State estimation for wave energy converters

Giorgio Bacelli and Ryan G. Coe

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-mission laboratory
managed and operated by National Technology and Engineering Solutions of Sandia, LLC.,
a wholly owned subsidiary of Honeywell International, Inc.,
for the U.S. Department of Energy's National Nuclear Security Administration
under contract DE-NA0003525.

Approved for public release; further dissemination unlimited.
State estimation for wave energy converters

Giorgio Bacelli
Water Power Technologies Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-9999
gbacell@sandia.gov

Ryan G. Coe
Water Power Technologies Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-9999
rcoe@sandia.gov
1 Introduction

This report gives a brief discussion and examples on the topic of state estimation for wave energy converters (WECs). These methods are intended for use to enable real-time closed loop control of WECs. The algorithm for the optimal estimation of unknown inputs and state requires the system to be expressed using a discrete-time state-space model as [1]

\begin{align*}
    x_{k+1} &= Ax_k + Bu_k + Gd_k + w_k \\
    y_k &= Cx_k + Du_k + Hd_k + v_k,
\end{align*}

(1a) (1b)

where $d_k$ is the unknown input at time step $k$ (i.e. the excitation force $d_k = F_e$ or the excitation pressure $d_k = P_e$), $u_k$ is the known input (i.e. the actuator’s force $u_k = F_a$), $y_k$ is the measurements vector (i.e. acceleration and pressure $y = [\ddot{z}, P]^T$), $w_k$ and $v_k$ are the process and measurements noise, respectively. Two examples are discussed here: using position and acceleration measurements (Section 2) and using pressure and acceleration measurements (Section 3).
2 Using position

This Section provides an example for using position and acceleration to predict the state of a WEC. Sample code for implementing this example is provided in Appendix A. The block diagram of the buoy’s dynamic model is depicted in Fig. 1. The “intrinsic” model of the buoy $G_i$ in continuous-time state space form is

$$
\dot{x}_i = A_i x_i + B_i (F_e + F_a) \\
y_i = C_i x_i + D_i (F_e + F_a)
$$

(2a)

(2b)

where the state vector is

$$x = \begin{bmatrix} z \\ \dot{z} \\ x_r \end{bmatrix},
$$

(3)

and

$$A_i = \begin{bmatrix}
\frac{B_f}{m+m_{\infty}} & -\frac{K}{m+m_{\infty}} & -\frac{C_r}{m+m_{\infty}} \\
1 & 0 & 0 \\
B_r & 0_{n_r \times 1} & A_r
\end{bmatrix}, \quad B_i = \begin{bmatrix}
1 & 0 \\
0_{n_r \times 1} & 0 \\
0_{n_r \times 1} & 0_{n_r \times 1}
\end{bmatrix}
$$

(4)

$$C_i = \begin{bmatrix}
0 & \frac{B_f}{m+m_{\infty}} & -\frac{K}{m+m_{\infty}} & -\frac{C_r}{m+m_{\infty}} \\
0_{1 \times n_r} & 1 & 0_{1 \times n_r}
\end{bmatrix}, \quad D_i = \begin{bmatrix}
0 \\
0 \\
1
\end{bmatrix}
$$

(5)

and where the matrices $A_r \in \mathbb{R}^{n_r \times n_r}, B_r \in \mathbb{R}^{n_r \times 1}$ and $C_r \in \mathbb{R}^{1 \times n_r}$ describe the radiation force $F_r$ dynamics as

$$
\dot{x}_r = A_r x_r + B_r \dot{z} \\
F_r = C_r x_r.
$$

(6a)

(6b)

The mass of the buoy is denoted by $m$, the hydrostatic restoring coefficient by $K$, the friction coefficient by $B_f$ and $m_{\infty}$ is the asymptotic value of the added mass for $\omega \to \infty$. Two steps are now required to formulate system in (2) as required in (1)

1. Convert to discrete time

2. Derive matrices $A, B, C, D, G$ and $H$

If the matrix $A_i$ is not singular, then step 1 can be carried out by using the

$$A = e^{A_i T_c}, \quad B = A_i^{-1} (A - I) B_i
$$

(7)

(8)

where $T_c$ is the sampling time. The matrices $C, D, G$ and $H$ are:

$$C = C_i, \quad D = D_i
$$

(9)

$$G = B, \quad H = D.
$$

(10)

The time-varying version of the algorithm given in [1] whereas the steady-state version (much faster computation) is given in [2]
Algorithm 1 Time-varying Unknown Input and State Estimator.

