ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

CE3: DIGITAL CONTROL

ADVANCED CONTROL SYSTEMS ME-524

PROF. ALIREZA KARIMI

GROUP B

Angelo Giovine 368440

Baptiste Bühler 326168

14th August 2024

Contents

CHAPTER 1

RST CONTROLLER

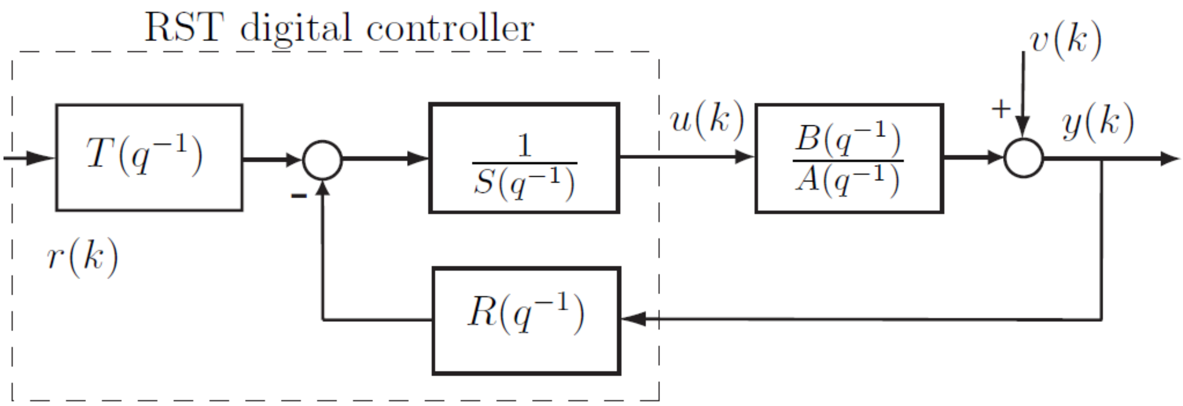

FIGURE 1.1
Control loop with RST digital controller

1.1 $A(q^{-1})$ AND $B(q^{-1})$

To implement an RST controller is first necessary to extract the numerator and denominator of our Plant using the command `tfdata`. Where our plant is:

$$G(q^{-1}) = \frac{B(q^{-1})}{A(q^{-1})} \tag{1.1}$$

1.2 POLE PLACEMENT FUNCTION

To perform pole placement we will write a function `poleplace`, that takes as inputs the following polynomials (expressed as vectors of coefficients):

- Plant's numerator: $B(q^{-1})$
- Plant's denominator: $A(q^{-1})$
- Fixed term of $R(q^{-1})$: $H_R(q^{-1})$
- Fixed term of $S(q^{-1})$: $H_S(q^{-1})$

- The desired poles: $P(q^{-1})$

And gives as an output the following polynomials:

- R of RST: $R(q^{-1})$
- S of RST: $S(q^{-1})$

After closing the loop with the RST controller we obtain the following closed loop equations:

If we assume $v(k) = 0$, the transfer function between $r(k)$ and $y(k)$ is:

$$F_{r\to y}(q^{-1}) = \frac{T(q^{-1})B(q^{-1})}{A(q^{-1})S(q^{-1}) + B(q^{-1})R(q^{-1})} \tag{1.2}$$

If we assume $r(k) = 0$, the transfer function between $v(k)$ and $y(k)$ is:

$$F_{v\to y}(q^{-1}) = \frac{A(q^{-1})S(q^{-1})}{A(q^{-1})S(q^{-1}) + B(q^{-1})R(q^{-1})} \tag{1.3}$$

Hence when we perform pole placement we want to solve:

$$A(q^{-1})S(q^{-1}) + B(q^{-1})R(q^{-1}) = P(q^{-1}) \tag{1.4}$$

A general solution is given by

$$x = M^{-1}p \tag{1.5}$$

Where:

$$x^T = \begin{bmatrix} 1 & s_1 & \cdots & s_{n_S} & r_0 & \cdots & r_{n_R} \end{bmatrix} \tag{1.6}$$

and M is a squared matrix of size $n_a + n_b$:

$$M = \begin{bmatrix} 1 & 0 & \cdots & 0 & b_0 & 0 & \cdots & 0 \\ a_1 & 1 & \cdots & 0 & b_1 & b_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n_A} & \cdots & \cdots & 1 & b_{n_B} & \cdots & \cdots & b_0 \\ 0 & a_{n_A} & \cdots & a_1 & 0 & b_{n_B} & \cdots & b_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n_A} & 0 & \cdots & 0 & b_{n_B} \end{bmatrix} \tag{1.7}$$

p is defined as:

$$p^T = \begin{bmatrix} 1 & p_1 & \cdots & p_{n_P} & 0 \cdots & 0 \end{bmatrix} \tag{1.8}$$

