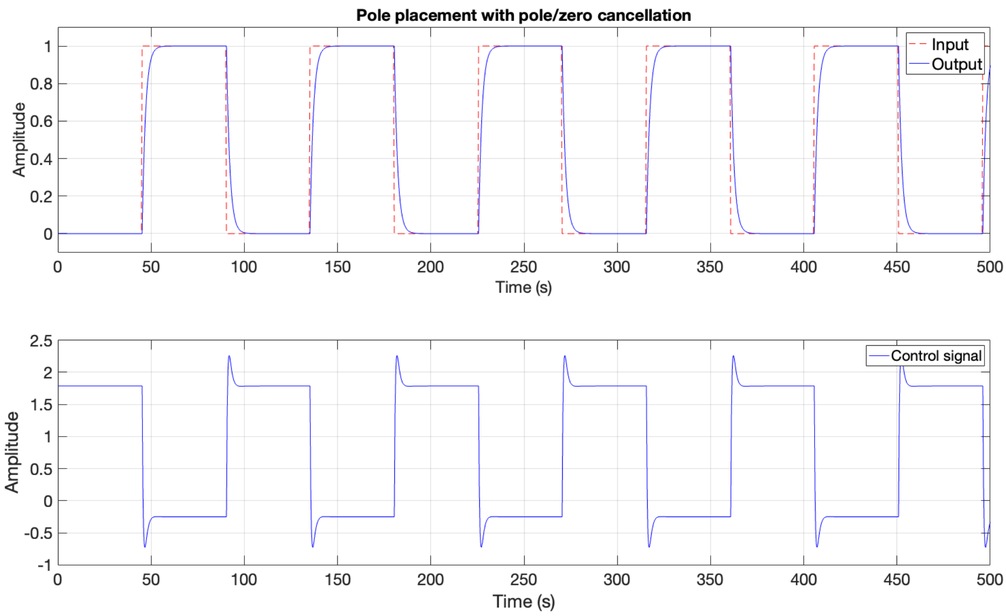

Figure 9: Pulse response of closed system with cancellation

1.4 Question 4

In this section, we mirror one of the zeros to a location outside the unit circle, as shown in Code 7. The poles and zeros of the modified system are illustrated in Figure 10. The step response of the system with the mirrored zero is presented in Figure 11, while the system's response to the pulse input is shown in Figure 12.

```
G_mirrored = ((5*((0.7*s)+1)*(s-0.8))/((((3*s)+1)^2)*((2*s)-1)));
Gz = c2d(G_mirrored, sampleTimeIntervals, 'zoh');
```

Code 7: Poles & zeros with a mirrored zero

The code for this section is available at assignment2/part1/PP1_4.m.

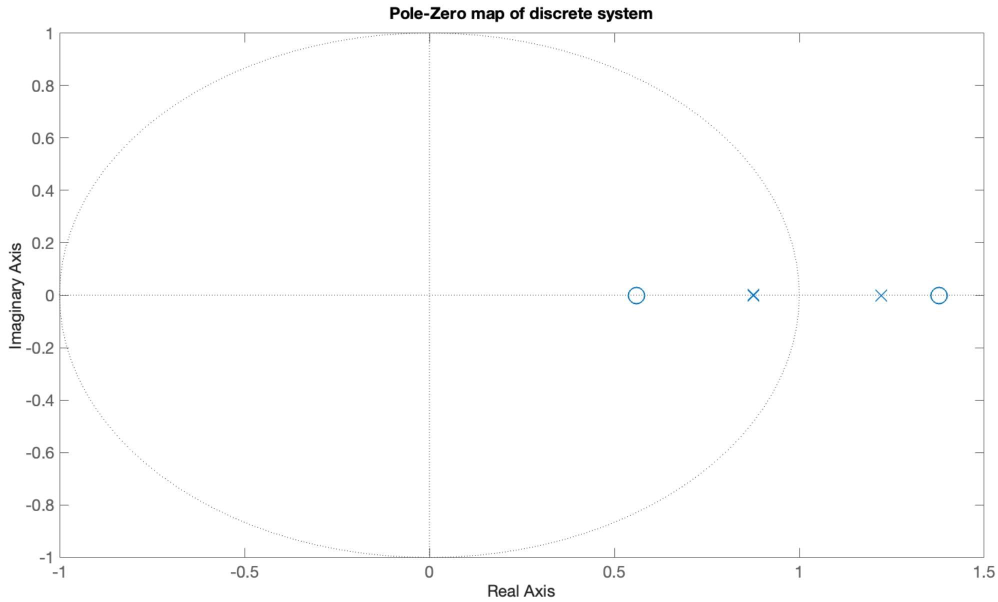

Figure 10: Step response of system with mirrored zero

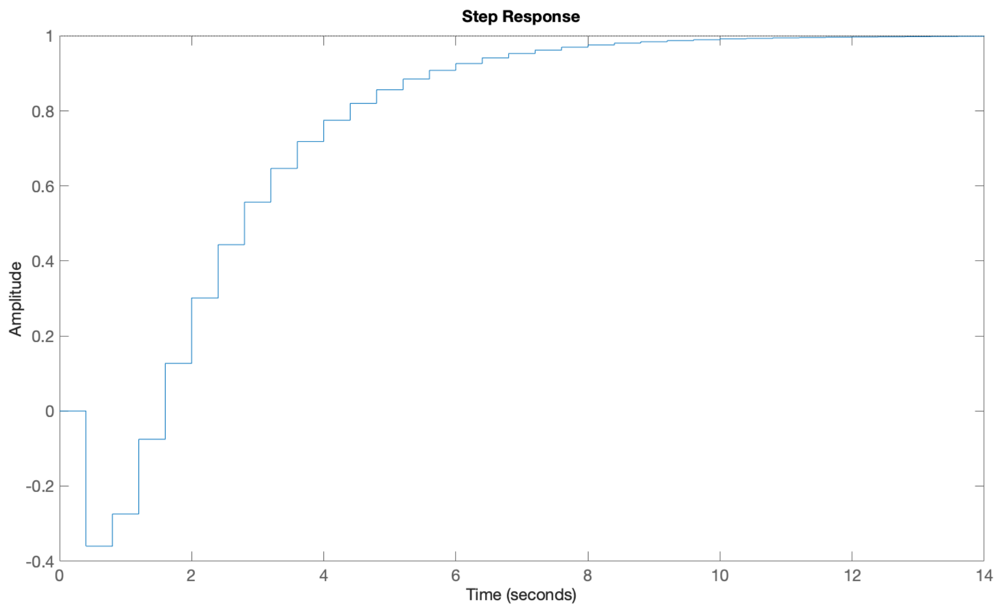

Figure 11: Pulse response of closed system without cancellation

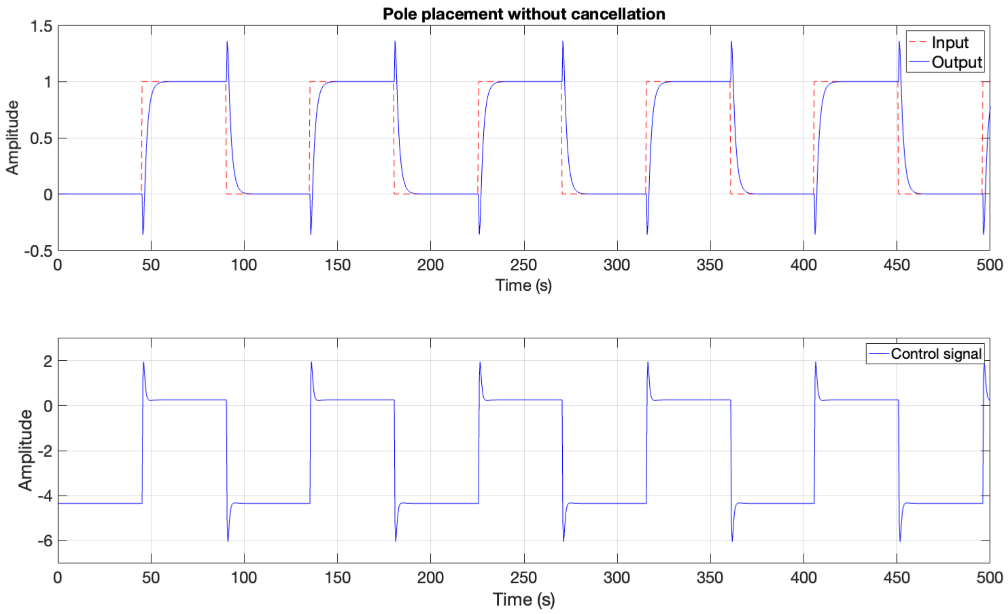

Figure 12: Pulse response of system with mirrored zero

2 STR for minimum phase system

In this section, we implement a Self-Tuning Regulator (STR)-based controller for the system. We begin with an indirect STR approach without pole-zero cancellation, followed by an indirect STR implementation with pole-zero cancellation. The same steps are then applied to the direct STR approach. Next, we analyze the effects of noise and disturbance on the indirect STR. Finally, we examine the impact of parameter variations on the performance of the direct STR. Throughout this section, we utilize Recursive Least Squares (RLS) and Diophantine equation functions, as shown in Code 8 and Code 9, respectively.

```
function [teta , P] = RLS(U , V ,Y, teta , P , Nv, lamda)
        U = U(:)' ;
        V = V(:)' ;
        phi = [U , V]' ;
        K = P*phi*(lamda+phi'*P*phi)^(-1) ;
        P = ((eye(Nv) - K*phi')*P)/lamda ;
        teta = teta + K*(Y - phi'*teta ) ;
end
```