▷ Initialize:
1: \( \hat{x}_{0 | 0} = \mathbb{E}[x_0] \)
2: \( \hat{d}_0 = H^T(y_0 - C\hat{x}_{0 | 0} - Du_0) \)
3: \( P_{0 | 0}^x = \mathbb{P}^x_0 \)
4: \( P_{0 | 0}^d = \mathbb{P}^d_0 \)
5: \( P_{0 | 0}^{xd} = \mathbb{P}^{xd}_0 \)
6: \( Q = \mathbb{E}[ww^T] \)
7: \( R = \mathbb{E}[vv^T] \)

▷ Estimation loop for \( N \) time steps (Time step = \( T_c \))
8: for \( k = 1 \) to \( N \) do
  9: One-Step prediction
  10: \( \hat{x}_{k | k-1} = A \hat{x}_{k-1 | k-1} + Bu_{k-1} + G\hat{d}_{k-1} \)
  11: \( P_{k | k-1}^x = AP_{k-1 | k-1}^x A^T + GP_{k-1 | k-1}^{xd} A^T + AP_{k-1 | k-1}^{xd} G^T + GP_{k-1 | k-1} G^T + Q \)
  12: Measurements update
  13: \( L_k = K_k \left( I - H^T \tilde{R}_k^{-1} H \right)^{-1} H^T \tilde{R}_k^{-1} \)
  14: \( \hat{x}_{k | k} = \hat{x}_{k | k-1} + L_k \left( y_k - C\hat{x}_{k | k-1} - Du_k \right) \)
  15: \( P_{k | k}^x = (I - L_k C) P_{k | k-1}^x (I - L_k C)^T + L_k RL_k^T \)
  16: Estimation of unknown input
  17: \( \hat{R}_k^* = (I - CL_k) \hat{R}_k (I - CL_k)^T \)
  18: \( P_{k | k}^d = (H^T \hat{R}_k^{-1} H)^{-1} \)
  19: \( M_k = P_{k | k}^d H^T \tilde{R}_k^{-1} \)
  20: \( \hat{d}_k = M_k \left( y_k - C\hat{x}_{k | k} - Du_k \right) \)
  21: \( P_{k | k}^{xd} = -P_{k | k}^x CM_k^T + L_k RM_k^T \)
end for

Algorithm 2 Steady-State Unknown Input and State Estimator.

▷ Initialize:
1: \( \hat{x}_{0 | 0} = \mathbb{E}[x_0] \)
2: \( \hat{d}_0 = H^T(y_0 - C\hat{x}_{0 | 0} - Du_0) \)
3: \( L_\infty = \lim_{k \to \infty} L_k \)
4: \( M_\infty = \lim_{k \to \infty} M_k \)

▷ Estimation loop for \( N \) time steps (Time step = \( T_c \))
5: for \( k = 1 \) to \( N \) do
  6: One-Step prediction
  7: State estimation
  8: Unknown input estimation
  9: end for
Figure 1. Block diagram of the buoy’s dynamic model when position and acceleration are measured.
3 Using pressure

This Section provides an example for using pressure and acceleration to predict the state of a WEC. Sample code for implementing this example is provided in Appendix B. The block diagram of the buoy’s dynamic model is depicted in Fig. 1. The “intrinsic” model of the buoy $G_i$ in continuous-time state space form is

\[ \dot{x}_i = A_i x_i + B_i (F_e + F_a) \]
\[ y_i = C_i x_i + D_i (F_e + F_a) \]

where the output vector $y_i$ is

\[ y_i = \begin{bmatrix} \ddot{z} \\ P_r \end{bmatrix}, \]

and the matrices composing the state space model in (11) have been identified form experimental data.