In our case, we are interested in solving pole placement with fixed terms H_R and H_S, so:

$$R(q^{-1}) = H_R(q^{-1})R'(q^{-1}), \quad S(q^{-1}) = H_S(q^{-1})S'(q^{-1}) \tag{1.9}$$

Therefore, we need to solve the following equation:

$$A(q^{-1})H_S(q^{-1})S'(q^{-1}) + B(q^{-1})H_R(q^{-1})R'(q^{-1}) = P(q^{-1}) \tag{1.10}$$

This can be done after replacing $A(q^{-1})H_S(q^{-1})$ by $A'(q^{-1})$ and $B(q^{-1})H_R(q^{-1})$ by $B'(q^{-1})$.

Here is the MATLAB function:

LISTING 1.1
poleplace

```
function [R, S] = poleplace(B, A, Hr, Hs, P)
    % Perform Pole placement
    % INPUTS:
    % A = [1 a1 a2...]
    % B = [b0 b1 b2...]
    % Hr = [hr0 hr1 ...]
    % Hs = [hs0 hs1 ...]
    % P = [1 p1 p2 ...]
    % OUTPUTS:
    % R = [r0 r1 r2...]
    % S = [1 s1 s2 ...]

    na = length(A) - 1;
    nb = length(B) - 1;
    nHs = length(Hs) - 1;
    nHr = length(Hr) - 1;
    np = length(P) - 1;

    if np > na + nHs + nb + nHr - 1
        error('Dimensions do not match!')
    end

    if size(A,2) ~= 1
        A = A';
    end

    if size(B,2) ~= 1
        B = B';
    end
    if size(Hs,2) ~= 1
        Hs = Hs';
    end
    if size(Hr,2) ~= 1
        Hr = Hr';
    end
    if size(P,2) ~= 1
        P = P';
    end

    na_prime = na + nHs;
```

```
    nb_prime = nb + nHr;

    A_prime = conv(A,Hs);
    B_prime = conv(B,Hr);

    for i = 1:nb_prime
        M1(:,i) = [zeros(i-1,1) ; A_prime ; zeros(nb_prime-i,1)];
    end
    for j = 1:na_prime
        M2(:,j) = [zeros(j-1,1) ; B_prime ; zeros(na_prime-j,1)];
    end

    M = [M1 M2];

    if np < na_prime+nb_prime
        P = [P ; zeros(na_prime+nb_prime-np-1,1)];
    end
    x = inv(M)*P;

    nr_prime = na_prime-1;
    ns_prime = nb_prime -1;

    S_prime = x(1:ns_prime+1);
    R_prime = x(ns_prime+2:end);

    S = conv(Hs,S_prime);
    R = conv(Hr,R_prime);
end
```

1.3 Choice of desired closed-loop poles

Following the requirements we want our poles in $0.99, 0.95, 0.95$. So $0.99, 0.95, 0.95$ should be the roots of $P(q^{-1})$. Hence:

$$P(q^{-1}) = 1 - 2.89q^{-1} + 2.7835q^{-2} - 0.893475q^{-3} \tag{1.11}$$

1.4 $R(q^{-1})$ and $S(q^{-1})$

Now we can simply run our function:

Listing 1.2
Calling pole place

```
[R, S] = poleplace(B,A,Hr,Hs,P);
```

1.5 Pole placement check

To check the pole placement function we can check Equation 1.4, so:

LISTING 1.3
Checking pole place

```
P_check = conv(A,S)+conv(B,R);
```

Since $P = P_{check}$, we have validated our function.

1.6 $T(q^{-1})$

Assuming the same dynamics for tracking and regulation and since we have an integrator in our controller, we can use:

$$T(q^{-1}) = R(1) = P(1)/B(1) = 0.0091$$

LISTING 1.4
T computation

```
T = sum(R)
```

1.7 TRACKING STEP RESPONSE OF THE CLOSED-LOOP SYSTEM

The tracking step response of the closed-loop system can be computed and observed in Fig. 1.2

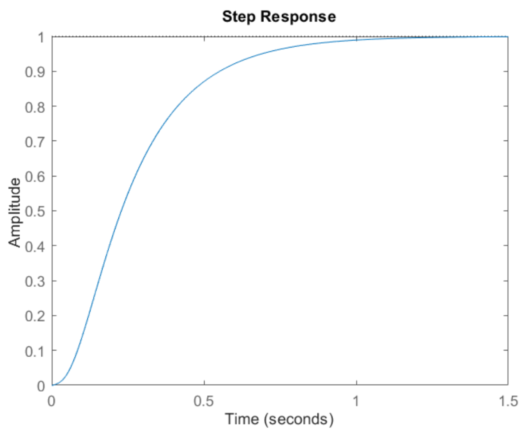

FIGURE 1.2
Tracking step response of the closed-loop system

Using `stepinfo` we can see that the settling time is 0.872s which is quite fast. Furthermore the zero steady-state tracking error for a step reference is met.

```
CL = tf(conv(T,B),P,Ts,'variable','z^-1');
step(CL)
stepinfo(CL)
```

1.8 VERIFICATION OF THE DESIGN

In Fig.1.3 we can observe that, first, the control signal has clearly a too high amplitude, overpassing the limit of ± 10V. Then, the plot of the bode of the sensitivity function $\mathcal{S} = \frac{AS}{P}$ shows that the modulus margin is not at least 0.5 as wanted.