Code 8: RLS function

```
function [R , S , Ac_result] = Diophantine(A , B , Ac)

        n = numel(A)-1 ; % degree of A polynomial
        m = numel(B)-1 ; % degree of B polynomial
        B = [zeros(1,n-m) , B] ;

        E = zeros(2*n , 2*n) ;

        for i = 1:n
        E(: , i) = [zeros(i-1 ,1) ; A' ; zeros(n-i , 1)
         ↪  ] ;
        E(: , i+n) = [zeros(i-1 ,1) ; B' ; zeros(n-i ,
         ↪  1) ] ;
        end

        nc = numel(Ac)-1 ;
        Ac = [zeros(1,2*n-(nc+1)) , Ac] ;

        RS = E\ Ac' ;

        R = RS(1:n);
        S = RS(n+1:end);

        % Ac_result = A*R+B*S ;
        AR = conv(A,R) ;
        BS = conv(B,S) ;
        Ac_result = poly2sym(AR) + poly2sym(BS) ;
        Ac_result = sym2poly(Ac_result);

        S = S(:)' ;
        R = R(:)' ;

end
```

Code 9: Diophantine function

2.1 Question 1

Our reference model is based on the approach presented in the textbook and is implemented as $sys_{ref} = tf(beta * B, Am, Gz.Ts)$, where $beta * B$ is chosen such that the step response of the reference model approaches 1. We define $A_o = [0, 0]$ and compute $Ac = poly([A0, roots(Am)'])$.

Code 10 outlines the necessary steps to compute the control output. The system polynomials are estimated using the Recursive Least Squares (RLS) method, and the R and S polynomials are determined using the Diophantine equation. Figure 13 shows the system output and control effort for the indirect STR without zero cancellation, while Figure 14 illustrates the evolution of parameter estimates over the course of the simulation.

```
%% Parameters
cancel = 0; % 0 no zero cancel, 1 all zero cancel
noise = 0; % if 1 white noise, 2 colored noise
distrubance = 1; % set to 1 for step disturbance
distrubance_fix = 0; % set to 1 to fix system
integral_fix = 0; % set 1 to limit u
vlimit = 4; % limit of u
lamda = 1;

beta = sum(Am)/sum(B);
sys_ref_dis = tf(beta*B,Am,Gz.Ts) ;
. . .
for i = Nv+1:N

        y(i) =-A(2:end)*y(i-1:-1:i-na)+B*(u(i-d0:-1:i-na)+vdist(i-(numel(A)-numel(B)):-1:i-(numel(A)-1))+ynoise(i-(numel(A)-numel(B)):-1:i-(numel(A)-1))) ;
        Y = [-y(i-1) , -y(i-2), -y(i-3)];
        U = [u(i-1)+vdist(i-1), u(i-2)+vdist(i-2), u(i-3)+vdist(i-3)] ;

        [teta , P] = RLS(Y ,U , y(i) , teta , P , Nv,lamda) ;
        tetas(:,i)=teta;

        Aes = [1 teta(1:Nv/2)'] ;
        Bes = teta(Nv/2+1:end)' ;

        [R , S] = Diophantine(Aes , Bes , Ac)  ;
        AcBm = conv(Ac , Bm) ;
        AmB = conv(Am , Bes) ;
        T = [(sum(AcBm) / sum(AmB))*poly(A0)] ;

        u(i) = (-R(2:end)*u(i-1:-1:i-(numel(R)-1))+T*uc(i-(numel(R)-numel(T)):-1:i-(numel(R)-1))-S*y(i-(numel(R)-numel(S)):-1:i-(numel(R)-1)))/R(1) ;
end
```

Code 10: Basic impelementation of Indirect STR without zero cancellation

The code for this section is available at assignment2/part2/STR1_indirect.m. The Diophantine equation solver code is at assignment2/part2/Diophantine.m and RLS impelementation is located at assignment2/part2/RLS.m.

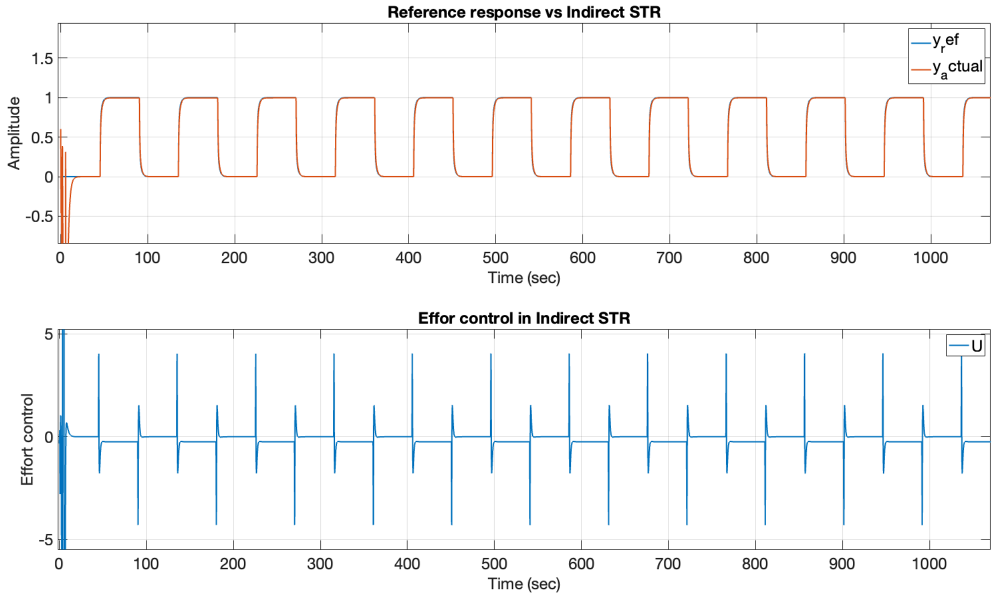

Figure 13: System response without zero cancelling Indirect STR

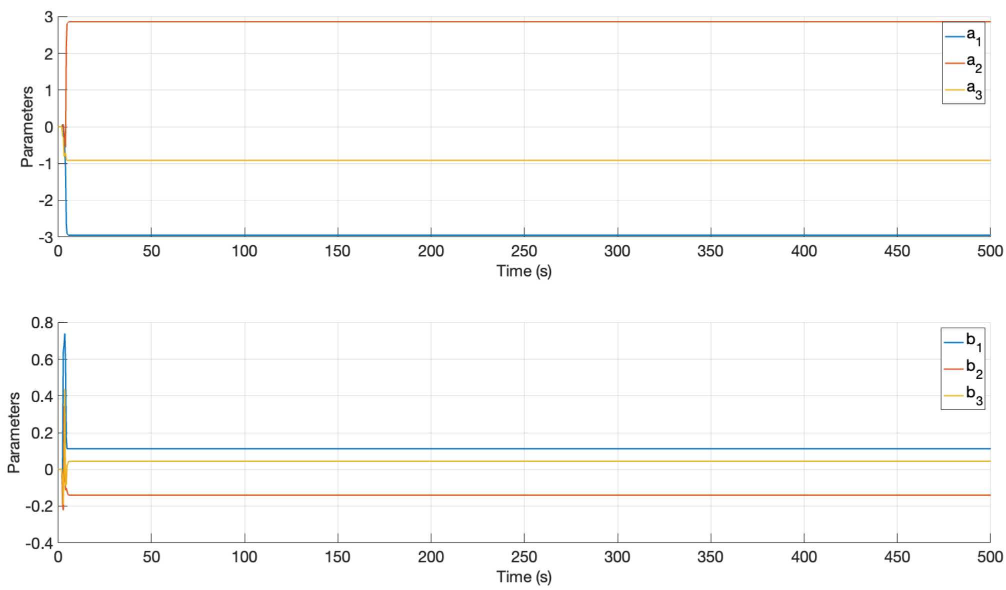

Figure 14: System parameters in indirect STR without zero cancellation

2.2 Question 2

By cancelling out the zeros of system, A_o will be of degre 0. The refrence model will be calculated based on Code 11 . The main difference in zero cancellation is the calculations required for determining S and R parameters. The system polynomials are estimated using RLS method. Figure 13 shows the system output and control effort for indirect STR with zero cancellation. Figure 14 demonstrates the variation of parameter values over simulation time.

```
%% Parameters
cancel = 1; % 0 no zero cancel, 1 all zero cancel
noise = 0; % if 1 white noise, 2 colored noise
distrubance = 1; % set to 1 for step disturbance
distrubance_fix = 0; % set to 1 to fix system
integral_fix = 0; % set 1 to limit u
vlimit = 4; % limit of u
lamda = 1;
. . .
if cancel
        sys_ref_dis = tf([0 sum(Am) 0],Am,Gz.Ts) ;
. . .
if cancel
        A0 = 0;
        Ac = poly([A0 roots(B)']) ;
. . .
if cancel
        S = zeros(1,numel(Am)-1);
        R = zeros(1,numel(Am)-1);
        R(1) = 1;
        for j = 1:numel(Am)-1
                S(j) = (Am(j+1)-Aes(j+1))/Bes(1);
        end
        for j = 2:3
                R(j) = Bes(j)/Bes(1);
        end
        T = [0,Bm(2)/Bes(1),0] ;
```

Code 11: Impelementation of Indirect STR with zero cancellation

The code for this section is available at assignment2/part2/STR1_indirect.m. By changing the $change = 0$ to 1 system will calculate the R and S and determine other parameters required for Indirect STR with zero cancellation. The Diophantine equation solver code is at assignment2/part2/Diophantine.m and RLS impelementation is located at assignment2/part2/RLS.m.

