Praveen Ravishankar 04/14/2021
ME 6403 HW SET #7

1) <u>Deriving Continuous-time State Equations:</u>

$$M\ddot{y} + K(y-q) = 0; \quad Ri + K_b\dot{q} = u; \quad F_{motor} = K_f i = K(q-y)$$

$$\Rightarrow i = \frac{K}{K_f}(q-y) \Rightarrow \frac{RK}{K_f}(q-y) + K_b\dot{q} = u$$

$$\Rightarrow \dot{q} = \left(\frac{RK}{K_bK_f}\right)y - \left(\frac{RK}{K_bK_f}\right)q + \left(\frac{1}{K_b}\right)u$$

$$\Rightarrow \ddot{y} = \left(\frac{K}{M}\right)q - \left(\frac{K}{M}\right)y$$

$Let\ x_1 = y, x_2 = q, x_3 = \dot{y}$:

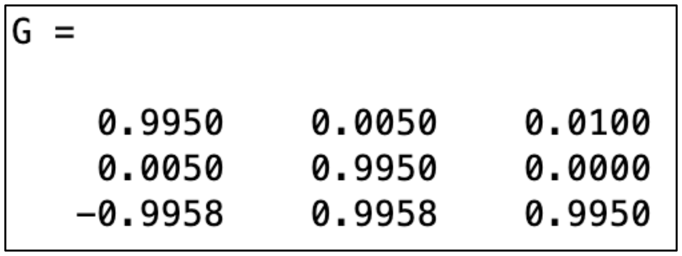
$$\begin{bmatrix}\dot{x_1}\\ \dot{x_2}\\ \dot{x_3}\end{bmatrix} = \begin{bmatrix}0 & 0 & 1\\ \frac{RK}{K_bK_f} & \frac{-RK}{K_bK_f} & 0\\ \frac{K}{M} & \frac{-K}{M} & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{K_b}\\ 0\end{bmatrix}u$$

$$\begin{bmatrix}\dot{x_1}\\ \dot{x_2}\\ \dot{x_3}\end{bmatrix} = \begin{bmatrix}0 & 0 & 1\\ 0.5 & -0.5 & 0\\ 100 & -100 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}0\\ 0.1\\ 0\end{bmatrix}u$$

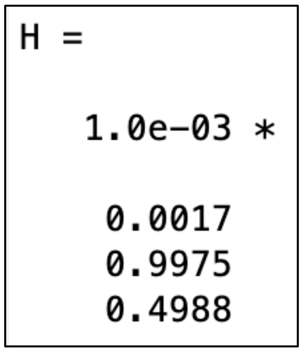
Discretizing State-Space Equations:

Using MATLAB's c2d() command on the continuous-time system above (with a sampling time of $T_s = 0.01$ s) yielded in the following:

$$\begin{bmatrix} x_1^+ \\ x_2^+ \\ x_3^+ \end{bmatrix} = G \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + Hu, where:$$



```
G =

    0.9950    0.0050    0.0100
    0.0050    0.9950    0.0000
   -0.9958    0.9958    0.9950
```



```
H =

   1.0e-03 *

    0.0017
    0.9975
    0.4988
```

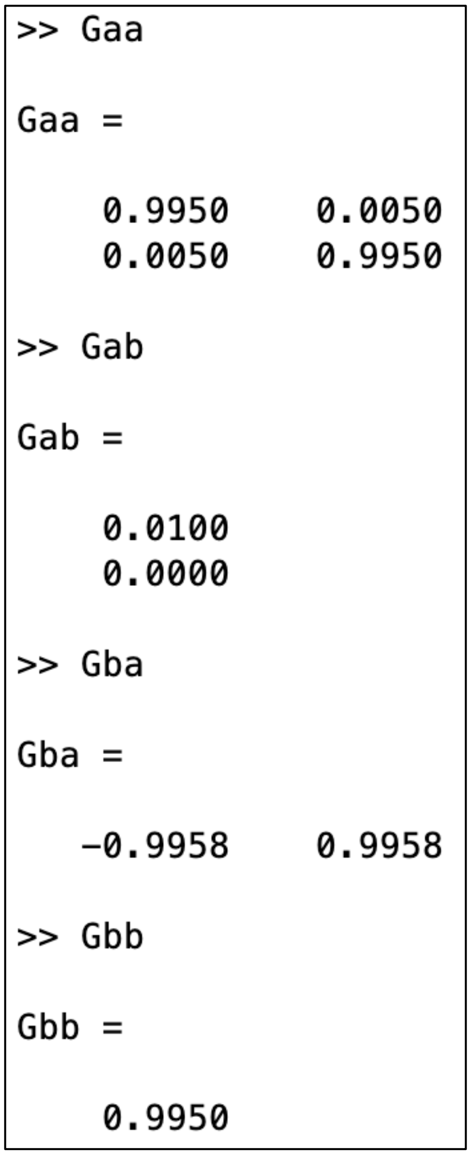
2) $Availability\ in\ q\ and\ i\ \Longrightarrow Availability\ in\ q\ and\ y$
$(since\ q\ and\ y\ are\ related\ through\ i\ in\ the\ following\ equation\ K_f i = K(q - y))$

$Hence, C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} and\ \frac{dy}{dt}\ is\ the\ only\ unobservable\ variable.$

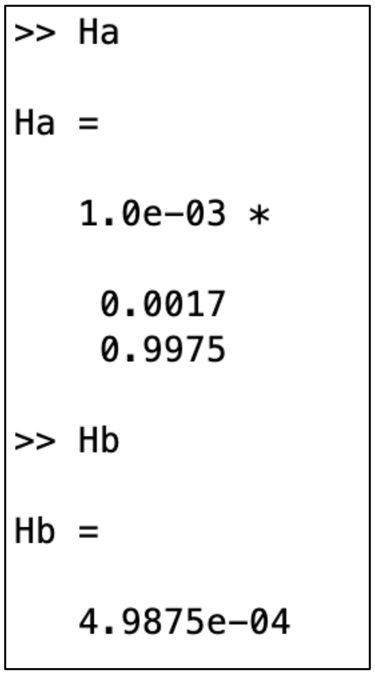
The system is divided as follows:

$$\begin{bmatrix} \mathbf{x}_a \\ \hdashline \mathbf{x}_b \end{bmatrix}^+ = \left[\begin{array}{c:c} G_{aa} & G_{ab} \\ \hdashline G_{ba} & G_{bb} \end{array}\right] \begin{bmatrix} \mathbf{x}_a \\ \hdashline \mathbf{x}_b \end{bmatrix} + \begin{bmatrix} H_a \\ \hdashline H_b \end{bmatrix} \mathbf{u}$$

Where:

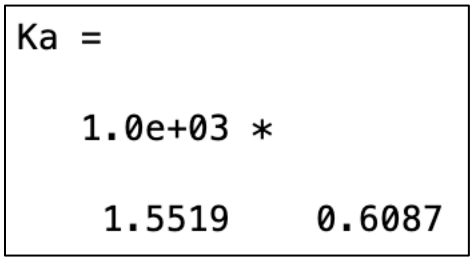


```
>> Gaa

Gaa =

    0.9950    0.0050
    0.0050    0.9950

>> Gab

Gab =

    0.0100
    0.0000

>> Gba

Gba =

   -0.9958    0.9958

>> Gbb

Gbb =

    0.9950
```

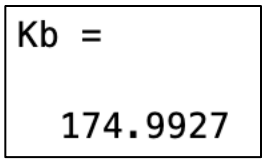


```
>> Ha

Ha =

   1.0e-03 *

    0.0017
    0.9975

>> Hb

Hb =

   4.9875e-04
```

Selecting the desired *continuous* closed-loop poles to be (-25 +/- 25j), (-25) yields in the following state-feedback controller gains:

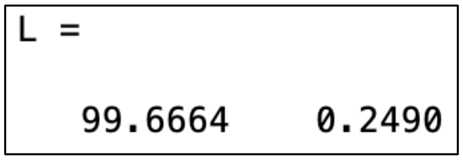


```
Ka =

   1.0e+03 *

    1.5519    0.6087
```



```
Kb =

  174.9927
```

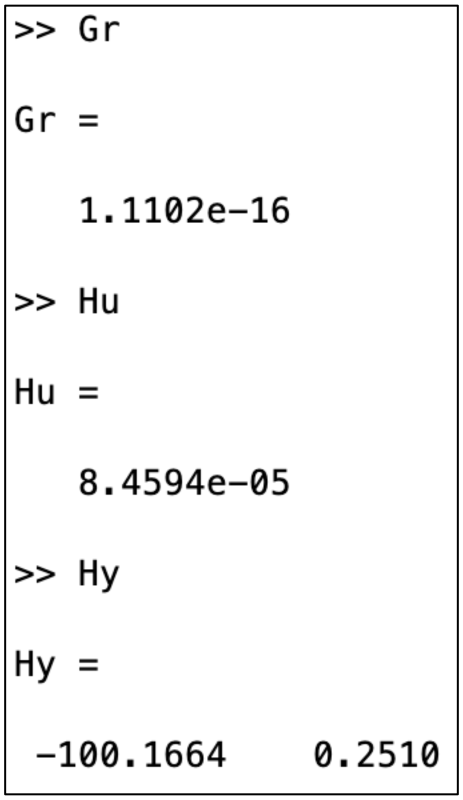
Additionally, the desired observer was desired to be a deadbeat observer (all poles equaling zero), resulting in the following observer gains:



```
L =

   99.6664    0.2490
```

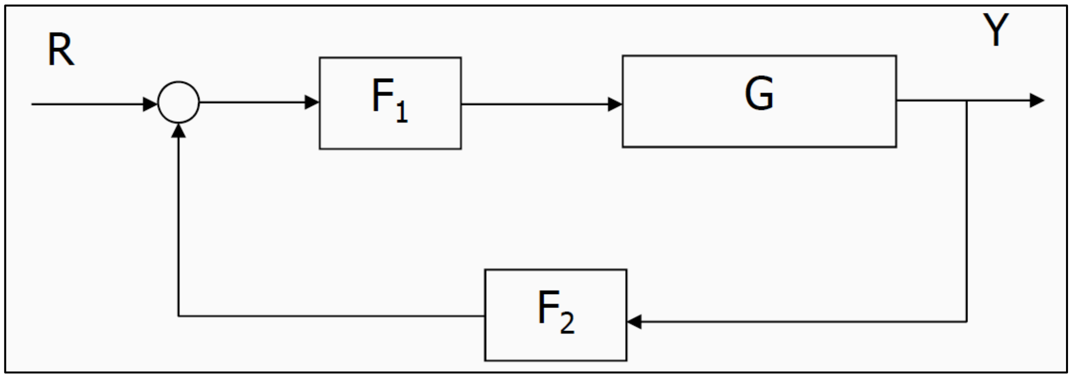
Since deadbeat-control is used, integral control is **NOT** necessary for this case because deadbeat controllers are guaranteed to produce zero steady-state error and less than 2% maximum overshoot with relatively small settling times.