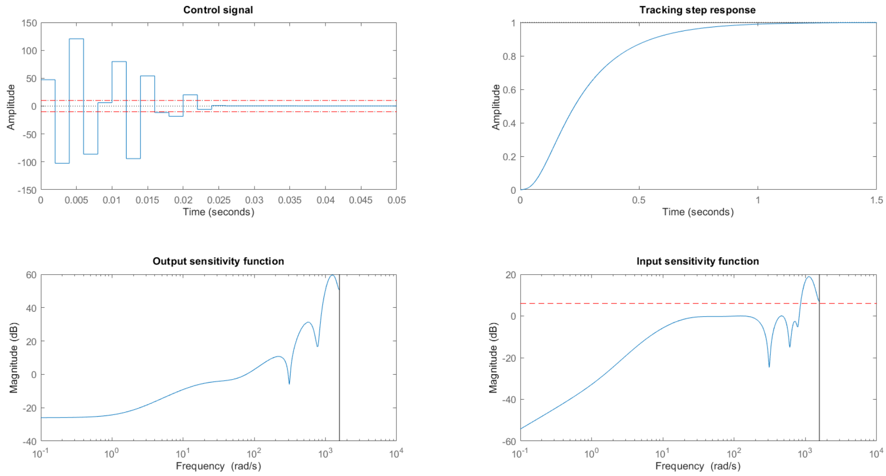

FIGURE 1.3
Performance check

1.9 INTRODUCTION OF THE FIXED TERM $H_r(q^{-1})$

Since the performance is not validated, we can use a fixed term $H_R(q^{-1}) = 1 + q^{-1}$ to open the loop at Nyquist frequency. This will reduce the magnitude of $\mathcal{U}$ as we can observe in Fig.1.4 but it is not sufficient to meet the performance.

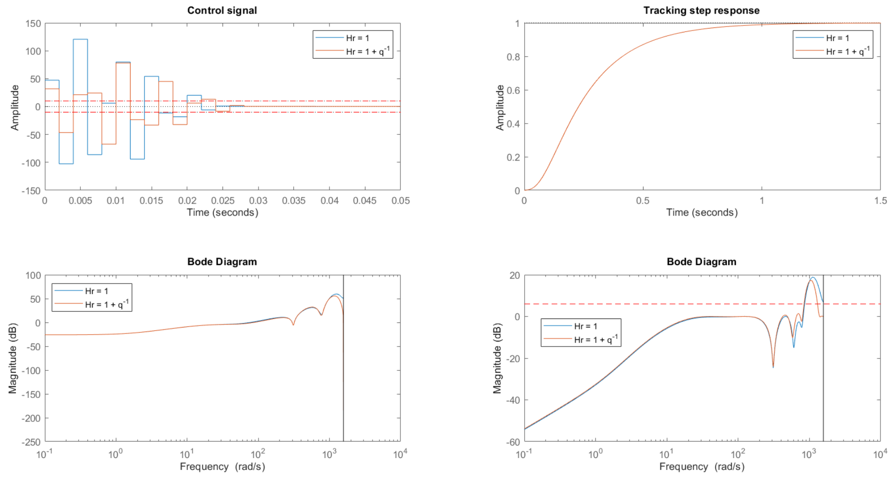

FIGURE 1.4
Performance check after the introduction of the fixed term H_R

1.10 Q-PARAMETRIZATION

To meet the performance we try to use a different approach : Q-Parametrization. This approach use a parameter, Q, used to defined R and S:

$$R(q^{-1}) = R_0(q^{-1}) + A(q^{-1})Q(q^{-1}) \tag{1.12}$$
$$S(q^{-1}) = S_0(q^{-1}) - B(q^{-1})Q(q^{-1}) \tag{1.13}$$

where $R_0(q^{-1})$ and $S_0(q^{-1})$ are computed for a nominal model and a given $P(q^{-1})$. Q is defined as:

$$Q(q^{-1}) = q_0 + q_1 q^{-1} + \cdots + q_{n_q} q^{-n_q}$$

By computing the closed-loop poles for the parametrized controller, we can see:

$$\begin{aligned} A(q^{-1})S(q^{-1}) + B(q^{-1})R(q^{-1}) =& A(q^{-1})S_0(q^{-1}) - A(q^{-1})B(q^{-1})Q(q^{-1}) \\ &+ B(q^{-1})R_0(q^{-1}) + B(q^{-1})A(q^{-1})Q(q^{-1}) \\ =& A(q^{-1})S_0(q^{-1}) + B(q^{-1})R_0(q^{-1}) \\ =& P(q^{-1}) \end{aligned} \tag{1.14}$$