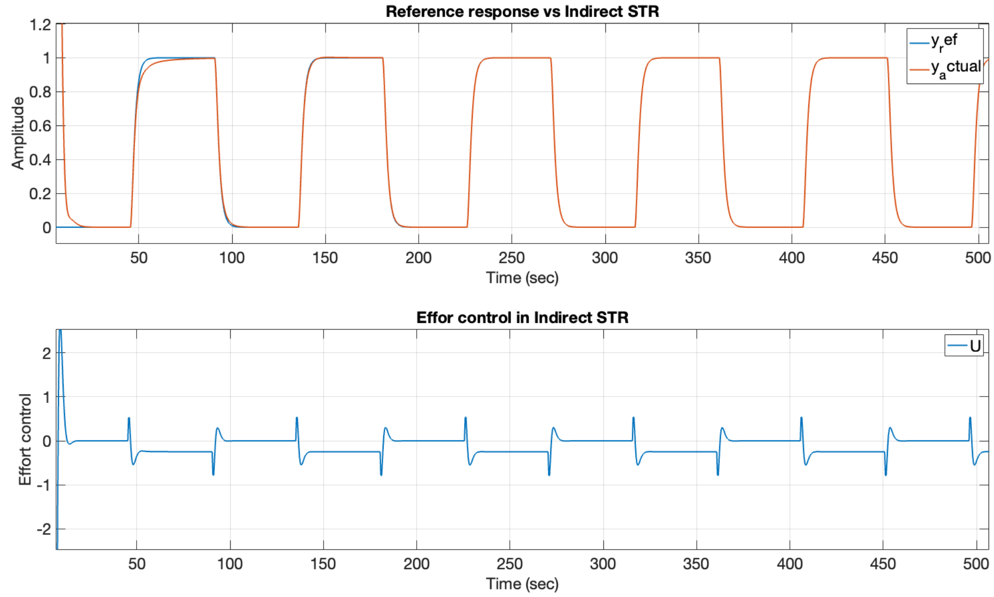

Figure 15: Indirect STR ystem response with zero cancellation

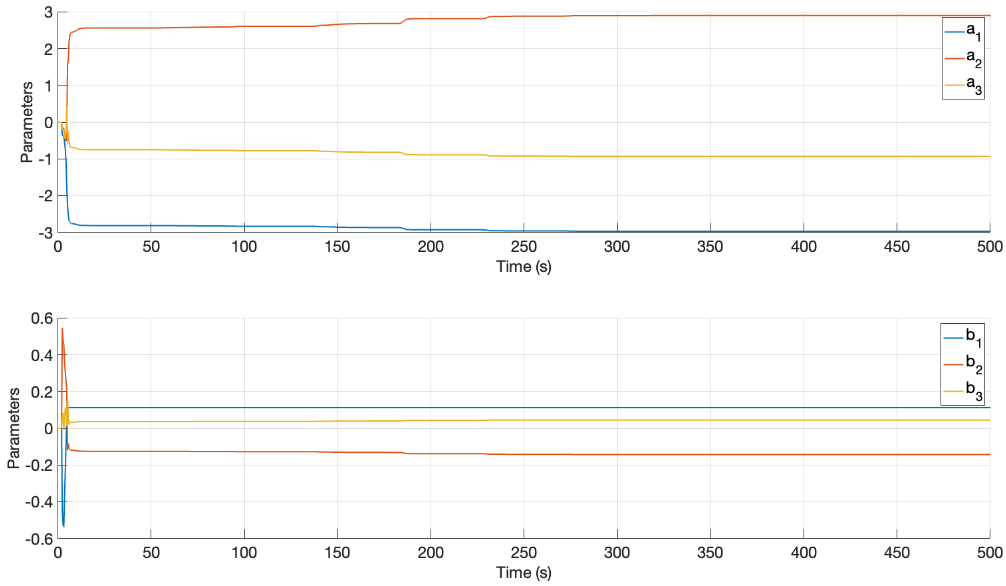

Figure 16: System parameters in indirect STR with zero cancellation

2.3 Question 3

Since we are not cancelling the zeros of the system $degA_o = degA - degB - 1 = 0$ so we implement $A_o = [1]$ as a constant. Code 12 and Code 13 show the required steps to calculate the proper control output. The values of R , S and T are directly estimated using RLS . Figure 17 shows the system output and control effort for direct STR without zero cancellation. Figure 18 demonstrates the variation of parameter values over simulation time.

```
cancel = 0; % 0 no zero cancel, 1 all zero cancel
%% generate Data
uc = u1 ;
lamda = 1;
system_dig = Gz; %system
[B  ,A] = tfdata(system_dig) ;
A = cell2mat(A) ;
B = cell2mat(B) ; B = B(2:end) ;
% reference model
Am = [1 -1.7 0.92 -0.16];
if cancel
        sys_ref_dig = tf([0 sum(Am) 0],Am,Gz.Ts) ;
        A0 = [roots(B)'];
else
        beta = sum(Am)/sum(B);
        sys_ref_dig = tf(beta*B,Am,Gz.Ts) ;
        A0 = [1];
end
[Bm  ,Am] = tfdata(sys_ref_dig);
Am = cell2mat(Am) ;
Bm = cell2mat(Bm) ; Bm = Bm(2:end) ;
y_ref = lsim(sys_ref_dig , uc , t) ;
%% initial parameters
n = numel(A)-1 ;
m = numel(B)-1 ;
d0 = n-m ;
A0Am = conv(A0 , Am) ;
Na0am = numel(A0Am)-1 ;
L = Na0am-d0 ;
Nv = 3*(L+1) ;
teta = zeros(Nv , 1) ;
P = 1e4*eye(Nv) ;
u  = randn(Nv , 1) ;  % initial effort control
y  = randn(Nv , 1) ;  % initial output
uf = randn(Nv , 1) ;  % initial filtered effort control
yf = randn(Nv , 1) ;  % initial filtered output
ucf= randn(Nv , 1) ;  % initial filtered command signal
N = numel(t) ;
for i =1:Nv
        tetas(:,i) = teta;
end
```

Code 12: Parameters of direct STR without zero cancellation

```matlab
%% main loop
for i = Nv+1:N
        y(i) = -A(2:end)*y(i-1:-1:i-n)+B*(u(i-d0:-1:i-n)) ;
        U = uf(i-d0:-1:i-L-d0) ;
        V = [yf(i-d0:-1:i-L-d0)' , -ucf(i-d0:-1:i-L-d0)']' ;
        Y = y(i)-y_ref(i) ;

        [teta , P] = RLS(U , V , Y , teta , P , Nv,lamda) ;
        tetas(:,i)=teta;

        Rst = teta(1:Nv/3)' ;
        Sst = teta(Nv/3+1:2*Nv/3)' ;
        Tst = teta(2*Nv/3+1:Nv)' ;

        u(i) = (-Rst(2:end)*u(i-1:-1:i-L)+Tst*uc(i:-1:i-L)-Sst*y(i:-1:i-L))/Rst(1) ;
        uf(i) = -A0Am(2:end)*uf(i-1:-1:i-Na0am)+u(i) ;
        yf(i) = -A0Am(2:end)*yf(i-1:-1:i-Na0am)+y(i) ;
        ucf(i) = -A0Am(2:end)*ucf(i-1:-1:i-Na0am)+uc(i) ;
end
```

Code 13: Basic impelementation of direct STR

The code for this section is available at assignment2/part2/STR1_direct.m. RLS impelementation is located at assignment2/part2/RLS.m.