The state-space model of the excitation pressure in continuous-time ($G_e$) is:

\[ \dot{x}_e = A_e x_e + B_e P_e \]
\[ F_e = C_e x_e + D_e P_e. \]

According to the diagram in Fig. 2 the models in (11) and (13) can be combined to form the state-state space model

\[ \dot{x} = A_c x + B_c \begin{bmatrix} F_a \\ P_e \end{bmatrix} \]
\[ y = C_c x + D_c \begin{bmatrix} F_a \\ P_e \end{bmatrix} \]

where the state vector is

\[ x = \begin{bmatrix} x_i \\ x_e \end{bmatrix} \]

and where the output vector is

\[ y = \begin{bmatrix} \dddot{z} \\ P \end{bmatrix} = \begin{bmatrix} \dddot{z} \\ P_r + P_e \end{bmatrix} = \begin{bmatrix} \ddot{z} \\ P_r \end{bmatrix} + \begin{bmatrix} 0 \\ P_e \end{bmatrix} = y_i + \begin{bmatrix} 0 \\ P_e \end{bmatrix}. \]

The system matrices are

\[ A_c = \begin{bmatrix} A_i & B_i C_e \\ 0 & A_e \end{bmatrix} \]
\[ B_c = \begin{bmatrix} B_i & B_i D_e \\ 0 & B_e \end{bmatrix} \]
\[ C_c = \begin{bmatrix} C_i & D_i C_e \end{bmatrix} \]
\[ D_c = \begin{bmatrix} D_i & D_i D_e + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} \]

Two steps are now required to formulate system in (14) as required in (1).
Figure 2. Block diagram of the buoy’s dynamic model when pressure and acceleration are measured.

1. Convert to discrete time

2. Derive matrices $A$, $B$, $C$, $D$, $G$ and $H$

If the matrix $A_c$ is not singular, then step 1 can be carried out by using the

$$A = e^{A_c T_c}$$

$$\tilde{B} = A_c^{-1} (A - I) B_c$$

where $T_c$ is the sampling time. $\tilde{B}$ and $D_c$ are $2 \times n$ matrices: $B$ is the first column of $\tilde{B}$, $G$ is the second column of $\tilde{B}$, $D$ is the first column of $D_c$ and $H$ is the second column of $D_c$, that is:

$$\tilde{B} = [B \quad G]$$

$$C = C_c$$

$$D_c = [D \quad H]$$

The time-varying version of the algorithm given in 3 whereas the steady-state version (much faster computation) is given in 4.
Algorithm 3 Time-varying Unknown Input and State Estimator.

\begin{algorithm}
\begin{algorithmic}[1]
\State Initialize:
\State \( \hat{x}_{0|0} = \mathbb{E}[x_0] \)
\State \( \hat{d}_0 = H^T (y_0 - C \hat{x}_{0|0} - D u_0) \)
\State \( P_{x0}^x = \mathbb{P}_0^x \)
\State \( P_{0}^d = \mathbb{P}_0^d \)
\State \( P_{0}^{xd} = \mathbb{P}_0^{xd} \)
\State \( Q = \mathbb{E}[w w^T] \)
\State \( R = \mathbb{E}[v v^T] \)
\State Estimation loop for \( N \) time steps (Time step = \( T_c \))
\For{k = 1 to \( N \)}
\State One-Step prediction
\State \( \hat{x}_{k|k-1} = A \hat{x}_{k-1|k-1} + B u_{k-1} + G \hat{d}_{k-1} \)
\State \( P_{x k|k-1} = A P_{x k-1|k-1} A^T + G P_{xd k-1} A^T + A P_{k-1}^{xd} G^T + G P_{k-1}^{d} G^T + Q \)
\State Measurements update
\State \( \tilde{R}_{k} = C P_{x k|k-1} C^T \)
\State \( L_k = K_k \left( I - H (H^T \tilde{R}_{k}^{-1} H)^{-1} H^T \tilde{R}_{k}^{-1} \right) \)
\State \( \hat{x}_{k|k} = \hat{x}_{k|k-1} + L_k (y_k - C \hat{x}_{k|k-1} - D u_k) \)
\State \( P_{x k|k} = (I - L_k C) P_{x k|k-1} (I - L_k C)^T + L_k R L_k^T \)
\State Estimation of unknown input
\State \( \hat{R}_{k}^* = (I - C L_k) \tilde{R}_{k} (I - C L_k)^T \)
\State \( P_{k}^d = (H^T \hat{R}_{k}^{-1} H)^{-1} \)
\State \( M_k = P_{k}^d H^T \hat{R}_{k}^{-1} \)
\State \( \hat{d}_{k} = M_k (y_k - C \hat{x}_{k|k} - D u_k) \)
\State \( P_{k}^{xd} = -P_{k|k}^{x} C^T M_k^T + L_k R M_k^T \)
\EndFor
\end{algorithmic}
\end{algorithm}