Thus, we want to find Q by minimizing the ∞-norm of $\mathcal{U}$. This leads to this optimization problem:

$$\min_{Q} \|\mathcal{U}(Q)\|_\infty$$
$$\text{subject to:} \|W_1 \mathcal{S}\|_\infty \leq 1$$

We need to define two hyper-parameters: the order of Q, n_q and the initial value of Q, Q_0. We choose $n_q = 3$ and Q_0 is a random vector of length n_q. Furthermore, we choose to use $W_1 = 0.5$.

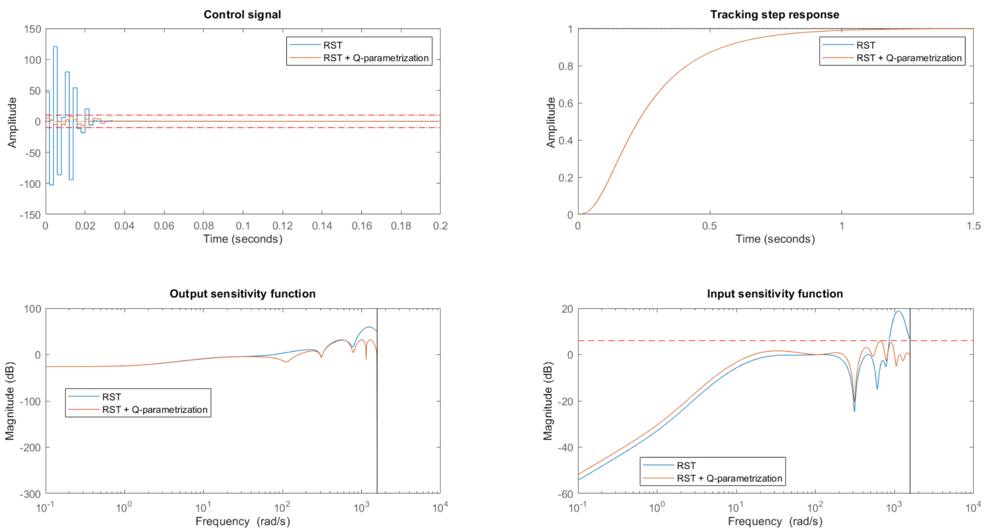

FIGURE 1.5
Performance check for Q-parametrization

The performance check of the given solution is shown in Fig.1.5. We can see that the performance is met, the control signal does not exceed the interval ± 10V and the modulus margin is at least 0.5.

The MATLAB code we used to perform our Q-parametrization:

LISTING 1.5
Q-parametrization

```
nq = 3;
rng(0)
Q0 = randn(1,nq);
W1 = 0.5;

Rq = @(Q) sumpol(R',conv(conv(A,Hr),conv(Hs,Q))); % R = R0+A*Hr*Hs*Q
Sq = @(Q) sumpol(S',-conv(conv(B,Hs),conv(Hr,Q))); % S = S0-B*Hs*Hr*Q
Uq = @(Q) tf(conv(A,Rq(Q)),P,Ts,'variable','z^-1'); % U = A*R/P
Tq = @(Q) sum(Rq(Q));
fun = @(Q) norm(Uq(Q),inf);

sensq = @(Q) tf(conv(A,Sq(Q)),P,Ts,'variable','z^-1'); % Ss = A*S/P
tauq = @(Q) 1-sensq(Q);
ineq = @(Q) [norm(0.5*sensq(Q),inf)-1];
eq = [];

const = @(Q) deal(ineq(Q), eq);
opts = optimoptions('fmincon',...
'Algorithm','interior-point','Display','iter',...
'MaxFunctionEvaluations',5e+05);
[Qopt,~,exitflag,~,~,~,~] =
    fmincon(fun,Q0,[],[],[],[],[],[],const,opts)

Rnew = sumpol(R',conv(conv(A,Hr),conv(Hs,Qopt)));
Snew = sumpol(S',-conv(conv(B,Hs),conv(Hr,Qopt)));
Pnew = sumpol(conv(A,Snew),conv(B,Rnew));
Tnew = sum(Rnew);
```

Chapter 2

Real-time Implementation

2.1 Data driven implementation

Now we want to implement our previously designed data-driven controller. To do that is necessary to convert it into an RST controller.