Furthermore, the following matrices were also calculated:



```
>> Gr

Gr =

   1.1102e-16

>> Hu

Hu =

   8.4594e-05

>> Hy

Hy =

 -100.1664    0.2510
```

The combined state feedback/reduced order observer-controlled system can be described by the following block diagram, where G is equal to the plant's transfer function:



The equations used to calculate $F_1(z)$ and $F_2(z)$ are as follows:

$$F_1(z) = \left[1 + \mathbf{K}_b(z\mathbf{I} - \mathbf{G}_r)^{-1}\mathbf{H}_u\right]^{-1}$$
$$F_2(z) = \mathbf{K}_b(z\mathbf{I} - \mathbf{G}_r)^{-1}\mathbf{H}_y + \mathbf{K}_a + \mathbf{K}_b\mathbf{L}$$

As a result, $F_1(z)$ and $F_2(z)$ were found to be:

```
F1 =

  z - 1.079e-16
  -------------
   z + 0.0148
```

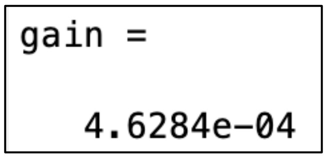
```
F2 =

  1.899e04 z - 1.753e04
  ---------------------
       z - 1.11e-16
```

This results in the following *closed-loop* transfer function Y(z)/R(z):

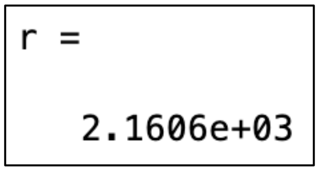
```
1.664e-06 z^4 + 6.643e-06 z^3 + 1.66e-06 z^2 - 7.615e-22 z + 7.152e-38
----------------------------------------------------------------------
  z^5 - 2.288 z^4 + 1.782 z^3 - 0.4724 z^2 + 2.62e-17 z + 1.244e-17
```

3) <u>Performing simulations assuming a constant desired output y_d=1:</u>

$$Since\ the\ desired\ output\ is\ a\ constant, r = \frac{y_d}{gain}, where:$$

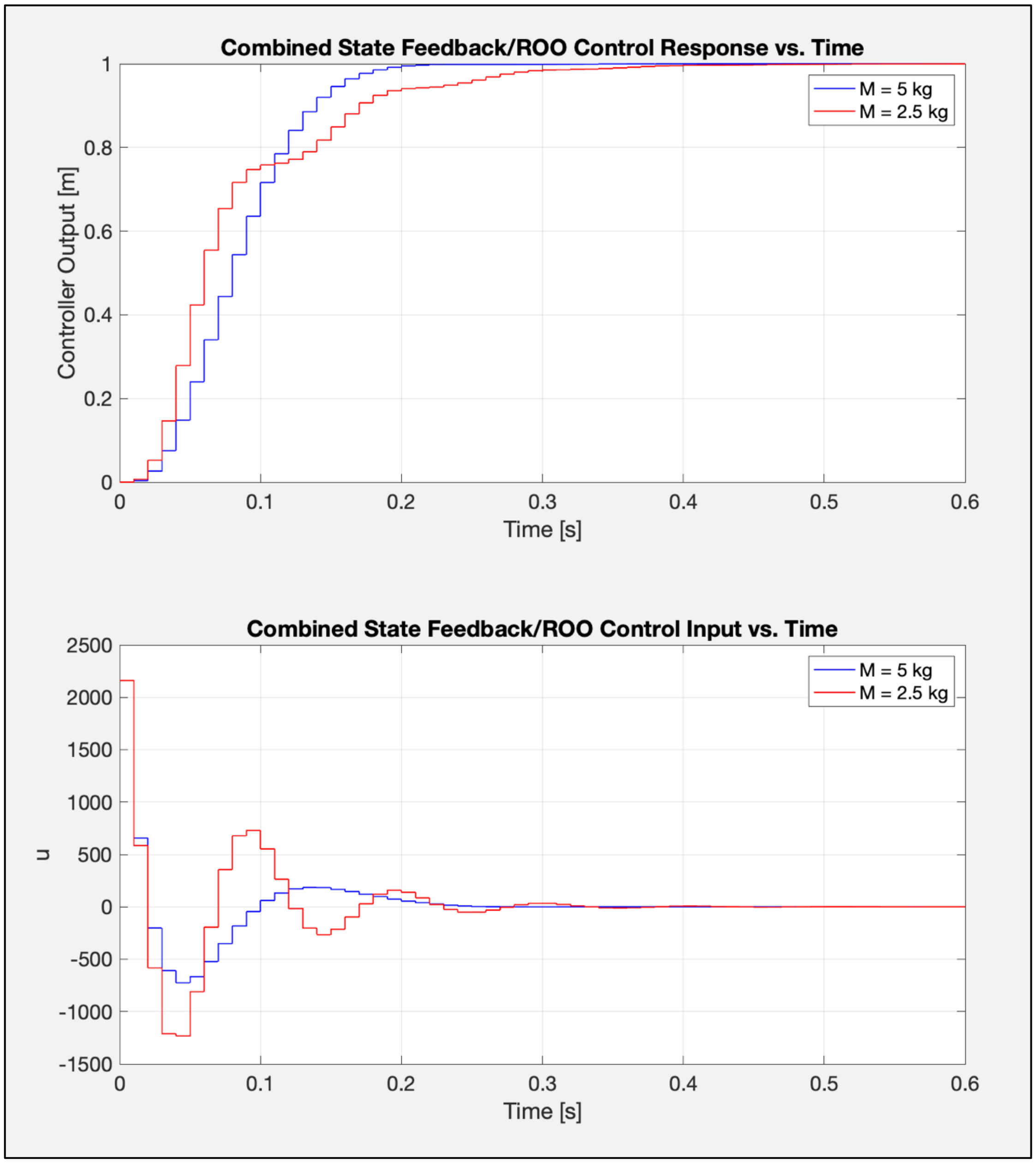


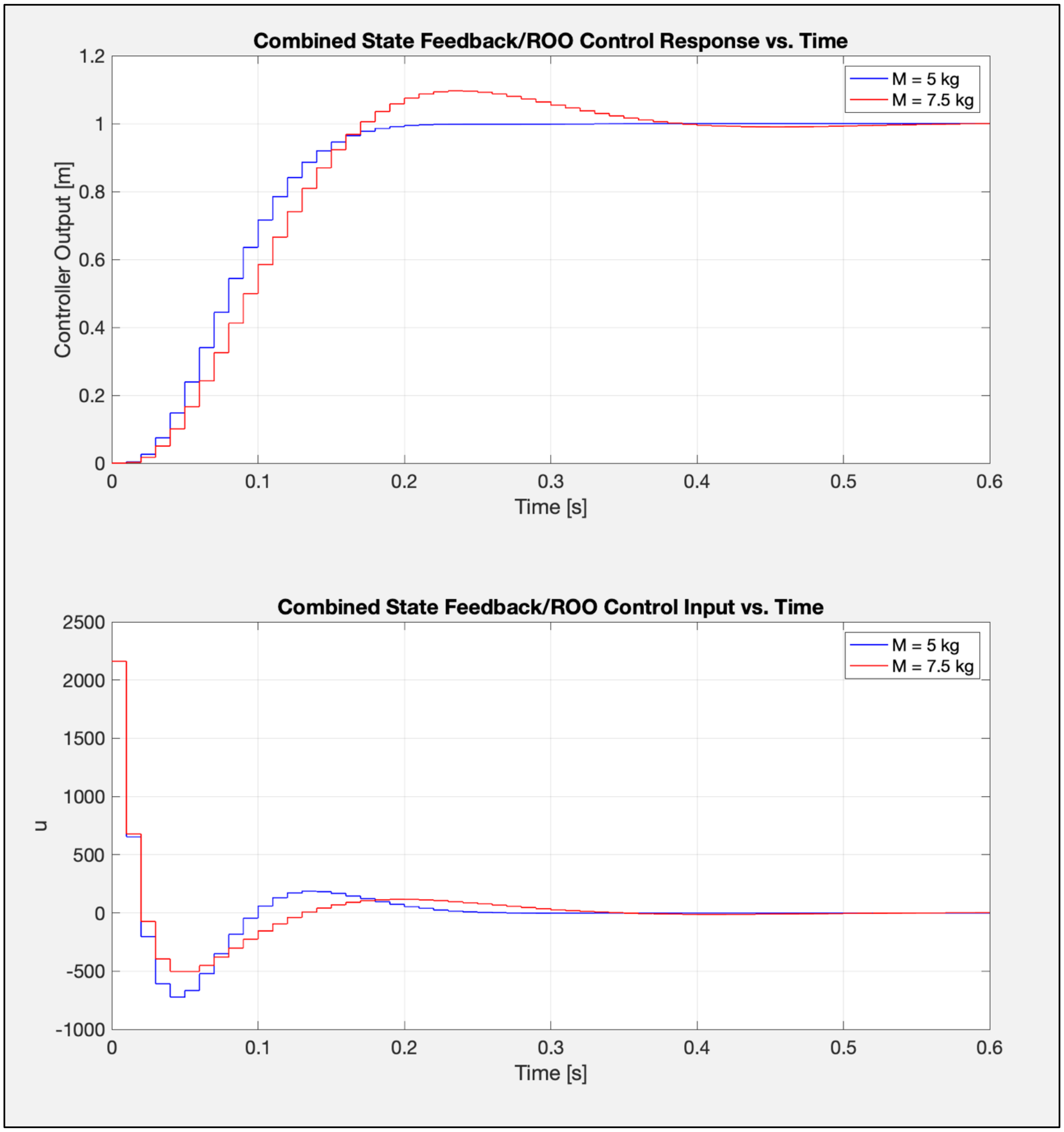
```
gain =

   4.6284e-04
```



```
r =

   2.1606e+03
```