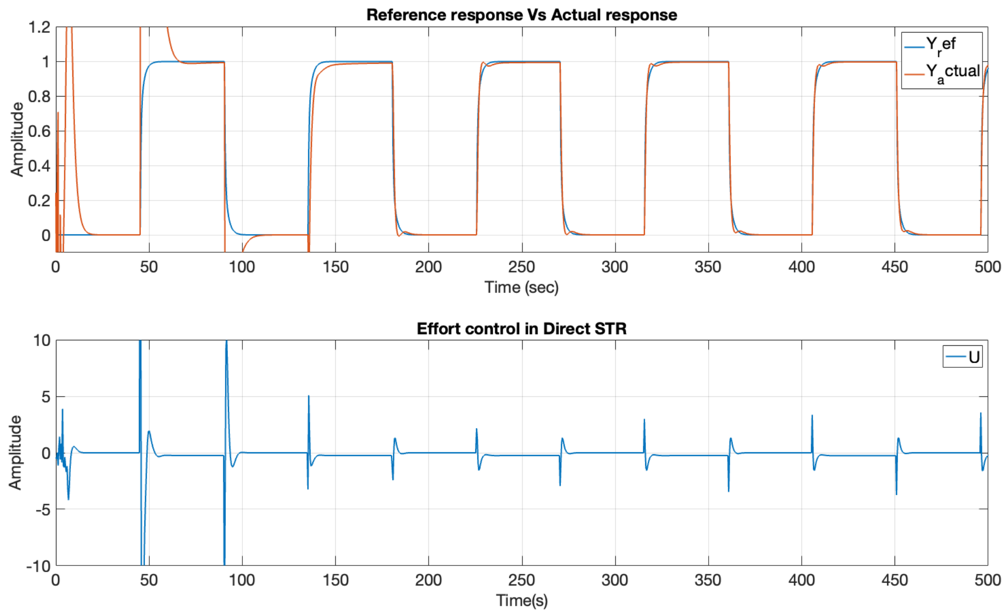

Figure 17: System response without zero cancelling direct STR

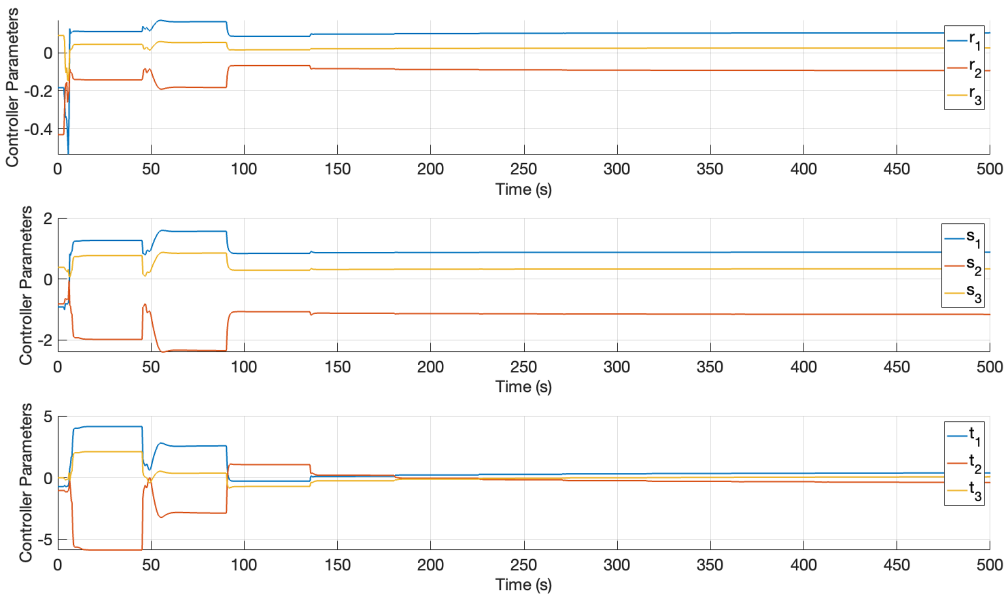

Figure 18: System parameters in direct STR without zero cancellation

2.4 Question 4

By changing the $cancel$ variable to 1 in Code 12, we can get the system response when all zeros are cancelled out. Figure 13 shows the system output and control effort for direct STR with zero cancellation. Figure 18 demonstrates the variation of parameter values over simulation time.

The code for this section is available at assignment2/part2/STR1_direct.m. By changing the $change = 0$ to 1 system will determine values of R and S and T that are required for direct STR with zero cancellation.

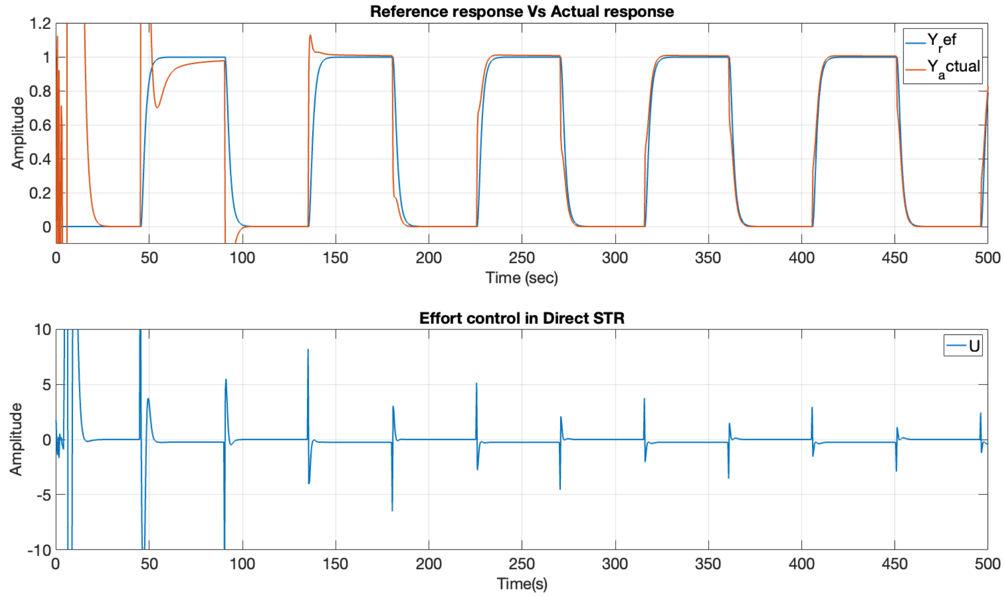

Figure 19: direct STR ystem response with zero cancellation

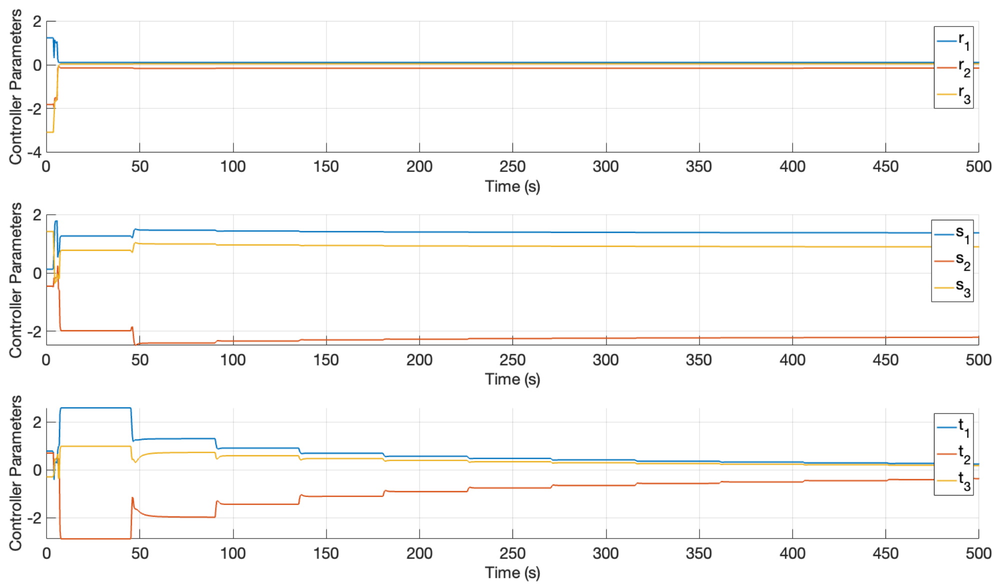

Figure 20: System parameters in direct STR with zero cancellation

2.5 Question 5

Figure 21 presents the system output when R, S and T are over-parameterized. Figure 22 shows the respective parameters over time.