Algorithm 4 Steady-State Unknown Input and State Estimator.

\begin{algorithm}
\begin{algorithmic}[1]
\State Initialize:
\State \( \hat{x}_{0|0} = \mathbb{E}[x_0] \)
\State \( \hat{d}_0 = H^T (y_0 - C \hat{x}_{0|0} - D u_0) \)
\State \( L_{\infty} = \lim_{k \to \infty} L_k \)
\State \( M_{\infty} = \lim_{k \to \infty} M_k \)
\State Estimation loop for \( N \) time steps (Time step = \( T_c \))
\For{k = 1 to \( N \)}
\State One-Step prediction
\State \( \hat{x}_{k|k-1} = A \hat{x}_{k-1|k-1} + B u_{k-1} + G \hat{d}_{k-1} \)
\State State estimation
\State \( \hat{x}_{k|k} = \hat{x}_{k|k-1} + L_{\infty} (y_k - C \hat{x}_{k|k-1} - D u_k) \)
\State \( \hat{d}_{k} = M_{\infty} (y_k - C \hat{x}_{k|k} - D u_k) \)
\State Unknown input estimation
\EndFor
\end{algorithmic}
\end{algorithm}
References

A Sample code: position and acceleration measurements

This section contains sample MATLAB code for implementing a position/acceleration state estimator for a WEC.

```matlab
% this script is to test the Unified Linear Input and State Estimator
% (ULISE algorithm) described in
% S. Z. Yong, M. Zhu, and E. Frazzoli, A unified filter for simultaneous
% input and state estimation of linear discrete-time stochastic systems,
% Both Time-Varying and Steady-State versions are implemented.
% Both Position and acceleration measurements are used to estimate state and
% excitation force
% Sandia National Laboratories is a multi-mission laboratory managed and
% operated by National Technology and Engineering Solutions of Sandia,
% LLC., a wholly owned subsidiary of Honeywell International, Inc., for the
% U.S. Department of Energy’s National Nuclear Security Administration
% under contract DE-NA0003525.
% G. Bacelli, R. Coe
% Sandia National Laboratories
% 2017

clear

% load identified parametric WEC model model
WEC = load('WEC_param_model_1DOF.mat');

Tc = 1e-3; % sampling time
N = 5e4; % number of simulation steps

% load excitation force and interpolate
Fe = load('Exc_time_series.mat');
t = (1:N)*Tc;
d = interp1(Fe.t_trim, Fe.Fexc_td, t, 'pchip')*1e-3; % excitation force in kN
u = 0.5*sin(2*pi*0.75*t); % control input (PTO force in kN). Open loop, no control implemented

mass = 858.3987;
Ainf_hat = 822.3799;
K = 2.3981e+04;
Ar = WEC.rad_sys.a;
Br = WEC.rad_sys.b;
Cr = WEC.rad_sys.c;
Bf = WEC.B_eq_mat(9);

Ac = [-Bf/(mass + Ainf_hat) -K/(mass + Ainf_hat) -Cr/(mass + Ainf_hat)];
Br zeros(length(Ar),1) Ar ];

% measurements: position and acceleration
```
\( C = \begin{bmatrix} 0 & 1 \\ \text{zeros}(1, \text{length}(A)) \end{bmatrix} \); \\