Simulation Plots:





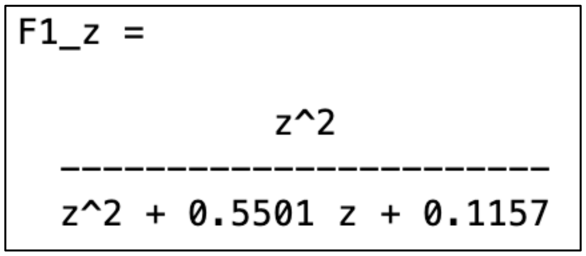
As seen in the above simulation plots, the controlled system was simulated for three different mass values: 2.5 kg, 5 kg, and 7.5 kg. As the controller was intended to be designed for a mass value of 5 kg, mass values of 2.5 kg and 7.5 kg were simulated to test the robustness of the controller by varying the expected mass by 50%. Firstly, for the 5 kg case, it's seen that the controller satisfies the design requirements: settling time of less than 0.2 seconds, having a maximum overshoot of less than 10%, and zero steady-state error. This indicates that the designed controller functions effectively for the given system model. Additionally, it's observed that the controlled system roughly satisfies these design requirements for the 2.5 kg and 7.5 kg cases as well, indicating the robustness of the controller of up to 50% variation in mass M.

4) Polynomial Control:

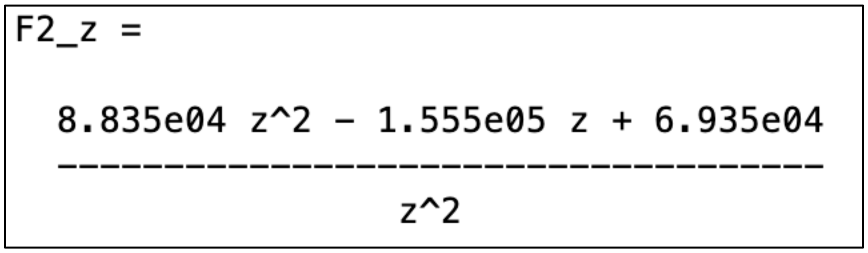
$$\Phi^{\text{des}}(z) = z^3 - 2.288z^2 + 1.782z - 0.4724\ (for\ desired\ poles\ being - 25 \pm 25j, -25)$$

$$\lambda(z) = z^2\ (for\ a\ deadbeat\ observer)$$

Since the system controlled by a polynomial controller can be expressed by the same block diagram as the combined state feedback/reduced order observer-controlled system, the designed polynomial controller's feedforward ($F_1(z)$) and feedback ($F_2(z)$) transfer functions are as follows:



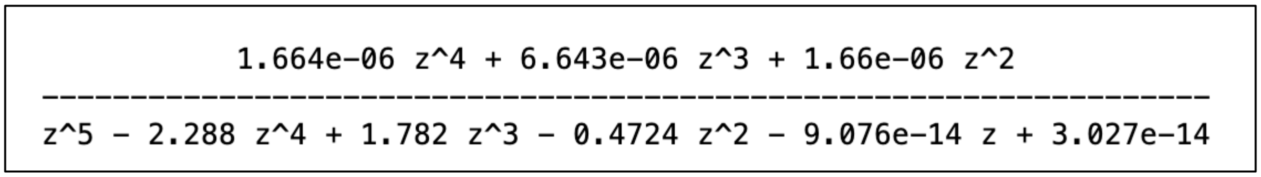
```
F1_z =

            z^2
   ---------------------
   z^2 + 0.5501 z + 0.1157
```



```
F2_z =

  8.835e04 z^2 - 1.555e05 z + 6.935e04
  ------------------------------------
                  z^2
```

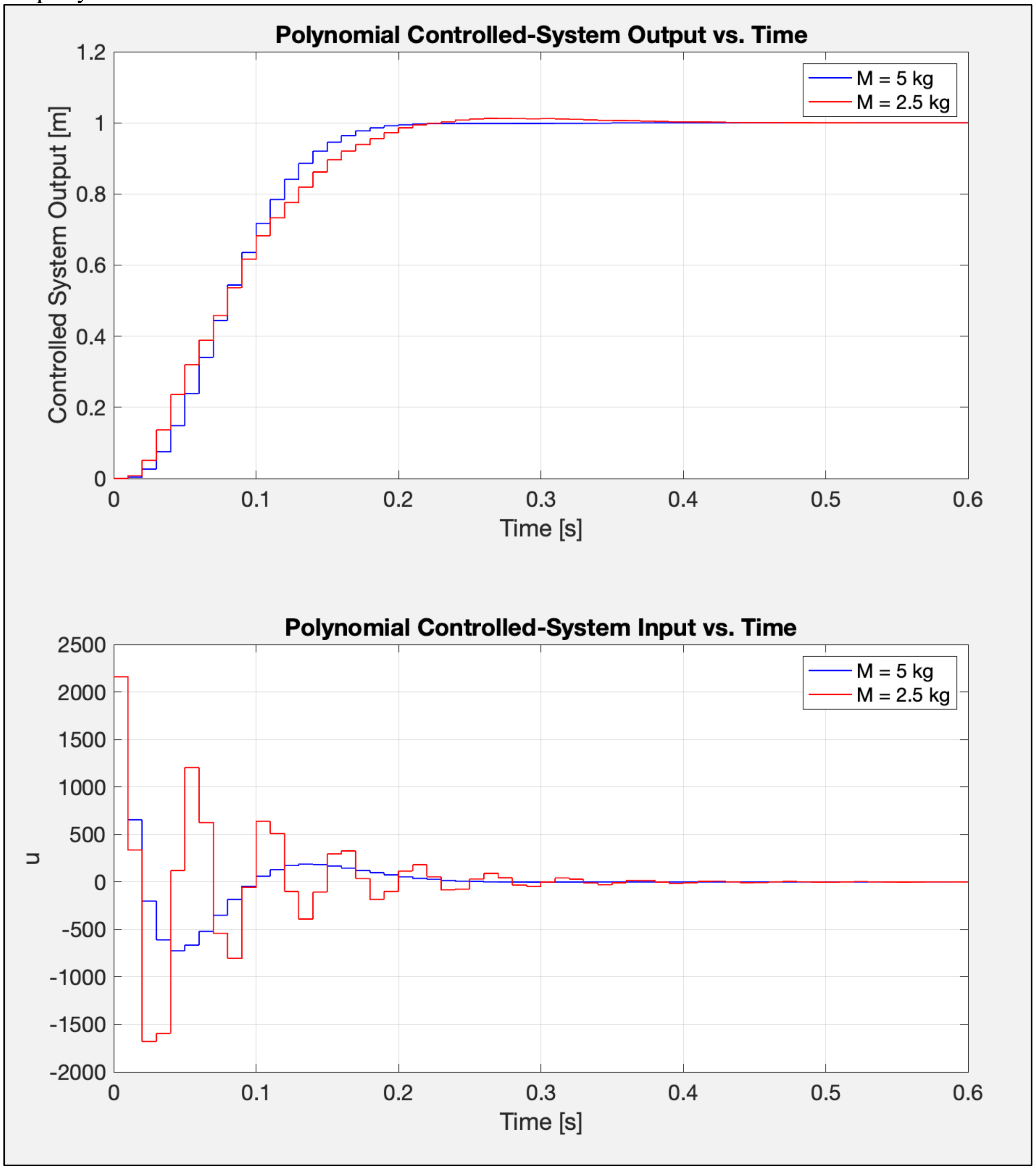
The process of creating $F_1(z)$ and $F_2(z)$ is performed and outlined in my MATLAB code attached at the end of this document.