The control law of a general RST controller is:

$$S(q^{-1})u(k) = T(q^{-1})r(k) - R(q^{-1})y(k) \tag{2.1}$$

The control law of our data-driven controller is:

$$Y(q^{-1})u(k) = X(q^{-1})y(k) - X(q^{-1})r(k) \tag{2.2}$$

where:

$$K(q^{-1}) = \frac{X(q^{-1})}{Y(q^{-1})} \tag{2.3}$$

Hence it's possible to observe that:

$$R(q^{-1}) = X(q^{-1}) \tag{2.4}$$

$$S(q^{-1}) = Y(q^{-1}) \tag{2.5}$$

$$T(q^{-1}) = X(q^{-1}) \tag{2.6}$$

Now that we have converted our controller is possible to test it on real systems, by positioning the weights to different locations on the flexible component. We have conducted tests on wights at 4,8 and 14 cm from the center.

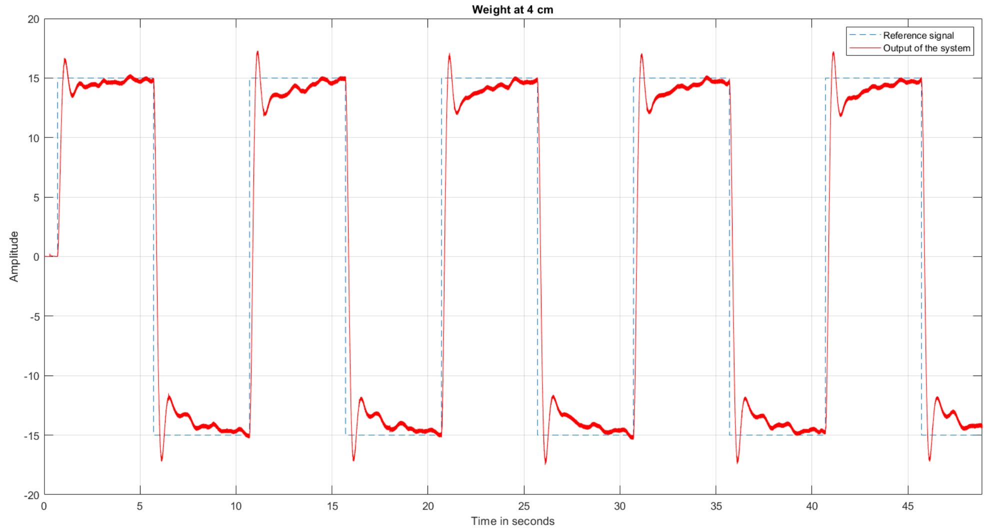

FIGURE 2.1
System response with weights at 4 cm

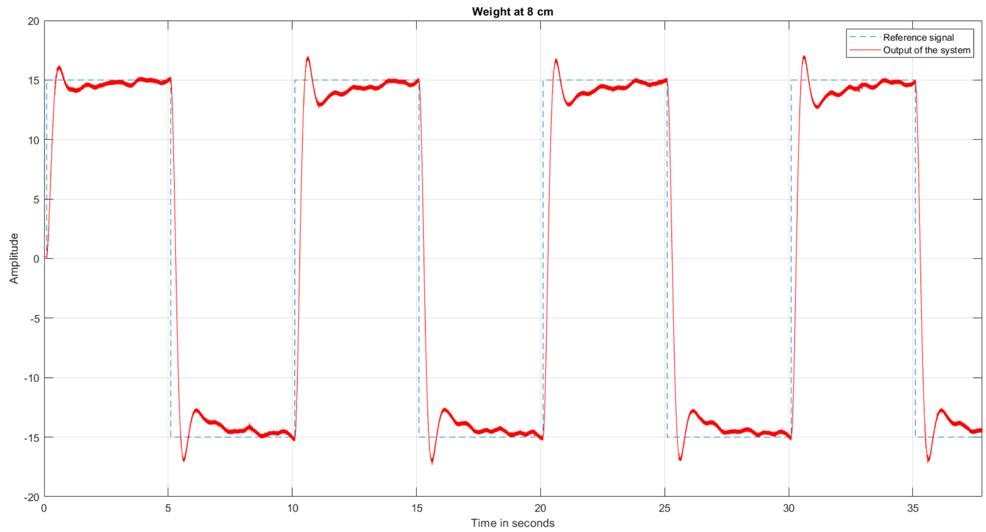

FIGURE 2.2
System response with weights at 8 cm

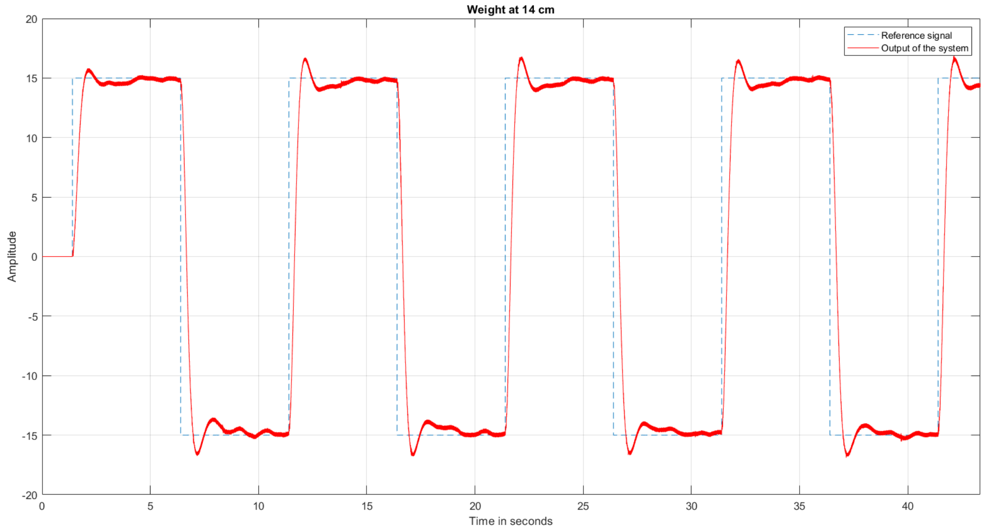

FIGURE 2.3
System response with weights at 14 cm

The effectiveness of this data-driven controller is highlighted by its ability to meet the criteria for robust stability, as demonstrated by the system's consistent stability across all tested loads. Regarding performance, although overshoots and oscillations are present, the setpoint is consistently achieved within the duration of the step. Furthermore, the greater the separation between the weights, the more effectively our controller performs. Finally, an important advantage of this method is its utility in controller synthesis. Once measurements are collected, one can utilize all available models to define the objectives and constraints for the optimization problem, so there is no need to compute a weighting filter W_2, the only attention should be paid to the choice of the initial controller that should be stabilizing.

2.2 RST IMPLEMENTATION

Now we want to implement our RST controller as designed in chapter 1. As for the Data driven implementation we positioned the weights to different locations on the flexible component. We have conducted tests on weights at 4, 8 and 14cm from the center.