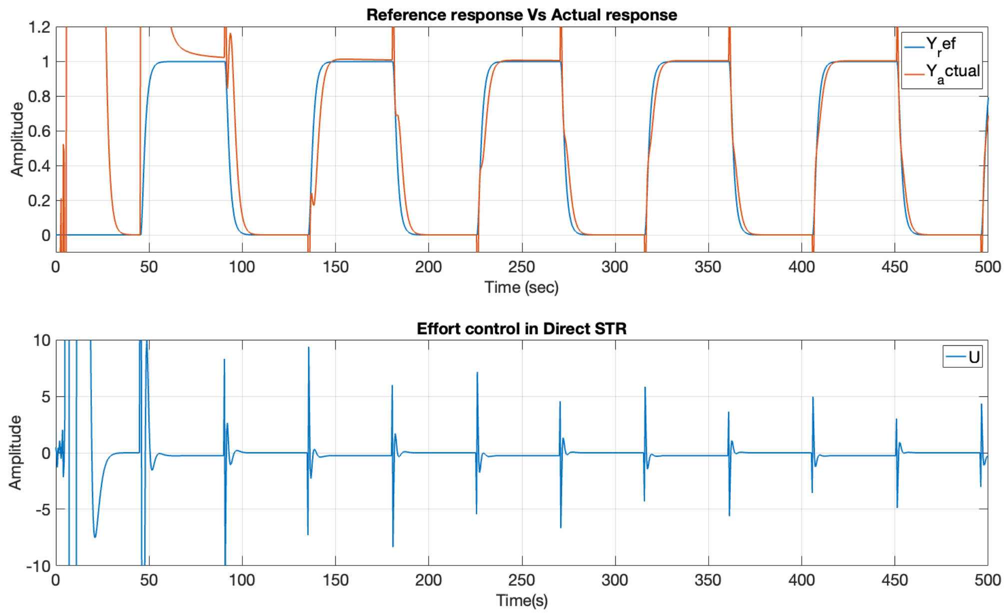

Figure 21: Over-parameterized system response

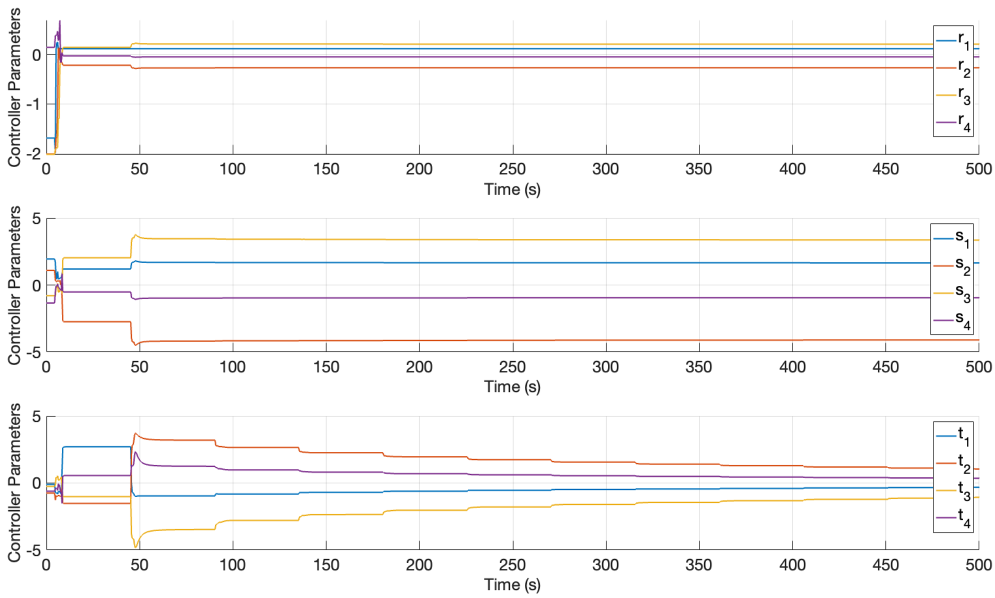

Figure 22: Parameter changes of over-parameterized system

2.6 Question 6

In this section, we vary the P and $teta$ matrix using Code 14. In Question 3 of this section, we reported the results when $teta = 1 * randn(Nv, 1);$ and $P = 1e6 * eye(Nv);$. Figure 23 and Figure 24 compares the system output results when P is chosen to have components smaller and bigger than $1e6$ accordingly. Figure 25 demonstrates the system outputs when initial $teta$ values are set to zero and $P = 1e6 * eye(Nv)$. Figure 26 is the result of setting closer values to actual calculated parameters for initial $teta$.

```
%teta = zeros(Nv , 1) ;
%teta(:,1) = [0.104;-0.0431;0.008;0.442;-0.167;-0.194;-0.050;0.086;0.0272];
teta = 1*randn(Nv,1) ;

P = 1e6*eye(Nv) ;
%P = 1e2*eye(Nv) ;
%P = 1e12*eye(Nv) ;
```

Code 14: Determining P and teta initial values

The code for this section is available at assignment2/part2/STR1_direct.m.