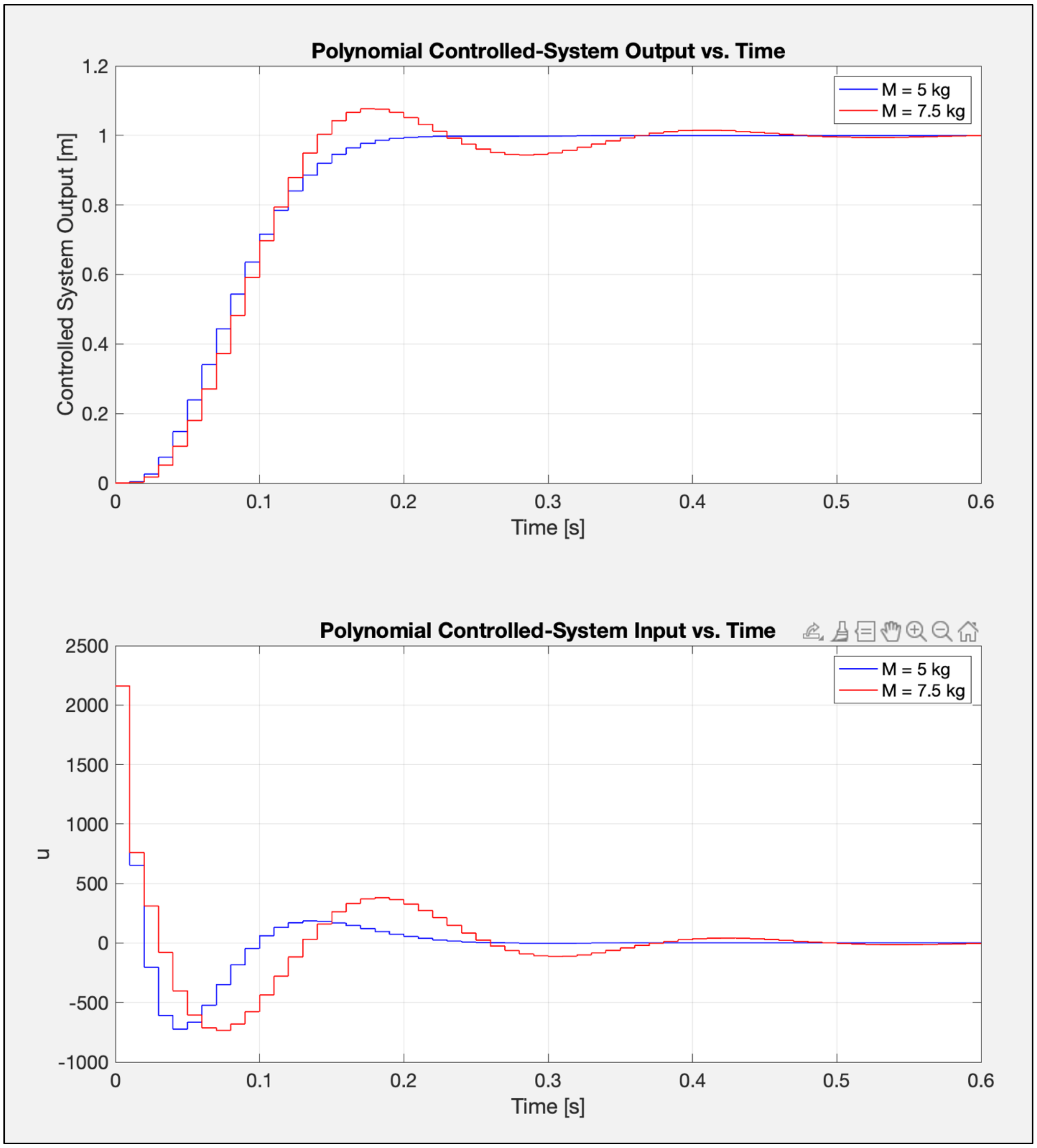
Furthermore, the $F_1(z)$ and $F_2(z)$ transfer functions from above result in the following *closed-loop* transfer function Y(z)/R(z) (which is extremely similar to the CLTF derived for the combined state feedback/ROO-controlled system):



```
            1.664e-06 z^4 + 6.643e-06 z^3 + 1.66e-06 z^2
  ---------------------------------------------------------------
  z^5 - 2.288 z^4 + 1.782 z^3 - 0.4724 z^2 - 9.076e-14 z + 3.027e-14
```

The simulated results of the system with this specific polynomial controller for a constant desired output y_d=1 are as follows:





Firstly, for the 5 kg case, it's seen that the controller satisfies the design requirements: settling time of less than 0.2 seconds, having a maximum overshoot of less than 10%, and zero steady-state error. This indicates that the designed polynomial controller functions effectively for the given system model. Additionally, it's observed that the controlled system roughly satisfies these design requirements for the 2.5 kg and 7.5 kg cases as well, indicating the robustness of the controller of up to 50% variation in mass M.

5) Adding a ZPET Controller (Q(z)) to the polynomial controller designed above:

```
3.725 z^6 - 7.523 z^5 + 4.35 z^4 + 0.0222 z^3 - 0.4724 z^2 - 5.708e-16 z - 3.469e-16
-----------------------------------------------------------------------------------
                         3.715e-05 z^6 + 9.947e-06 z^5
```

The process by which Q(z) was created is performed and described further in my MATLAB code.

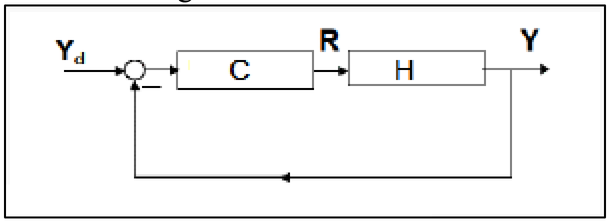
6) Adding a stable repetitive controller to the ZPET Controller above:

$$C(z) = K_l \frac{Q(z)}{z^N - 1}$$

For $K_l = 0.8$, $N = 200$, the controller transfer function C(z) was found to be:

```
2.98 z^6 - 6.019 z^5 + 3.48 z^4 + 0.01776 z^3 - 0.3779 z^2 - 4.566e-16 z - 2.776e-16
------------------------------------------------------------------------------------
        3.715e-05 z^206 + 9.947e-06 z^205 - 3.715e-05 z^6 - 9.947e-06 z^5
```

Stable Repetitive Controller Block Diagram:



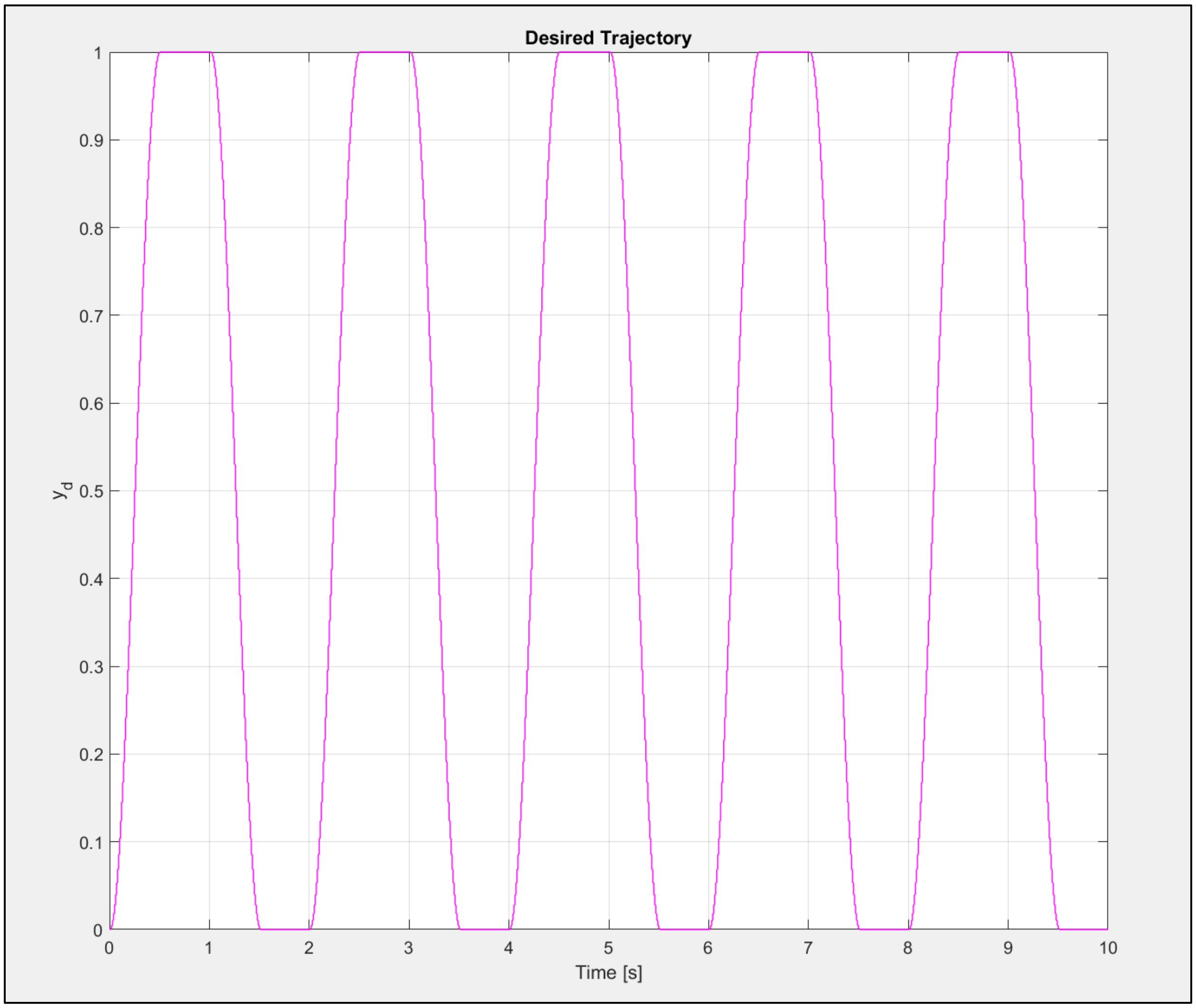
Following the block diagram above resulted in the following *closed-loop* transfer function $Y(z)/Y_d(z)$:

```
4.958e-06 z^10 + 9.785e-06 z^9 - 2.925e-05 z^8 + 1.316e-05 z^7 + 5.265e-06 z^6

               - 2.481e-06 z^5 - 6.272e-07 z^4 - 2.602e-21 z^3 - 4.606e-22 z^2

-------------------------------------------------------------------------------

3.715e-05 z^211 - 7.505e-05 z^210 + 4.343e-05 z^209 + 1.77e-07 z^208

        - 4.699e-06 z^207 - 5.71e-21 z^206 - 3.451e-21 z^205 - 3.715e-05 z^11

        + 8.001e-05 z^10 - 3.365e-05 z^9 - 2.943e-05 z^8 + 1.786e-05 z^7

        + 5.265e-06 z^6 - 2.481e-06 z^5 - 6.272e-07 z^4 - 2.602e-21 z^3

                                                             - 4.606e-22 z^2
```

7) Desired trajectory for the controlled system to follow:

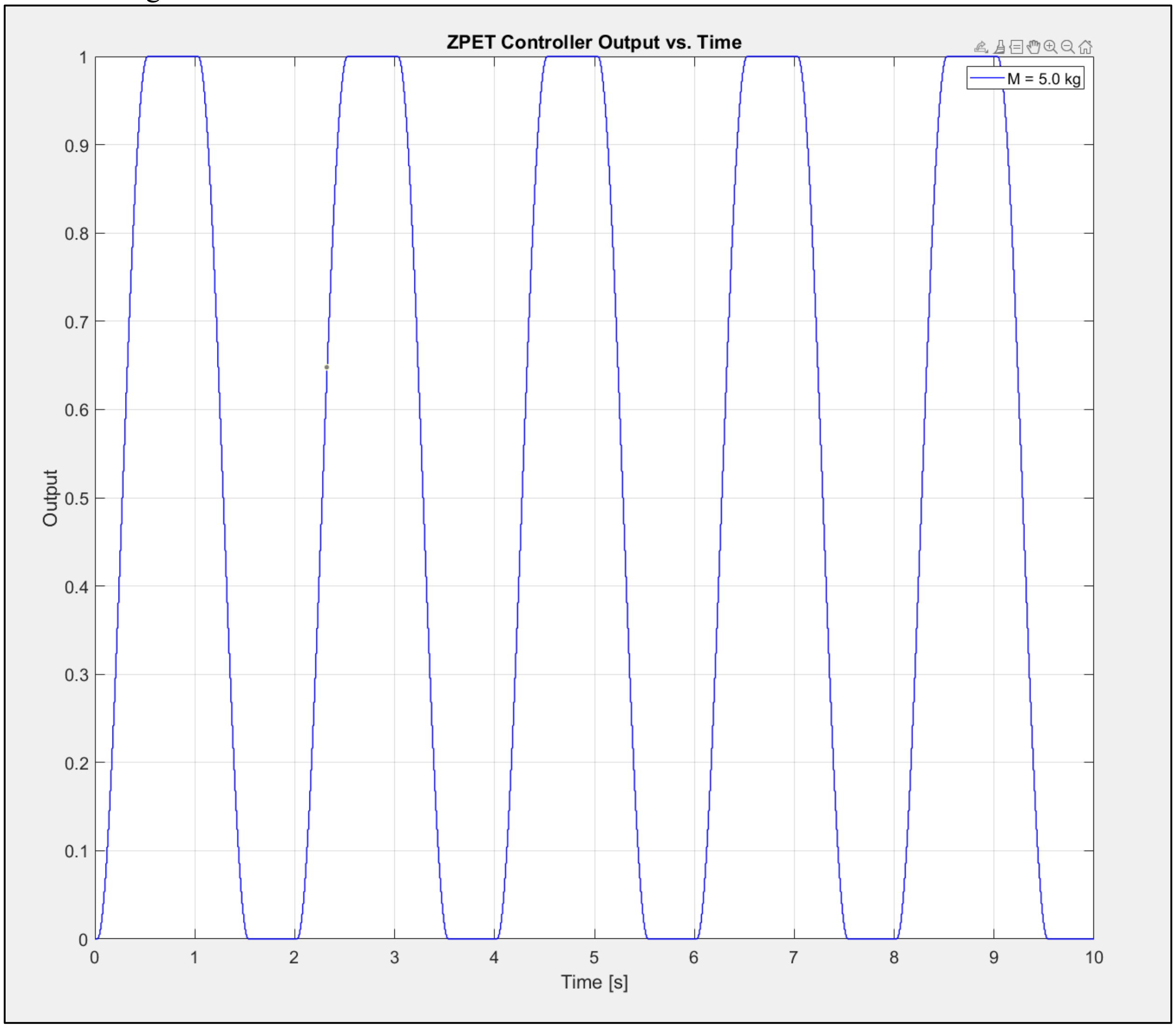
$$y(t) = \begin{Bmatrix} 4t^2(3-4t) & 0 \le t \le 0.5 \\ 1 & 0.5 \le t \le 1 \\ 1 - y_d(t-1) & 1 \le t \le 2 \\ y_d(t+2) & t \ge 2 \end{Bmatrix}$$

All simulations were performed for 5 cycles (10 seconds) of y(t) **(ALL OUTPUT UNITS ARE IN METERS)**:

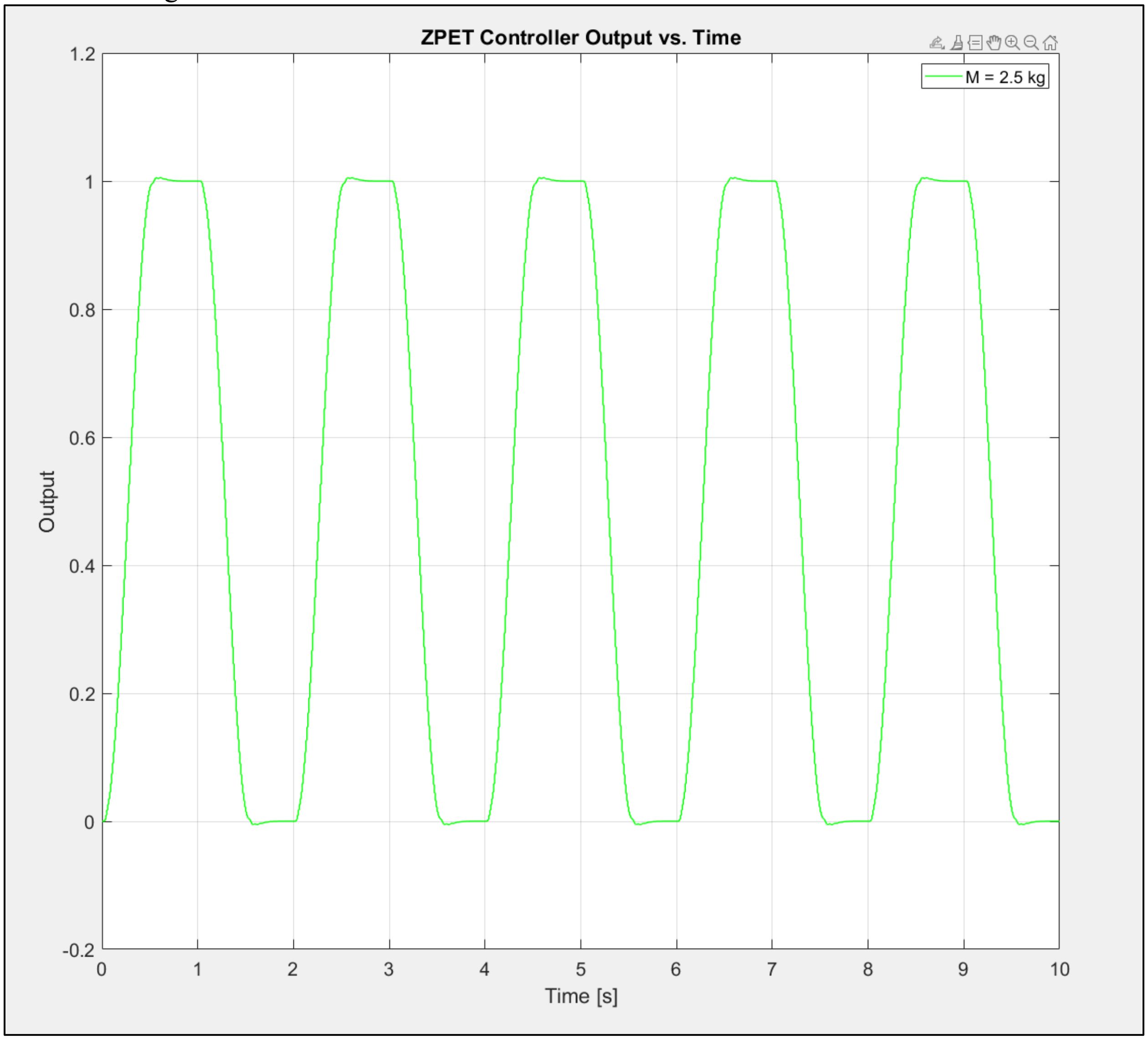


Performing simulations on the ZPET-controlled system to track the above trajectory:

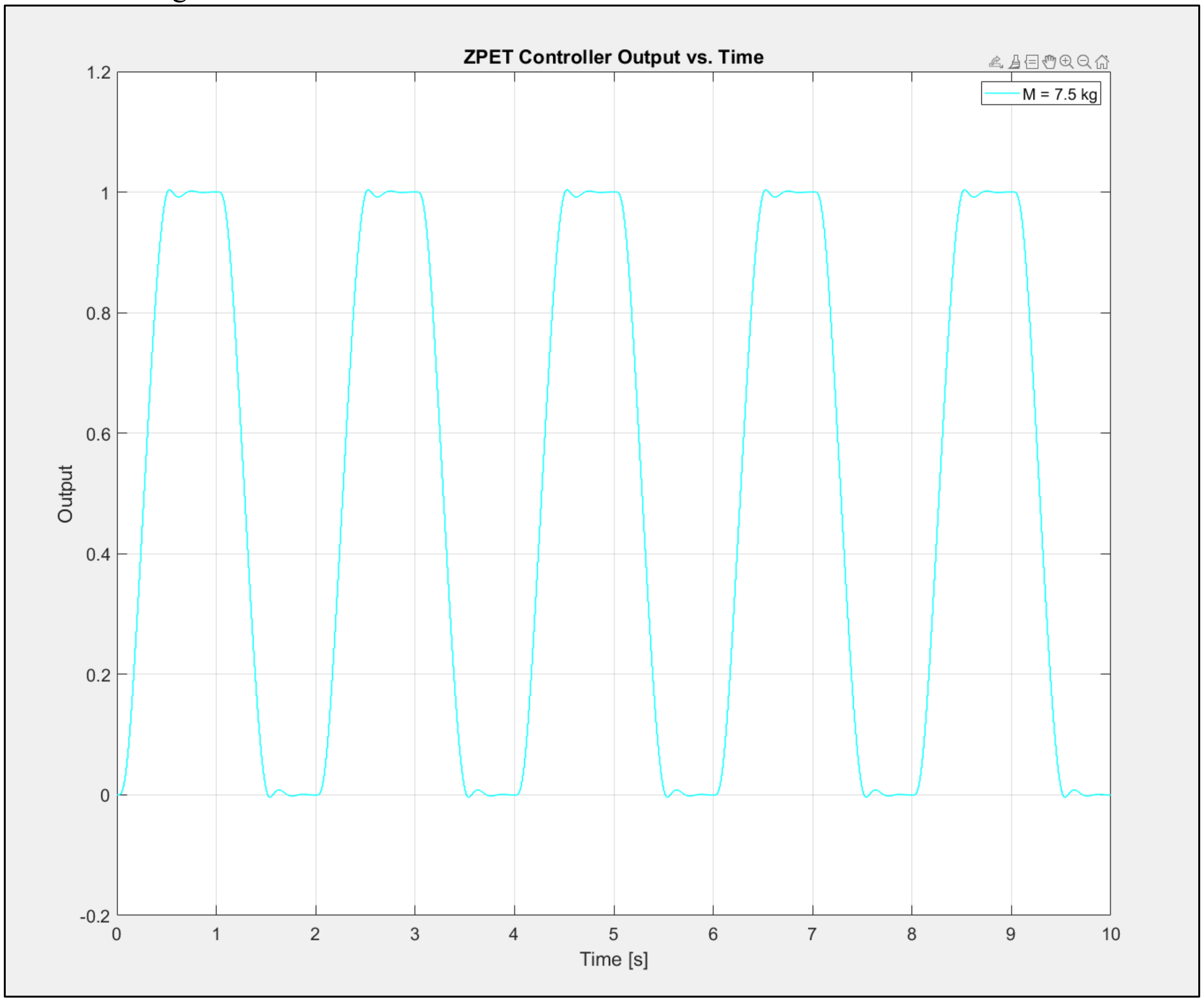
For M = 5 kg:



For M = 2.5 kg:


ZPET Controller Output vs. Time
M = 2.5 kg
Output
Time [s]
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
1
2
3
4
5
6
7
8
9
10

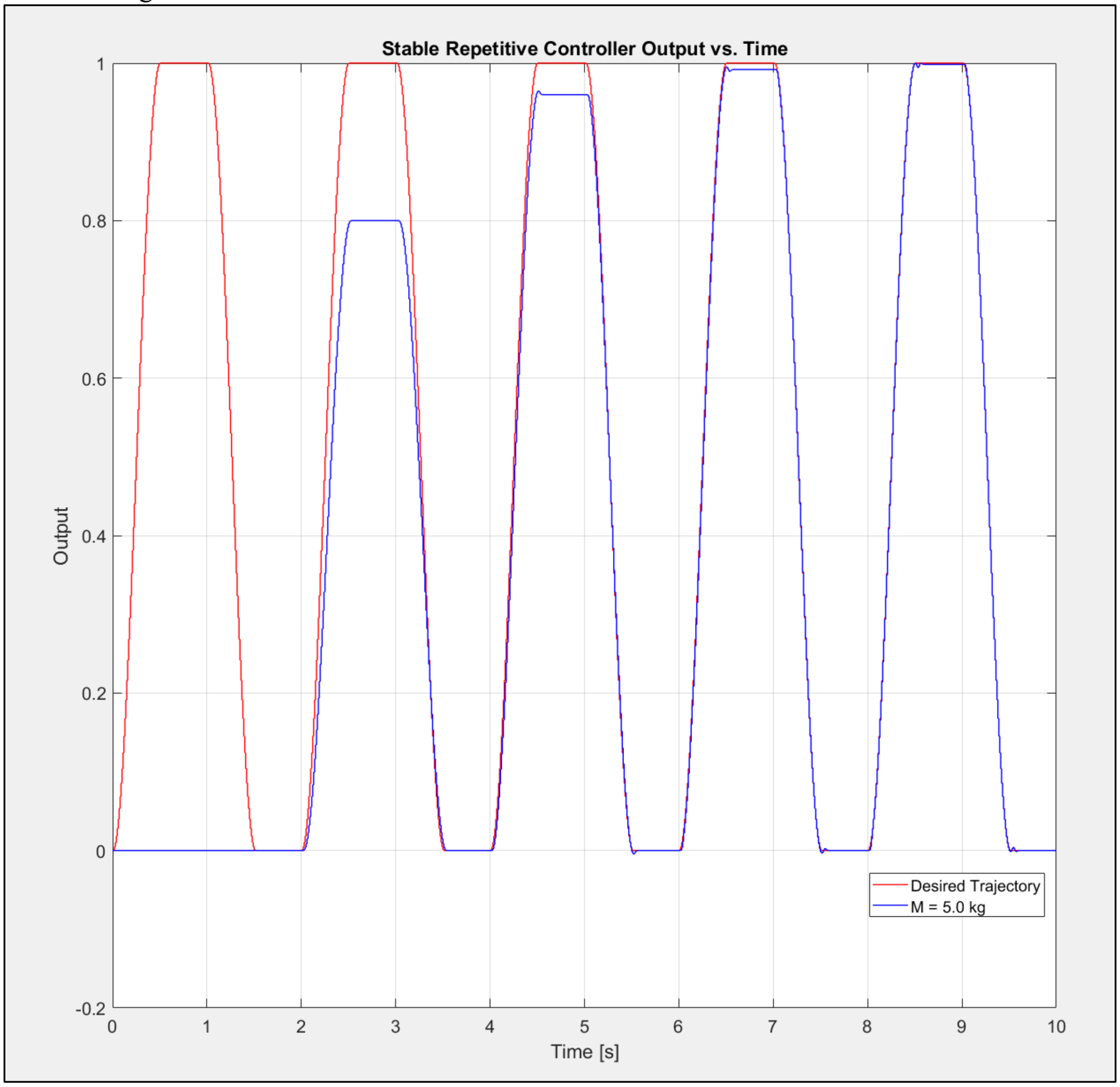
For M = 7.5 kg:



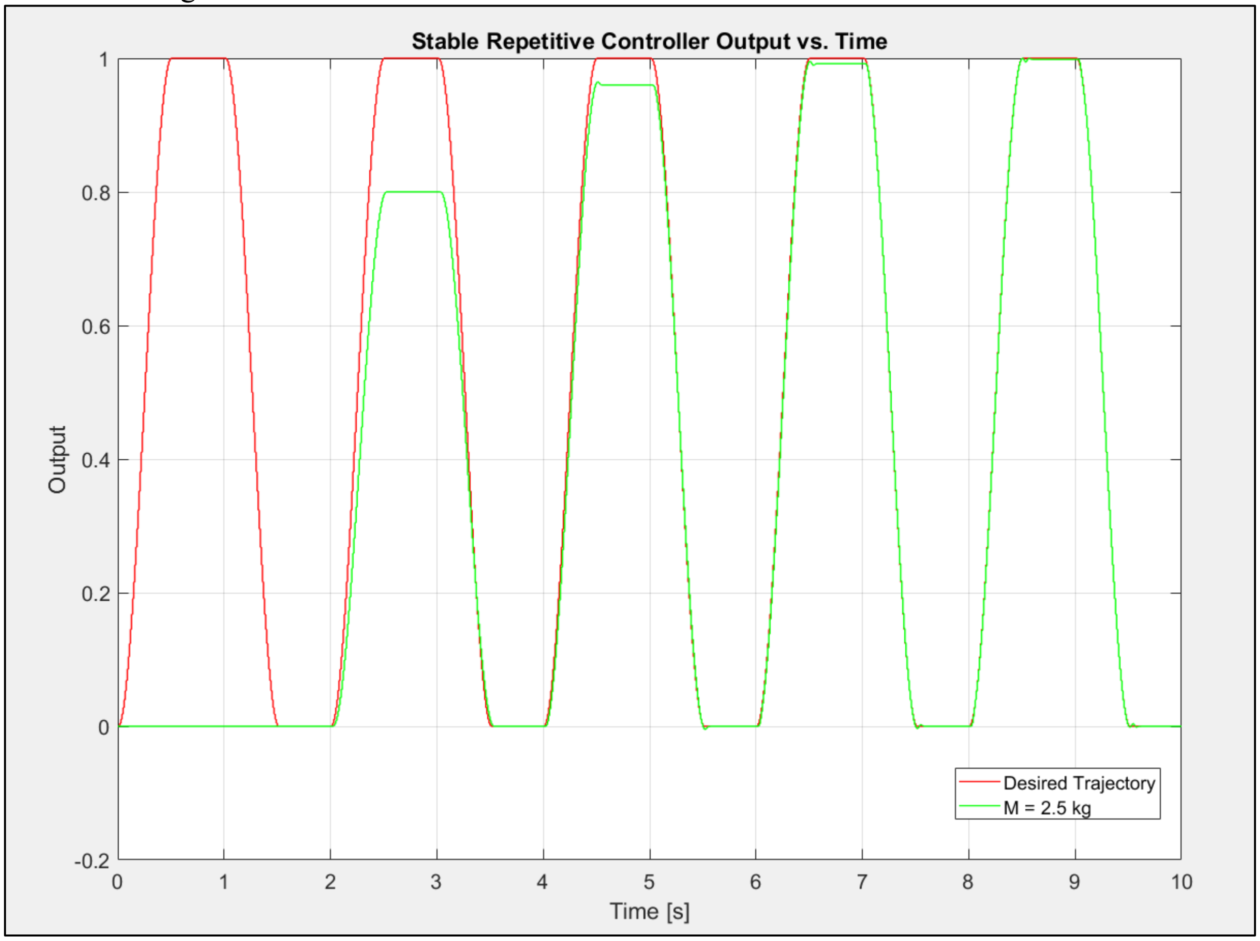
By comparing the above plots of the ZPET Controller Output with the plot of the desired trajectory, it's seen that for an expected mass value of 5 kg (the mass value that the ZPET controller was designed for), the ZPET controller tracks the desired output extremely well. Furthermore, it's seen that the ZPET Controller tracks the desired output fairly well for mass values of 2.5 kg and 7.5 kg as well; although relatively small tracking errors are present, these errors stabilize rather quickly over time. Hence, it's indicated that this controller is indeed robust for mass values varying up to 50%.