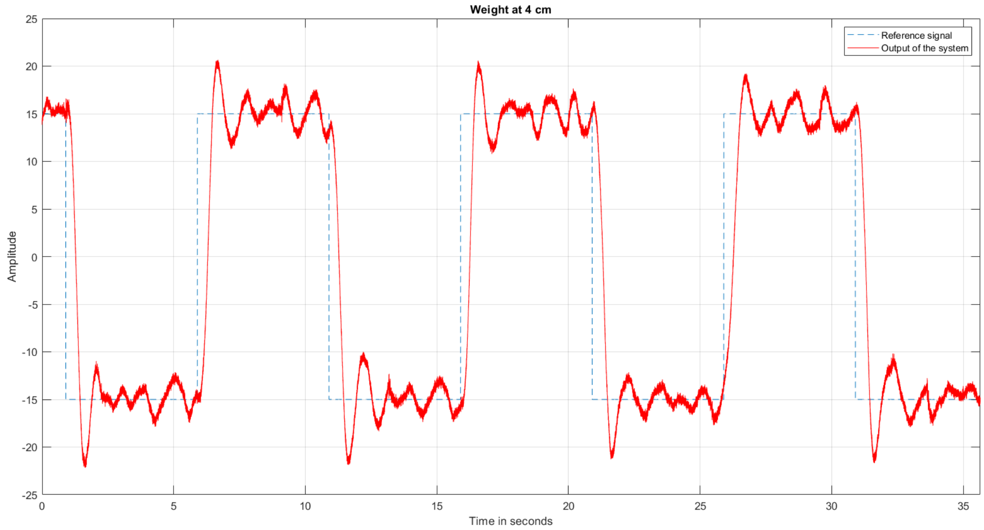

FIGURE 2.4
System response with weights at 4 cm

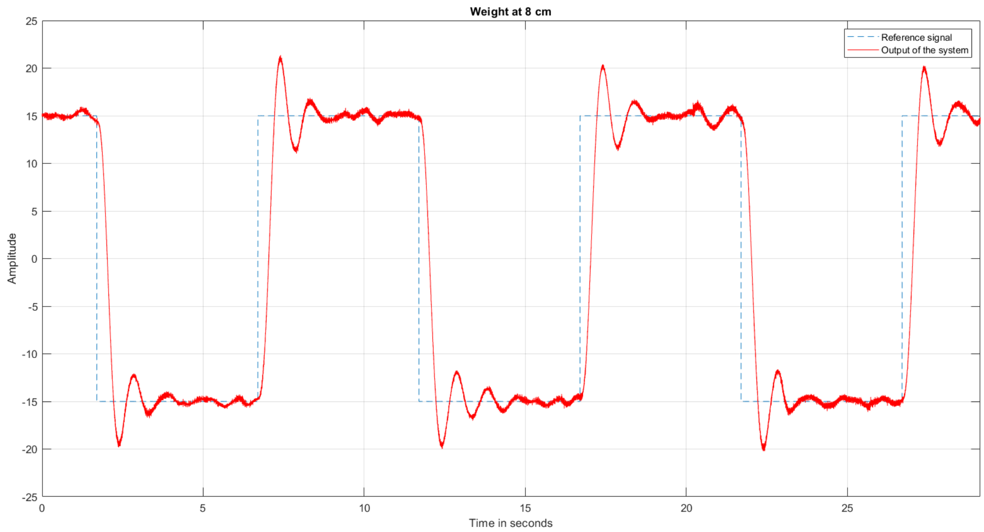

FIGURE 2.5
System response with weights at 8 cm

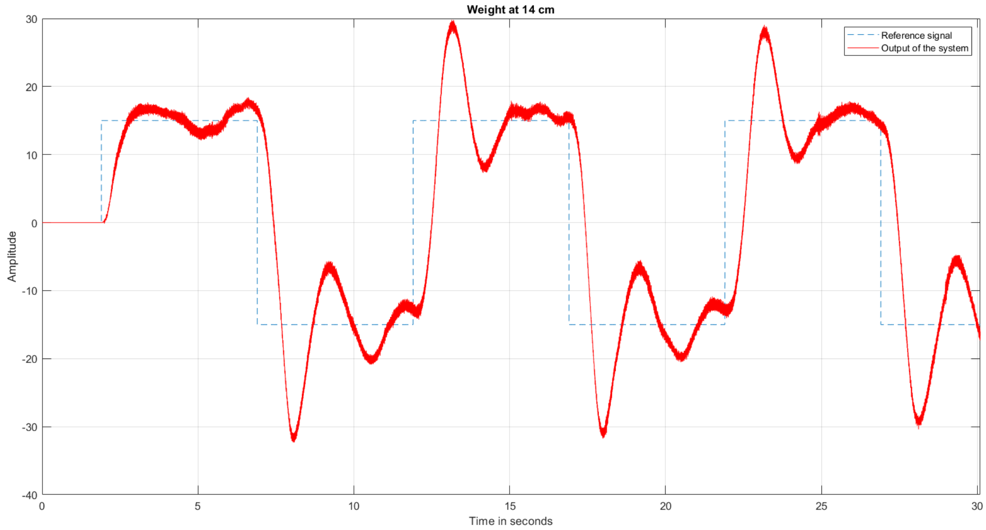

FIGURE 2.6
System response with weights at 14 cm

We can observe that first the controller is working since the stability condition is met. But the settling time is not optimal and the damping coefficient of the closed-loop transfer function is too low. To improve this controller it is possible to increase the damping coefficient, this leads to a smoother Bode diagram of the sensitivity function $\mathcal{S}$, close to a high-pass filter, and thus it is possible to add a constraint to the Q-parametrization formulation:

$$\begin{aligned} &\min_{Q} \|\mathcal{U}(Q)\|_\infty \\ \text{subject to:} &\|W_1\mathcal{S}\|_\infty \leq 1 \\ &\|W_2\mathcal{T}\|_\infty \leq 1 \end{aligned}$$

The resulting controller should be then more robust than the actual one since for the current controller we obtain:

$$\|W_2\mathcal{T}\|_\infty = 11.1752 \geq 1$$

APPENDIX A

MATLAB CODE

```
%% ME524 - Computer exercise 3
clc, clear, close all

load Gnom
Gnom = G11;
[Gu74, info74] = ucover(Gmm, Gnom, 7);
W2 = info74.W1;

% 1.
[B, A] = tfdata(Gnom,'v');

% 2. poleplace function

% 3.
% P = poly([0.8 0.9 0.95]);
P = poly([0.99 0.95 0.95]);

% Fixed parts
Hs = [1 -1];
Hr = [1];

% 4.
[R, S] = poleplace(B,A,Hr,Hs,P);

% 5.
Pcheck = conv(A,S)+conv(B,R);

% 6.
T1 = sum(P)/sum(B);
T2 = sum(R);
% T = T1 = T2 = P(1)/B(1) = R(1) since we have an integrator in the
% controller