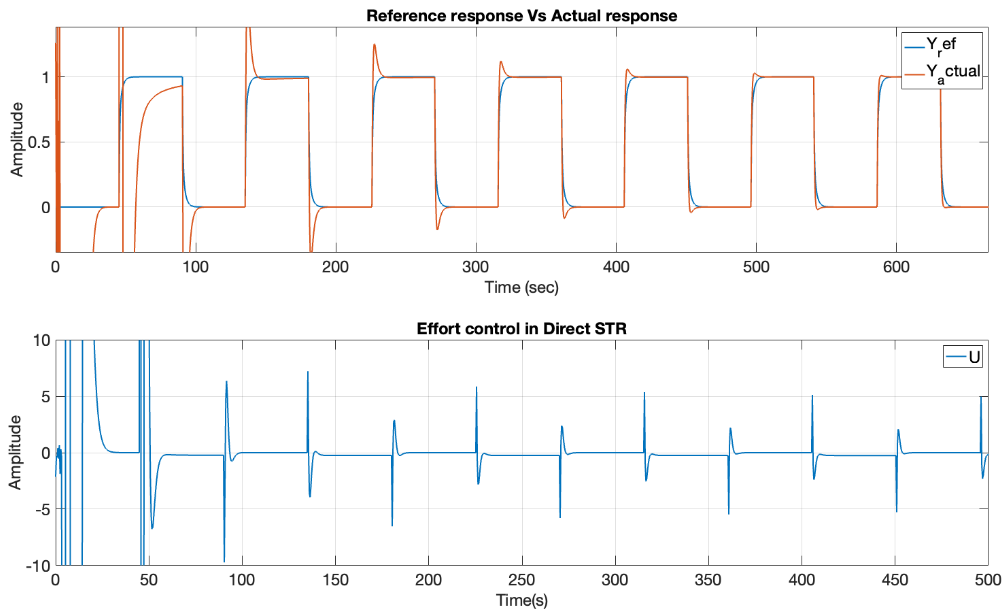

Figure 23: Smaller P values with random teta

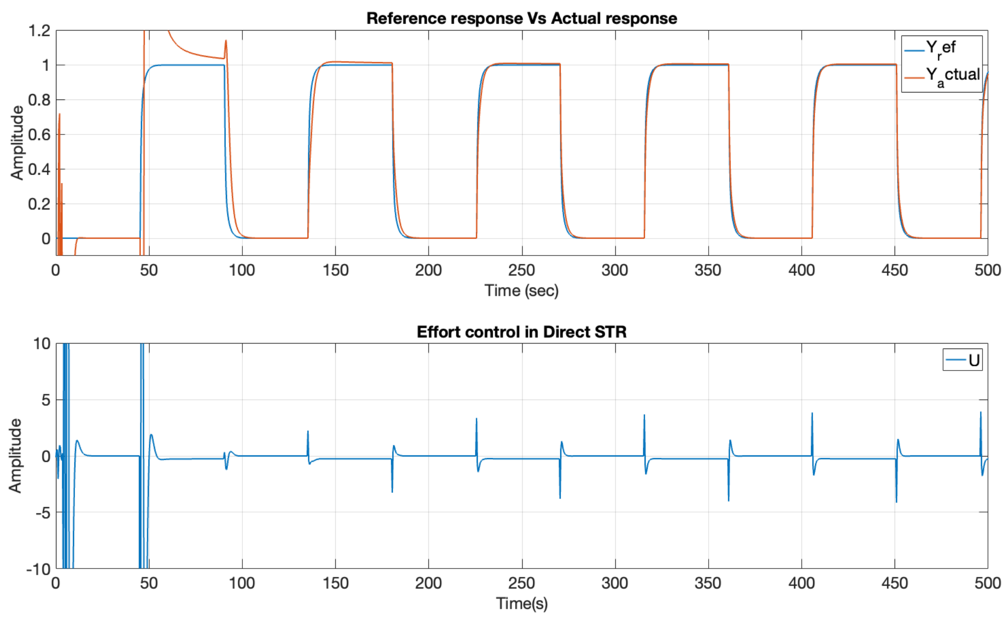

Figure 24: Bigger P values with random teta

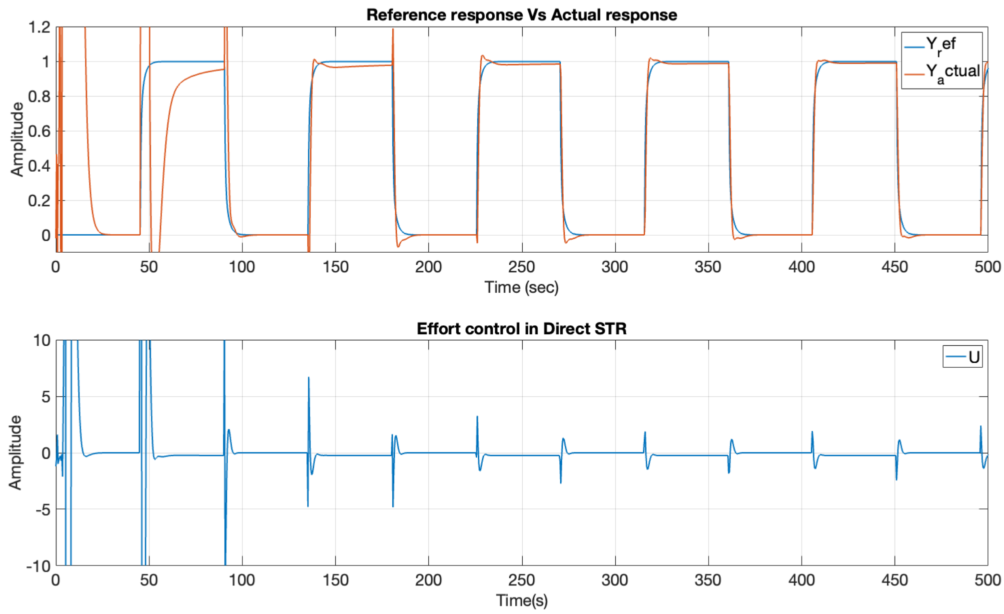

Figure 25: P = 1e6*eye(Nv) and zero initial values for teta

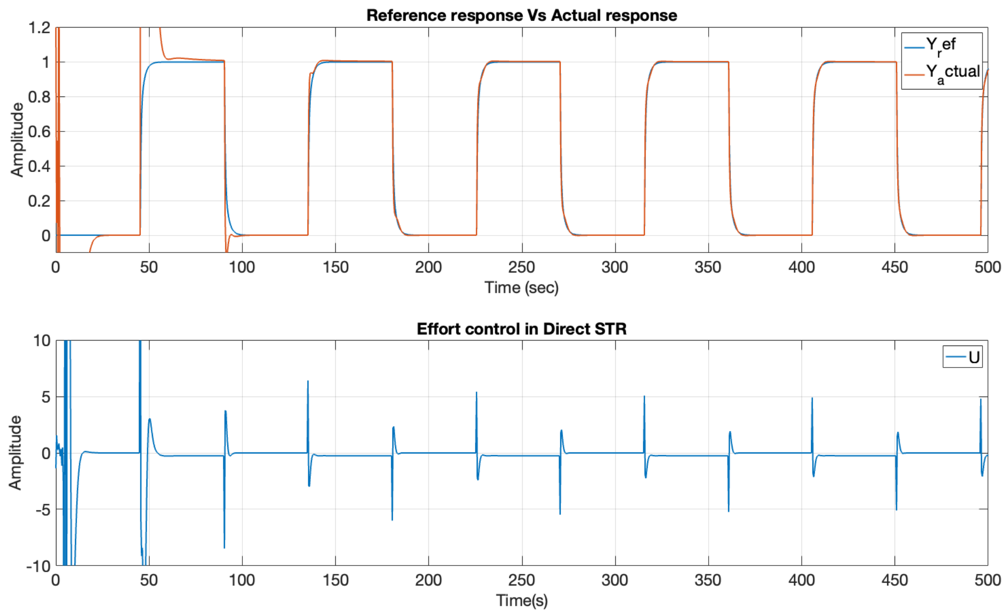

Figure 26: P = 1e6*eye(Nv) and more accurate initial values for teta

2.7 Question 7

Code 15 is the necessary changes in the codebase to implement white and colored noise and related fixes in Indirect STR controller. Effect of white noise on the system is shown in Figure 27. Figure 28 represents the variation on estimation parameters when white noise is present. By varying the $lamda$ of RLS, we can compensate for white noise as shown in Figure 29. Figure 30 and Figure 31 present the output and parameter changes of the system when colored noise is applied to the output. Figure 32 presents the fixed output of the system with colored noise. By varying the R parameter and $Lamda$ and introducing further A_o polynomial roots we can compensate for colored noise.

```
noise = 1; % if 1 white noise, 2 colored noise
lamda = 0.9;
. . .
if noise
        e = 0.05*randn(length(t),1);
        if noise == 2
                sys_dist = tf(1,[1 -1] , Gz.Ts) ;
                ynoise = lsim(sys_dist , e , t) ;
        else
                ynoise = e;
        end
else
        ynoise = zeros(length(t),1);
end
. . .
    y(i) = -A(2:end)*y(i-1:-1:i-na)+B*(u(i-d0:-1:i-na)+vdist(i-(numel(A)-numel(B)):-1:i-(numel(A)-1))+ynoise(i-(numel(A)-numel(B)):-1:i-(numel(A)-1))) ;

```

Code 15: Noise implementation in Indirect STR

The code for this section is available at assignment2/part2/STR1_indirect.m. By changing the $noise = 0$ to 1 we can introduce white noise to the system. changing the same variable to 2 implements a colored noise in the system. $lamda$ is used as a forgetting factor for RLS.