Performing simulations on the stable repetitive-controlled system to track the above trajectory:

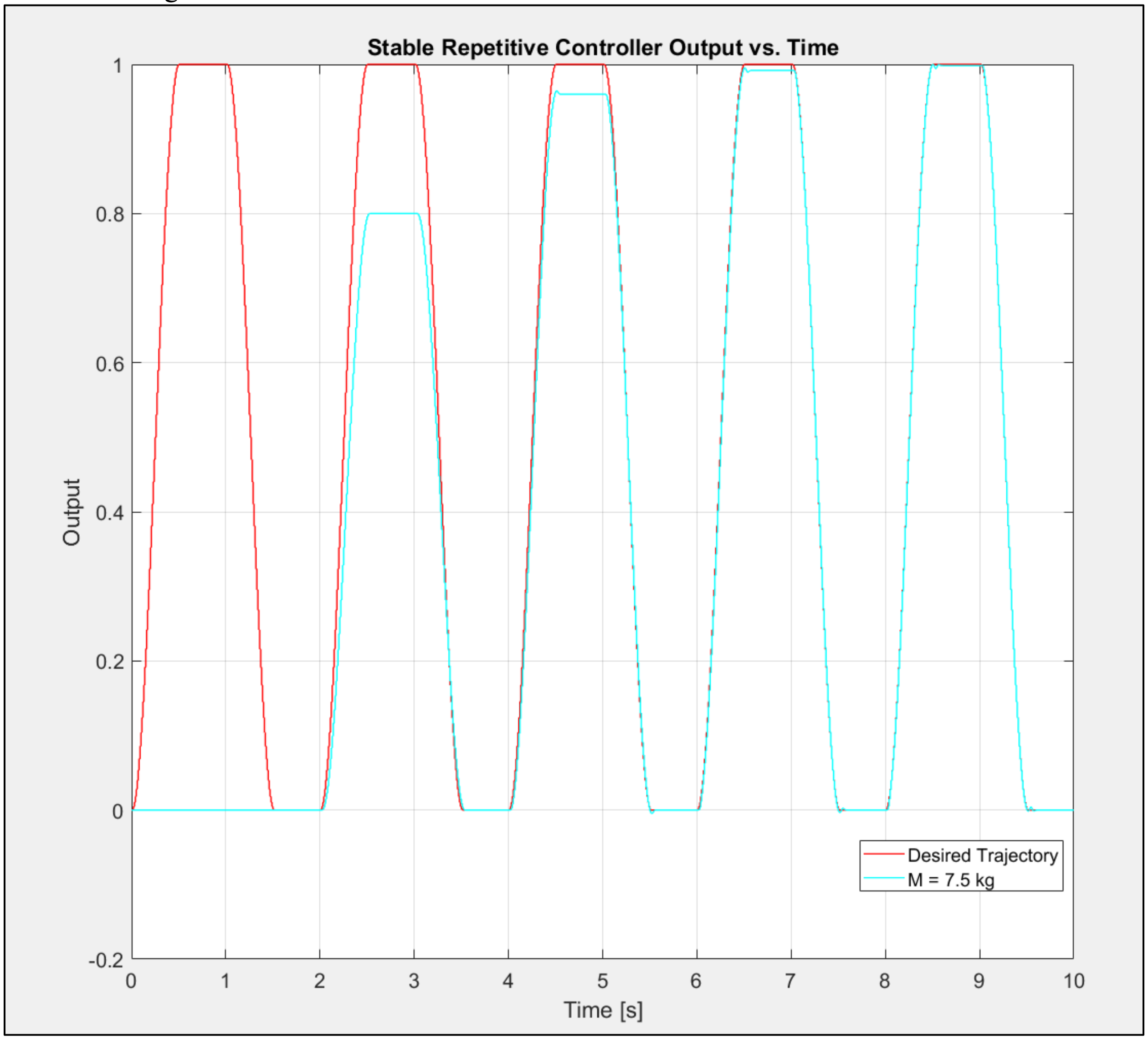
For M = 5 kg:



For M = 2.5 kg:


Stable Repetitive Controller Output vs. Time
Output
Time [s]
Desired Trajectory
M = 2.5 kg
1
0.8
0.6
0.4
0.2
0
-0.2
0
1
2
3
4
5
6
7
8
9
10

For M = 7.5 kg:



All simulations for the stable repetitive controller were performed for a K_I value of 0.8 and an N value of 200. Those values were chosen as they were the values that allowed the controller to track the desired output relatively optimally (in terms of efficiency and error-avoidance) when the mass is equal to 5 kg. By observing the plot for the 5 kg case, it's seen that the stable repetitive controller tracks the desired trajectory extremely well, properly following it after just 5 cycles with minimal errors. Additionally, by observing the plots for the 2.5 kg and 7.5 kg cases, it's additionally noticed that this controller tracks the desired trajectory really effectively as well, indicating that the stable repetitive controller is robust in various of the mass M of up to 50%.