A = C(1, :); \\
D = \begin{bmatrix} 0 \\ Bc(1) \end{bmatrix}; \\
n = \text{size}(C, 2); \\
p = \text{size}(C, 1); \\
\% \text{convert continuous time model to discrete time} \\
A = \expm(Ac \times Tc); \\
B = Ac \backslash \begin{bmatrix} A - \text{eye}(n) \end{bmatrix} \times Bc; \\
G = B; \\
H = D; \\
\% \text{process noise} \\
w = 0.00001 \times \begin{bmatrix} \text{rand}(N, 4) - 0.5 \end{bmatrix}; \\
\% \text{measurements noise} \\
v = \begin{bmatrix} \text{rand}(N, p) - 0.5 \end{bmatrix} \times \text{diag} \begin{bmatrix} 0.0001, 0.05 \end{bmatrix}; \\
Q = \text{cov}(w); \\
R = \text{cov}(v); \\
\% \text{Time varying filter} \\
\% \text{initialize variables} \\
P_x_k_k = 0.001 \times \text{eye}(n); \\
P_xd_k = 0.001 \times \text{ones}(n, 1); \\
P_d_k = 0.001; \\
x = \text{zeros}(n, 1); \\
x_k_k = x; \\
d_k = 0; \\
In = \text{eye}(n); \\
Ip = \text{eye}(p); \\
\% \text{preallocation} \\
d_k_vec = \text{zeros}(N, 1); \\
x_k_vec = \text{zeros}(n, N); \\
x_vec = \text{zeros}(n, N); \\
y_vec = \text{zeros}(p, N); \\
tic \\
for ii = 1:N \\
\text{x = } \begin{bmatrix} A \times x + B \times u(ii) + G \times d(ii) + w(ii, :) \end{bmatrix}; \\
\text{y = } \begin{bmatrix} C \times x + D \times u(ii) + v(ii, :) \end{bmatrix}; \\
x_vec(:, ii) = x; \\
y_vec(:, ii) = y; \\
x_k_k1 = \begin{bmatrix} A \times x_k_k + B \times u(ii) + G \times d(ii) \end{bmatrix}; \\
P_x_k_k1 = \begin{bmatrix} A \times P_x_k_k \times A' + G \times P_xd_k \times A' + A \times P_xd_k \times G' + G \times P_d_k \times G' + 0 \end{bmatrix}; \\
R_t_k = \begin{bmatrix} C \times P_x_k_k1 \times C' + R \end{bmatrix}; \\
Kk = \begin{bmatrix} P_x_k_k1 \times C' \times R_t_k - Kk \end{bmatrix}; \\
Lk = \begin{bmatrix} (C' \times (C' \times R_t_k - Kk) \times R_t_k - Kk) \end{bmatrix}; \\
x_k_k1 = \begin{bmatrix} x_k_k1 + Lk \times (y - C \times x_k_k1 - D \times u(ii)) \end{bmatrix}; \\
P_x_k_k1 = \begin{bmatrix} (C \times x_k_k1 \times C' \times Lk + Lk \times R \times Lk') \end{bmatrix}; \\
P_d_k = \begin{bmatrix} \text{inv}(H' \times R_t_k - Kk) \end{bmatrix}; \\
Mk = \begin{bmatrix} ((H' \times R_t_k - Kk) \times H') \times R_t_k, 1 \end{bmatrix}; \\
R_t_k = \begin{bmatrix} (C \times x_k_k1 \times C' \times Mk + Lk \times R \times Mk') \end{bmatrix}; \\
d_k = \begin{bmatrix} Mk \times (y - C \times x_k_k1 - D \times u(ii)) \end{bmatrix}; \\
P_xd_k = \begin{bmatrix} -P_x_k_k1 \times C' \times Mk' + Lk \times R \times Mk' \end{bmatrix}; \\
d_k_vec(ii) = d_k; \)
x_k_vec(:,ii) = x_k_k;
end

t_tv = toc;
disp('done TV')