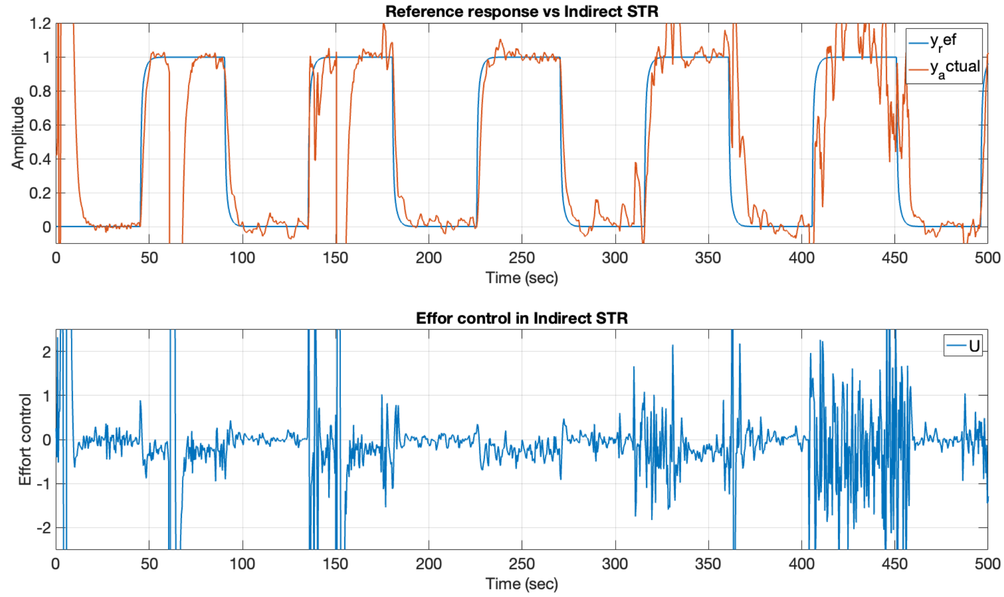

Figure 27: System output with white noise

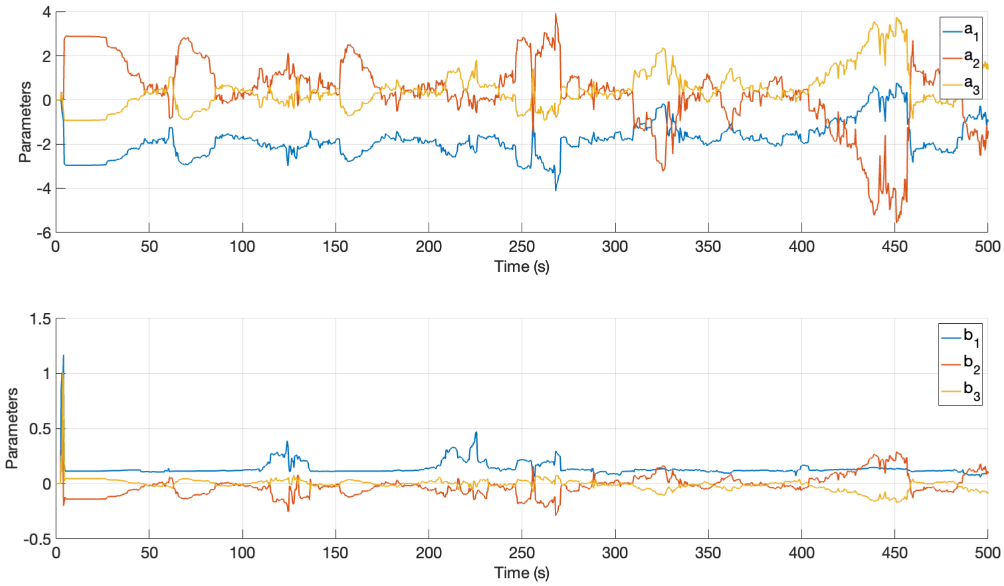

Figure 28: System parameters with white noise

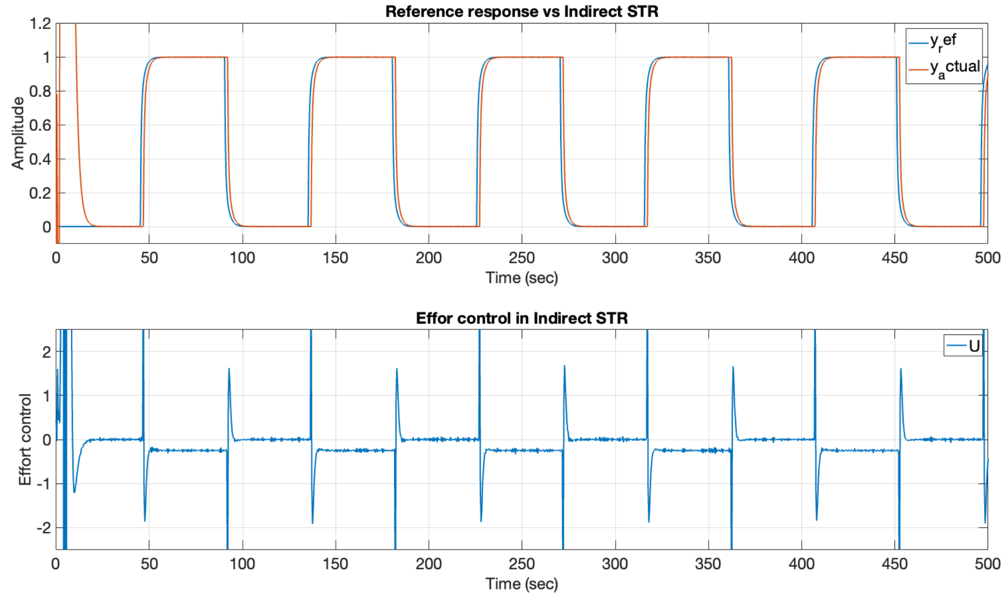

Figure 29: Compensating for system with white nois

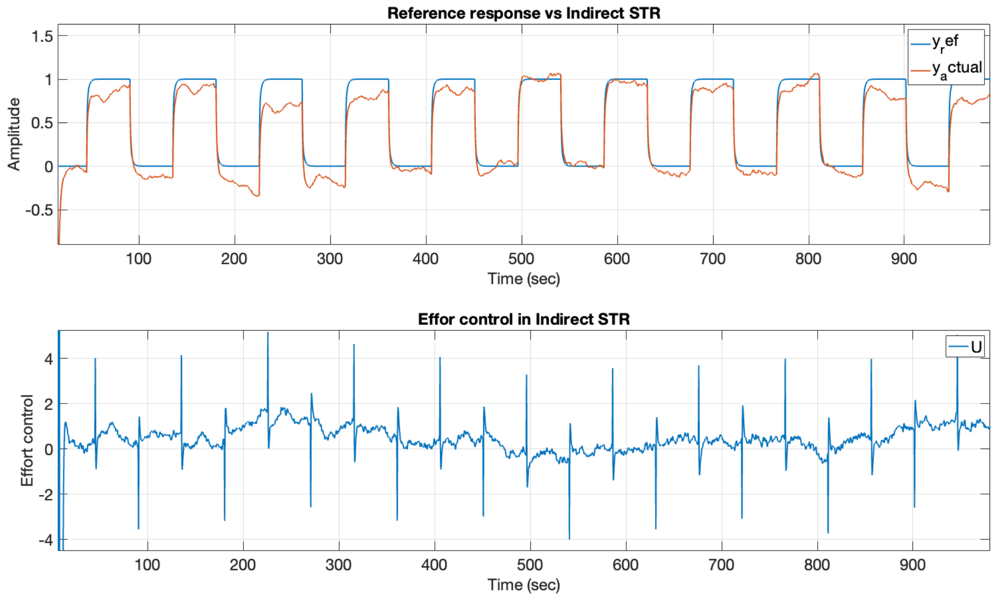

Figure 30: System output with colored noise

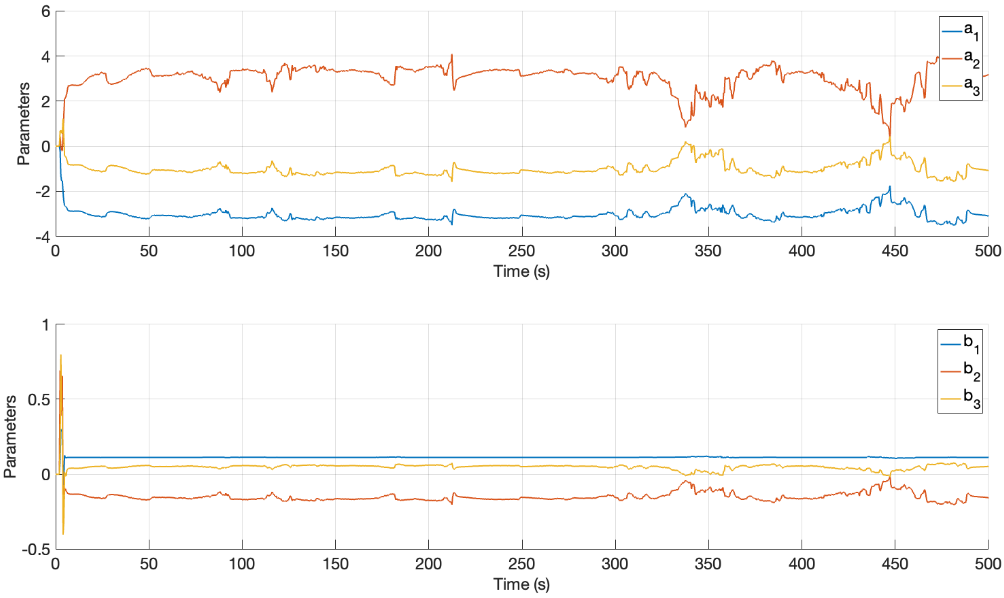

Figure 31: System parameters with colored noise

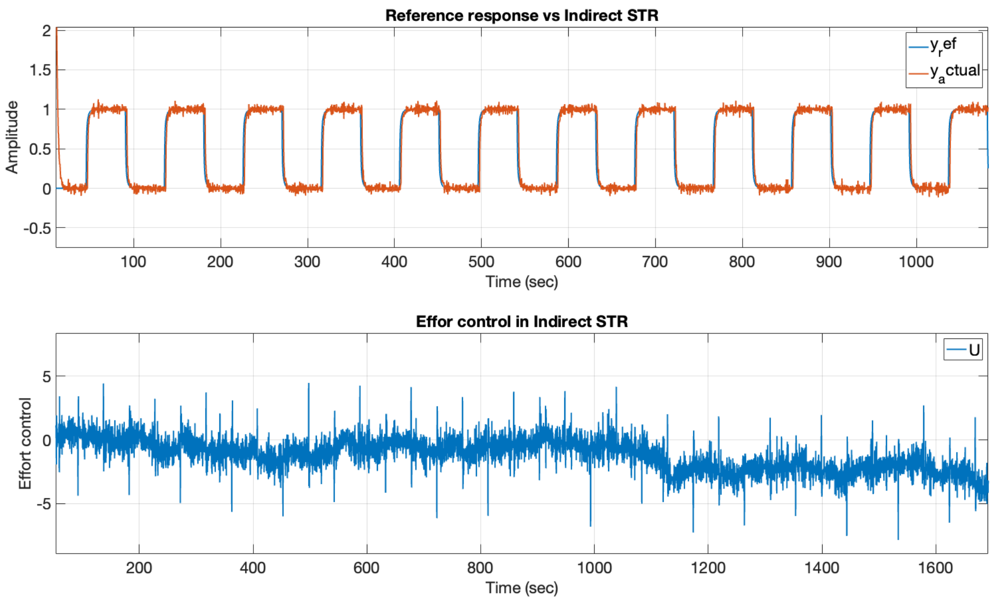

Figure 32: Compensating for system with colored noise

2.8 Question 8

Code 16 and Code 17 present the necessary modifications to the codebase for introducing a disturbance and implementing corresponding fixes in the indirect STR controller. A steady disturbance, $vdist$, is generated by filtering the signal e through a first-order filter. This disturbance affects both $y(i)$ and U. The impact of the disturbance on the system output is illustrated in Figure 33. By incorporating the characteristics of the step disturbance into the R polynomial, the system response can be corrected, as demonstrated in Figure 34. However, with sufficiently large disturbances, the implemented R may experience windup, severely degrading system performance, as shown in Figure 35. A corrected version of the system, accounting for this issue, is shown in Figure 36. The evolution of parameter estimates over time is depicted in Figure 37.

```
%% Parameters
distrubance = 1; % set to 1 for step disturbance
distrubance_fix = 1; % set to 1 to fix system
integral_fix = 1; % set 1 to limit u
vlimit = 4; % limit of u
                . . .
if distrubance
        e = 0.1*randn(length(t),1);
        for i =50:length(e)
                e(i)=0;
        end
        sys_dist = tf(1,[1 -1] , Gz.Ts) ;
        vdist = lsim(sys_dist , e , t) ;
else
        vdist = zeros(length(t),1);
end
. . .
if distrubance_fix
        A0 = [0 0 0 0] ;
else
        A0 = [0 0] ;
        %A0 = q^2 -0,5 q +0.06
end
. . .
```

Code 16: Disturbance effect on Indirect STR implementation

```
%main loop
for i = Nv+1:N
        y(i) = -A(2:end)*y(i-1:-1:i-na)+B*(u(i-d0:-1:i-na)+vdist(i-(numel(A)-numel(B)):-1:i-(numel(A)-1))+ynoise(i-(numel(A)-numel(B)):-1:i-(numel(A)-1))) ;
        Y = [-y(i-1) , -y(i-2), -y(i-3)];
        U = [u(i-1)+vdist(i-1), u(i-2)+vdist(i-2), u(i-3)+vdist(i-3)] ;

        [teta , P] = RLS(Y ,U , y(i) , teta , P , Nv,lamda) ;
        tetas(:,i)=teta;

        Aes = [1 teta(1:Nv/2)'] ;
        Bes = teta(Nv/2+1:end)' ;

        if distrubance_fix
                [Rbar , S] = Diophantine(conv(Aes,[1 -1]) , Bes , Ac)  ;
                R = conv(Rbar , [1 -1]);
                AcBm = conv(Ac , Bm) ;
                AmB = conv(Am , Bes) ;
                T = [0,0,sum(AcBm) / sum(AmB)] ;
        else
                [R , S] = Diophantine(Aes , Bes , Ac)  ;
                AcBm = conv(Ac , Bm) ;
                AmB = conv(Am , Bes) ;
                T = [(sum(AcBm) / sum(AmB))*poly(A0)] ;
                end
        end

        u(i) = (-R(2:end)*u(i-1:-1:i-(numel(R)-1))+T*uc(i-(numel(R)-numel(T)):-1:i-(numel(R)-1))-S*y(i-(numel(R)-numel(S)):-1:i-(numel(R)-1)))/R(1) ;
        if integral_fix
                if u(i)<-vlimit
                        u(i)=-vlimit;
                elseif u(i) > vlimit
                        u(i) = vlimit;
                end
        end
end
```

Code 17: Impelementation of Indirect STR with disturbance and integral windup fix