MATLAB Code:

```matlab
close all
clear all
clc

M = 5; % Mass in kg
M_var = 7.5; % Variable mass in kg
Kc = 500; % Spring constant in N/m
Kb_c = 10; % Motor back-emf constant
Kf = 10; % Motor force constant
R = 0.1; % Motor armature resistance in Ohms

%% Part 1:

% Continuous-time State Matrices:
A = [0 0 1; ((R*Kc)/(Kb_c*Kf)) ((-R*Kc)/(Kb_c*Kf)) 0; (-Kc/M) (Kc/M) 0];
B = [0; (1/Kb_c); 0];
%C = [1 0 0];
C = [1 0 0; 0 1 0];
%D = 0;
D = [0; 0];
sysc = ss(A, B, C, D);

% Discretizing State-Space Equation:
T = 0.01;
sysd = c2d(sysc, T);
G = sysd.A;
H = sysd.B;

% Mass-Varied System:
M_sys = M_var; % Varied mass of the system (can be changed to see how the
controller behaves)
A_sys = [0 0 1; ((R*Kc)/(Kb_c*Kf)) ((-R*Kc)/(Kb_c*Kf)) 0; (-Kc/M_sys)
(Kc/M_sys) 0];
B_sys = [0; (1/Kb_c); 0];
%C_sys = [1 0 0];
C_sys = [1 0 0; 0 1 0];
%D_sys = 0;
D_sys = [0; 0];
P_s = ss(A_sys, B_sys, C_sys, D_sys);
P_z = c2d(P_s, T);

%% Part 2 - State-Feedback Controller with Reduced-Order Deadbeat Observer:

% Partitioning the matrices for the reduced-order observer:
n_o = 2;
Gaa = G((1:n_o), 1:n_o);
Gab = G((1:n_o), ((n_o+1):end));
Gba = G(((n_o+1):end), (1:n_o));
Gbb = G(((n_o+1):end), ((n_o+1):end));
Ha = H(1:n_o);
Hb = H((n_o+1):end);

%%% State-Feedback Controller:
pd_k = [(-25 + 25i), (-25 - 25i), -25];
pd_k = exp(pd_k * T);

K = place(G, H, pd_k); % Calculating state-feedback gains

% Calculating reference input gain:
G_cl = G - (H*K);
gain = C * inv(eye(3) - G_cl) * H;
gain = gain(1);

Ka = K(1:n_o);
Kb = K((n_o+1):end);

%%% Reduced-Order Deadbeat Observer:

% Forming the deadbeat poles:
if(n_o == 1)
    pd_o = [0, 0];
elseif(n_o == 2)
    pd_o = 0;
end

L = place(Gbb', Gab', pd_o);
L = L';

% Calculating more system matrices:
Gr = Gbb - (L*Gab);
Hu = Hb - (L*Ha);
Hy = (Gr*L) + Gba - (L*Gaa);

%%% Feedback System:
F2 = ss(Gr, Hy, Kb, (Ka+(Kb*L)), T);
%F2 = tf(F2);
%F2 = F2(1);

%%% Feedforward System:
ff1 = ss(Gr, Hu, Kb, 0, T);
F1 = feedback(1, ff1);
%F1 = tf(F1);

%% Part 3:

% Controlled System:
G_cltf = feedback((F1*sysd), F2);
G_cltf_m = feedback((F1*P_z), F2);

t = [0:T:0.6]; % Time vector for linear simulation

% Calculating the reference input:
yd = 1;
r = yd./gain; % Reference input r
u = ones(1, length(t)).*r; % Reference input vector

% Actual system:
[y, tOut, x_states] = lsim(G_cltf, u, t);
```

```
y = y(:, 1);

U_R_z = F1/(1 + (F2*F1*sysd));
u_input = ones(1, length(t)) .* r;
[u_ctrl, ~,  ~] = lsim(U_R_z, u_input, t);

% Varied-mass system:
[y_m, ~, x_states_m] = lsim(G_cltf_m, u, t);
y_m = y_m(:, 1);

U_R_z_m = F1/(1 + (F2*F1*P_z));
[u_ctrl_m, ~,  ~] = lsim(U_R_z_m, u_input, t);

% Plotting Results:
lgnd_lbl = sprintf('M = %0.1f kg', M_var);

figure
subplot(2, 1, 1)
stairs(tOut, y, 'b', 'LineWidth', 1)
hold on
stairs(tOut, y_m, 'r', 'LineWidth', 1)
xlabel('Time [s]')
ylabel('Controller Output [m]')
title('Combined State Feedback/ROO Control Response vs. Time')
legend('M = 5 kg', lgnd_lbl)
set(gca, 'FontSize', 15)
grid on

subplot(2, 1, 2)
stairs(tOut, u_ctrl, 'b', 'LineWidth', 1)
hold on
stairs(tOut, u_ctrl_m, 'r', 'LineWidth', 1)
title('Combined State Feedback/ROO Control Input vs. Time')
xlabel('Time [s]')
ylabel('u')
legend('M = 5 kg', lgnd_lbl)
set(gca, 'FontSize', 15)
grid on

%% Re-defining Plant Systems:

% Actual Plant:
M_sys = 5; % Mass of the system (can be changed to see how the controller
behaves)

A_sys = [0 0 1; ((R*Kc)/(Kb_c*Kf)) ((-R*Kc)/(Kb_c*Kf)) 0; (-Kc/M_sys)
(Kc/M_sys) 0];
B_sys = [0; (1/Kb_c); 0];
C_sys = [1 0 0];
%C_sys = [1 0 0; 0 1 0];
D_sys = 0;
%D_sys = [0; 0];
P_s = ss(A_sys, B_sys, C_sys, D_sys);
P_s = tf(P_s);
P_z = c2d(P_s, T); % Discretized plant transfer function
```

```
% Mass-Varied Plant:
M_sys2 = M_var; % Mass of the system (can be changed to see how the
controller behaves)

A_sys = [0 0 1; ((R*Kc)/(Kb_c*Kf)) ((-R*Kc)/(Kb_c*Kf)) 0; (-Kc/M_sys2)
(Kc/M_sys2) 0];
B_sys = [0; (1/Kb_c); 0];
C_sys = [1 0 0];
%C_sys = [1 0 0; 0 1 0];
D_sys = 0;
%D_sys = [0; 0];
P_s = ss(A_sys, B_sys, C_sys, D_sys);
P_s = tf(P_s);
P_z_m = c2d(P_s, T); % Discretized plant transfer function

%% Part 4 - Polynomial Controller:

P_tf = tf(sysd); % Converting plant system to transfer function
P_tf = P_tf(1); % Taking only the first output (which is y)

[A_z, B_z] = tfdata(P_tf, 'v'); % Extracting numerator and denominator
coefficients from plant transfer function

% Creating phi(z):
syms z
phi_z = expand(prod(z - pd_k));
phi_coef = double(coeffs(phi_z));
phi_coef = phi_coef(end:-1:1);

lamd_z = [1 0 0]; % Lambda(z) for a deadbeat observer

% Controller coefficients:
[numc, denc] = fbcp(A_z, B_z, phi_coef); % Designing the polynomial
controller

% Creating the controller feedforward and feedback blocks:
F1_z = tf(lamd_z, denc, T); % Feedforward
F2_z = tf(numc, lamd_z, T); % Feedback

%%% Performing Linear Simulations:

% Actual system:
G_cltf_poly = feedback((F1_z*P_z), F2_z);
G_cltf_poly = minreal(G_cltf_poly);
[y_poly, tOut, ~] = lsim(G_cltf_poly, u, t);

U_R_z = F1_z/(1 + (F1_z*F2_z*P_z));
u_input = ones(1, length(t)) .* r;
[u_poly, ~,  ~] = lsim(U_R_z, u_input, t);

% Varied-mass system:
G_cltf_poly_m = feedback((F1_z*P_z_m), F2_z);
G_cltf_poly_m = minreal(G_cltf_poly_m);
```

```
[y_poly_m, ~, ~] = lsim(G_cltf_poly_m, u, t);

U_R_z_m = F1_z/(1 + (F1_z*F2_z*P_z_m));
[u_poly_m, ~,  ~] = lsim(U_R_z_m, u_input, t);

% Plotting Results:
figure
subplot(2, 1, 1)
stairs(tOut, y_poly, 'b', 'LineWidth', 1)
hold on
stairs(tOut, y_poly_m, 'r', 'LineWidth', 1)
xlabel('Time [s]')
ylabel('Controlled System Output [m]')
title('Polynomial Controlled-System Output vs. Time')
legend('M = 5 kg', lgnd_lbl)
set(gca, 'FontSize', 15)
grid on

subplot(2, 1, 2)
stairs(tOut, u_poly, 'b', 'LineWidth', 1)
hold on
stairs(tOut, u_poly_m, 'r', 'LineWidth', 1)
xlabel('Time [s]')
ylabel('u')
title('Polynomial Controlled-System Input vs. Time')
legend('M = 5 kg', lgnd_lbl)
set(gca, 'FontSize', 15)
grid on

%% Part 5 - Addition of ZPET Controller:

G_cltf_poly = tf(G_cltf_poly);
[B_z, D_z] = tfdata(G_cltf_poly, 'v'); % Extracting numerator and denominator
coefficients from polynomial-controlled transfer function
D_z_tf = tf(D_z, [1], T); % Creating transfer function using denominator
coefficients

[~, ~, gain2] = zpkdata(G_cltf_poly, 'v'); % Extracting the gain of the
transfer function

G_zeros = roots(B_z); % Obtaining all zeros of the closed loop system
B_z_plus = G_zeros(abs(G_zeros) <= 1); % Obtaining the good zeros
B_z_minus = G_zeros(abs(G_zeros) > 1); % Obtaining the bad zeros

%%% Creating Q(z):

% Creating B+(z)'s z-polynomial:
syms z
B_z_plus = expand(prod(z - B_z_plus));
B_z_plus = double(coeffs(B_z_plus));
B_z_plus = B_z_plus(end:-1:1);
B_z_plus = tf(B_z_plus, [1], T); % B+(z)

% Creating B-(z)'s z-polynomial, B-(z^-1)'s z-polynomial, and B-(1):
B_z_minus = expand(prod(z - B_z_minus));
```

```
B_z1_minus = double(coeffs(B_z_minus));
B_z_minus = B_z1_minus(end:-1:1);
B_z_minus = tf(B_z_minus, [1], T); % B-(z)
B_z1_minus = tf(B_z1_minus, [1 0], T); % B-(z^-1)
B_minus1 = evalfr(B_z_minus, 1); % B-(-1)

Q_z = (B_z1_minus * D_z_tf)/((B_minus1.^2) * B_z_plus * gain2); % Formula for
Q(z) from slides
Q_z = tf([1], [1 0 0 0 0], T) * Q_z; % Adding poles at z = 0 until the
transfer function's numerator degree equals its denominator degree

% Formulating time and desired trajectory vectors:
t_sim1 = [0:T:0.5];
t_sim2 = [(0.5+T):T:1];
t_sim3 = [(1+T):T:1.5];
t_sim4 = [(1.5+T):T:2];

y1_t = 4.*(t_sim1.^2).*(3 - (4.*t_sim1));
y2_t = ones(1, length(t_sim2));
y3_t = y2_t - y1_t(1:(end-1));
y4_t = zeros(1, length(t_sim4));

t_tr1 = [t_sim1 t_sim2 t_sim3 t_sim4]; % Time vector for 1 cycle
y_tr_t1 = [y1_t, y2_t, y3_t, y4_t]; % Desired trajectory vector for 1 cycle

n_des = 5; % Number of desired cycles
for i = 0:(n_des-1)
    if(i == 0)
        t_tr = t_tr1;
        y_tr = y_tr_t1;
    else
        t_tr_cur = t_tr1 + (i.*2);
        t_tr = [t_tr, t_tr_cur(2:end)];
        y_tr = [y_tr, y_tr_t1(2:end)];
    end
end

% Performing linear simulations:
[r_t, ~, ~] = lsim(Q_z, y_tr, t_tr);
[y_sim_zept, ~, ~] = lsim(G_cltf_poly, r_t, t_tr);
[y_sim_zept_m, ~, ~] = lsim(G_cltf_poly_m, r_t, t_tr);

%% Part 6 - Addition of Stable Repetitive Controller

% Learning parameters for the controller:
Kl = 0.8;
N = 200;

% Creating the (z^N - 1) term:
N_poly = [1, zeros(1, (N-1)), -1];
N_poly = tf([1], N_poly, T); % (z^N - 1)

C_z = Kl * Q_z * N_poly; % Stable repetitive controller

% Systems' OLTFs:
```

```
G_oltf_st = C_z * G_cltf_poly; % Actual system
G_oltf_st_m = C_z * G_cltf_poly_m; % Varied-mass system

% Systems' CLTFs:
G_cltf_st = feedback(G_oltf_st, 1); % Actual system
G_cltf_st_m = feedback(G_oltf_st_m, 1); % Varied-mass system

% Performing linear simulations:
[y_sim_st, ~, ~] = lsim(G_cltf_st, y_tr, t_tr);
[y_sim_st_m, ~, ~] = lsim(G_cltf_st_m, y_tr, t_tr);

%% Part 7 - Plotting Results from 5 and 6

% Plotting Desired Trajectory:
figure
stairs(t_tr, y_tr, 'm', 'LineWidth', 1);
xlabel('Time [s]')
ylabel('y_d')
title('Desired Trajectory')
set(gca, 'FontSize', 15)
grid on

% Plotting ZEPT Controller Results:
figure
stairs(t_tr, y_sim_zept, 'b', 'LineWidth', 1)
%stairs(t_tr, y_sim_zept_m, 'c', 'LineWidth', 1)
xlabel('Time [s]')
ylabel('Output')
title('ZPET Controller Output vs. Time')
legend(lgnd_lbl)
set(gca, 'FontSize', 15)
grid on

% Plotting Stable Repetitive Tracking Results w/ Desired Trajectory:
figure
stairs(t_tr, y_tr, 'r', 'LineWidth', 1);
hold on
stairs(t_tr, y_sim_st, 'b', 'LineWidth', 1)
%stairs(t_tr, y_sim_st, 'c', 'LineWidth', 1)
xlabel('Time [s]')
ylabel('Output')
title('Stable Repetitive Controller Output vs. Time')
legend('Desired Trajectory', lgnd_lbl)
set(gca, 'FontSize', 15)
grid on
```