```

```
T = [T1];

% 7. + 8. Plots
figure(1)
subplot(2,2,1)
U1 = tf(conv(A,R)',P,Ts,'variable','z^-1');
step(U1,tf(10),'--r',tf(-10),'--r')
title('Control signal')

subplot(2,2,2)
CL1 = tf(conv(T,B),P,Ts,'variable','z^-1');
step(CL1)
title('Tracking step response')
% stepinfo(CL1)

subplot(2,2,3)
bodemag(U1)
title('Output sensitivity function')

subplot(2,2,4)
Ss1 = tf(conv(A,S)',P,Ts,'variable','z^-1');
bodemag(Ss1,tf(1,0.5),'--r')
title('Input sensitivity function')

% 9.
Hr = [1 1];
[R, S] = poleplace(B,A,Hr,Hs,P);

Pcheck = conv(A,S)+conv(B,R); % OK
T = sum(R);

% Plots
figure(2)
subplot(2,2,1)
step(U1)
hold on
U = tf(conv(A,R)',P,Ts,'variable','z^-1');
step(U,tf(10),'--r',tf(-10),'--r')
legend('Hr = 1','Hr = 1 + q^{-1}')
title('Control signal')

subplot(2,2,2)
step(CL1)
hold on
CL = tf(conv(T,B),P,Ts,'variable','z^-1');
step(CL)
legend('Hr = 1','Hr = 1 + q^{-1}')
title('Tracking step response')
% stepinfo(CL)
```

```
subplot(2,2,3)
bodemag(U1)
hold on
bodemag(U)
legend('Hr = 1','Hr = 1 + q^{-1}')
title('Output sensitivity function')

subplot(2,2,4)
bodemag(Ss1)
hold on
Ss = tf(conv(A,S)',P,Ts,'variable','z^-1');
bodemag(Ss,tf(1,0.5),'--r')
legend('Hr = 1','Hr = 1 + q^{-1}')
title('Input sensitivity function')

% 9. Q parametrization
nq = 3;
rng(0)
Q0 = randn(1,nq);
W1 = 0.5;

Rq = @(Q) sumpol(R',conv(conv(A,Hr),conv(Hs,Q))); % R = R0+A*Hr*Hs*Q
Sq = @(Q) sumpol(S',-conv(conv(B,Hs),conv(Hr,Q))); % S = S0-B*Hs*Hr*Q
Uq = @(Q) tf(conv(A,Rq(Q)),P,Ts,'variable','z^-1'); % U = A*R/P
Tq = @(Q) sum(Rq(Q));
fun = @(Q) norm(Uq(Q),inf);

sensq = @(Q) tf(conv(A,Sq(Q)),P,Ts,'variable','z^-1'); % Ss = A*S/P
tauq = @(Q) 1-sensq(Q);
ineq = @(Q) [norm(0.5*sensq(Q),inf)-1]; % remove W2T constraint
eq = [];

const = @(Q) deal(ineq(Q), eq);
opts = optimoptions('fmincon','Algorithm','interior-point',...
'Display','iter','MaxFunctionEvaluations',5e+05);
[Qopt,~,exitflag,~,~,~,~] =
    fmincon(fun,Q0,[],[],[],[],[],[],const,opts)

Rnew = sumpol(R',conv(conv(A,Hr),conv(Hs,Qopt)));
Snew = sumpol(S',-conv(conv(B,Hs),conv(Hr,Qopt)));
Pnew = sumpol(conv(A,Snew),conv(B,Rnew));
Tnew = sum(Rnew);

% Plots
figure(3)
subplot(2,2,1)
step(U1)
hold on
```

```
U = tf(conv(A,Rnew),Pnew,Ts,'variable','z^-1');
step(U,tf(10),'--r',tf(-10),'--r')
legend('RST','RST + Q-parametrization')
title('Control signal')

subplot(2,2,2)
step(CL1)
hold on
CL = tf(conv(Tnew,B),Pnew,Ts,'variable','z^-1');
step(CL)
legend('RST','RST + Q-parametrization')
title('Tracking step response')
% stepinfo(CL)

subplot(2,2,3)
bodemag(U1)
hold on
bodemag(U)
legend('RST','RST + Q-parametrization')
title('Output sensitivity function')

subplot(2,2,4)
bodemag(Ss1)
hold on
Ss = tf(conv(A,Snew),Pnew,Ts,'variable','z^-1');
bodemag(Ss,tf(1,0.5),'--r')
legend('RST','RST + Q-parametrization')
title('Input sensitivity function')

function p = sumpol(p1,p2)
    % Function to add two polynomials of different orders
    % Inputs : p1,p2 two polynomials
    % Output : p = p1+p2
    n1 = length(p1);
    n2 = length(p2);

    if n1 ~= n2
        if n1 < n2
            p1 = [p1 zeros(1,n2-n1)];
        else
            p2 = [p2 zeros(1,n1-n2)];
        end
    end
    p = p1 + p2;
end
```