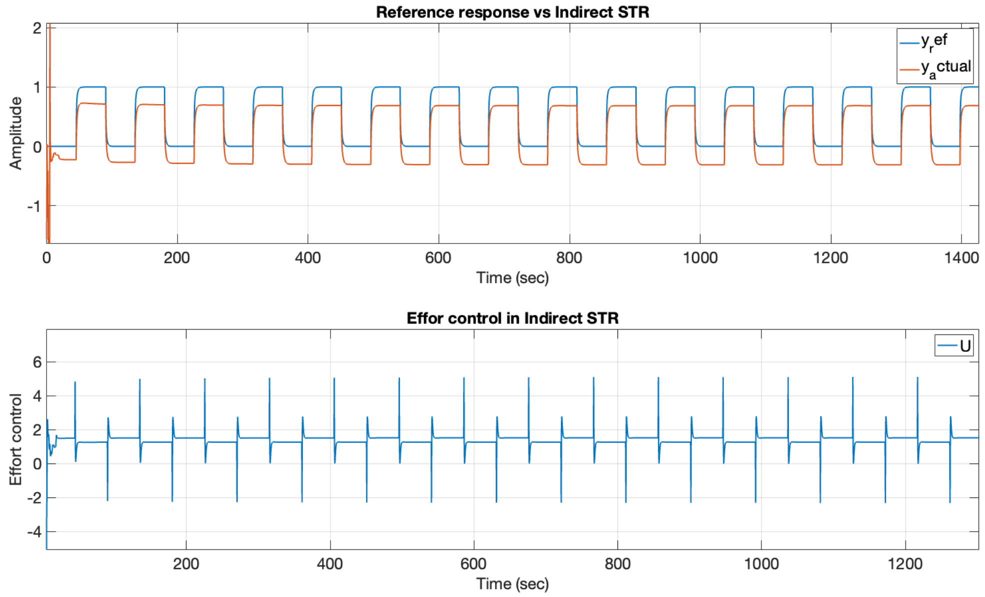

Figure 33: Output with disturbance

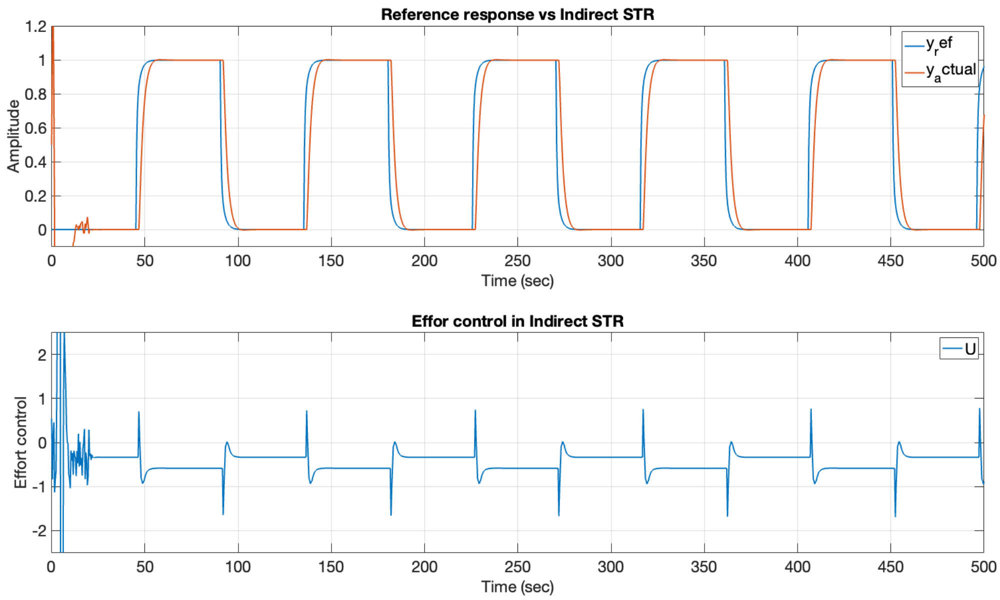

Figure 34: Fixing the disturbance

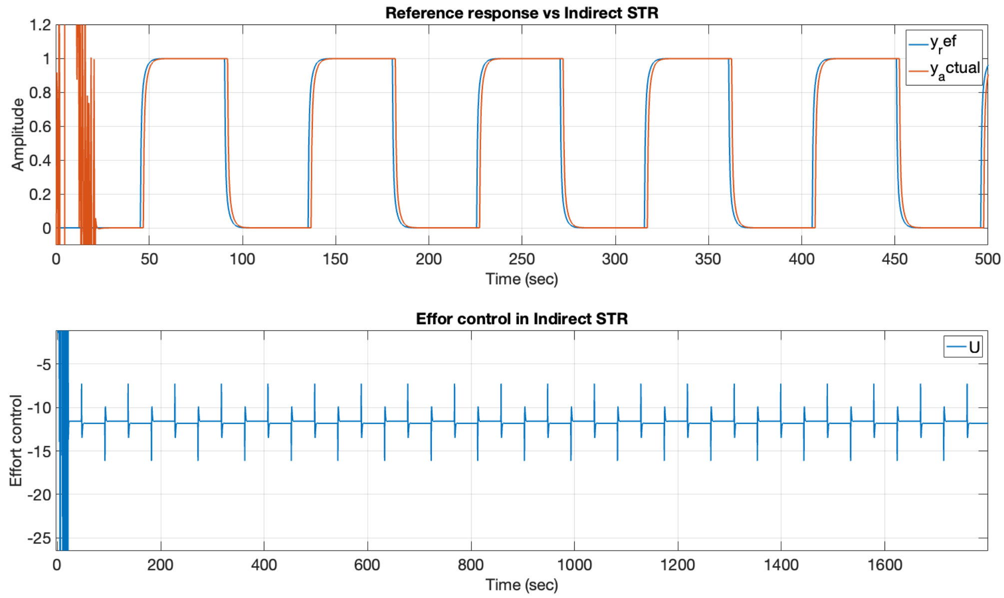

Figure 35: Integral windup

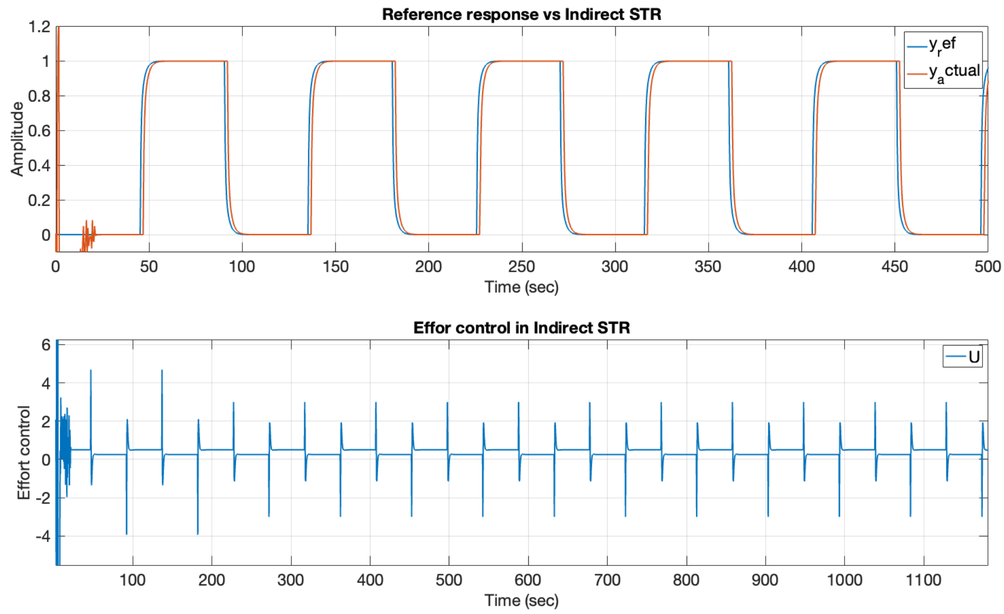

Figure 36: Integral windup fix

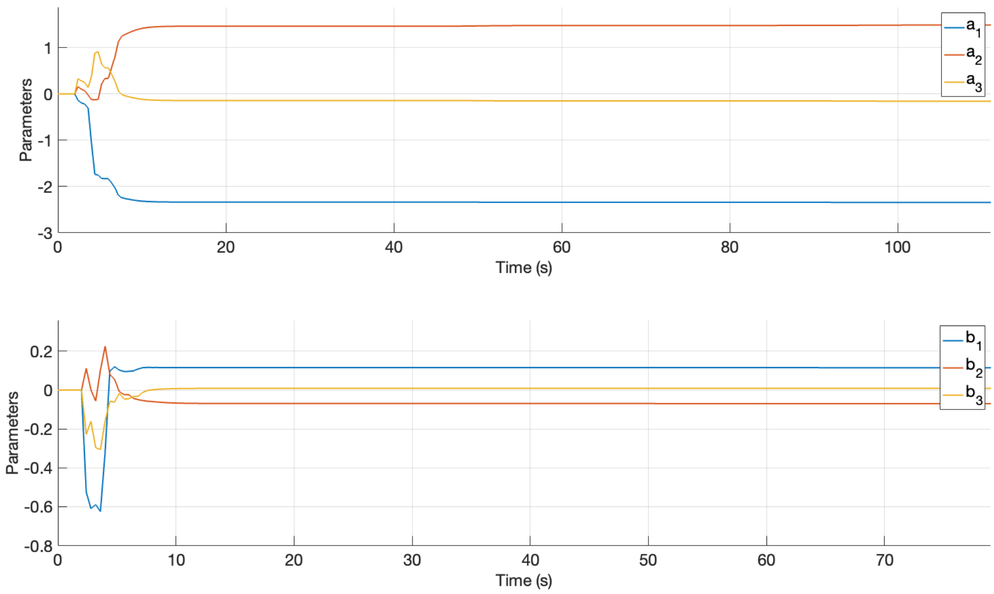

Figure 37: System parameters with integral windup fix

The code for this section is available at assignment2/part2/STR1_indirect.m. By changing the $distrubance = 0$ to 1 we can introduce disturbance to the system. Variables $distrubance\_fix$ and $integral\_fix$ are used to enable respective system fixes. $vlimit$ is used to limit the control output when fixing integral windup.