% steady state filter
L_inf = Lk;
M_inf = Mk;

% preallocation
d_k_vec_inf = zeros(N,1);
x_vec_inf = zeros(n,N);
y_vec_inf = zeros(p,N);
x_vec_est_inf = x_vec_inf;

% initialization
x = zeros(n,1);
x_k_k1_inf = zeros(n,1);
d_k1_inf = 0;
tic
for ii = 1:N
    x = A*x + B*u(ii) + G*d(ii) + w(ii,:);
y = C*x + D*u(ii) + v(ii,:);
    x_vec_inf(:,ii) = x;
y_vec_inf(:,ii) = y;
    x_k_k1_inf = A*x_k_k1_inf + B*u(ii) + G*d_k1_inf;
x_k_k_inf = x_k_k1_inf + L_inf*(y - C*x_k_k1_inf - D*u(ii));
d_k1_inf = M_inf*(y - C*x_k_k_inf - D*u(ii));
d_k_vec_inf(ii) = d_k1_inf;
x_vec_est_inf(:,ii) = x_k_k_inf;
end
t_ss = toc;
disp('done SS')

% plotting
disp(['Time to compute Time-Varying filter: ' num2str(t_tv) 's'])
disp(['Time to compute Steady-State filter: ' num2str(t_ss) 's'])
figure(1)
plot(t, d_k_vec, t, d)
xlabel('time (s)')
ylabel('(kN)')
grid on
title('Time-Varying filter')
legend({'$\hat{F}_e$', '$F_e$'}, 'Interpreter', 'latex')
figure(2)
plot(t, d_k_vec_inf, t, d)
xlabel('time (s)')
ylabel('(kN)')
grid on
title('Steady-State filter')
legend({'$\hat{F}_e$', '$F_e$'}, 'Interpreter', 'latex')
figure(3)
plot(t, d_k_vec - d_k_vec_inf)
grid on
xlabel('Time (s)')
title('Difference between Time-Varying and Steady-State filters')

figure(4)
subplot 211
plot(t, y_vec(1,:))
xlabel('Time (s)')
ylabel('m')
title('Measured (noisy) Outputs')

legend({'$z$'}, 'Interpreter', 'latex')
grid on
subplot 212
plot(t, y_vec(2,:))
xlabel('Time (s)')
ylabel('m/s^2')

legend({'$\ddot{z}$'}, 'Interpreter', 'latex')
grid on

figure(5)
subplot 211
plot(t, x_vec(1,:), t, x_k_vec(1,:), t, x_vec_est_inf(1,:))
grid on
xlabel('time (s)')
ylabel('m/s')
legend({'$v$ ', '$\hat{v}$', '$\hat{v}_{\infty}$'}, 'Interpreter', 'latex')
title('Estimated states')

grid on
subplot 212
plot(t, x_vec(2,:), t, x_k_vec(2,:), t, x_vec_est_inf(2,:))
grid on
xlabel('time (s)')
ylabel('m')
legend({'$z$ ', '$\hat{z}$', '$\hat{z}_{\infty}$'}, 'Interpreter', 'latex')
grid on

State_and_unknown_input_estimator_position_acceleration.m
B Sample code: pressure and acceleration measurements

This section contains sample MATLAB code for implementing a pressure/acceleration state estimator for a WEC.

```matlab
% this scripts is to test the Unified Linear Input and State Estimator
% (ULISE algorithm) described in
% S. Z. Yong, M. Zhu, and E. Frazzoli, A unified filter for simultaneous
% input and state estimation of linear discrete-time stochastic systems,
% Both Time-Varying and Steady-State versions are implemented. Pressure and
% acceleration measurements are used to estimate state and excitation force
% Sandia National Laboratories is a multi-mission laboratory managed and
% operated by National Technology and Engineering Solutions of Sandia,
% LLC., a wholly owned subsidiary of Honeywell International, Inc., for the
% U.S. Department of Energy's National Nuclear Security Administration
% under contract DE-NA0003525.
% G. Bacelli, R. Coe
% Sandia National Laboratories
% 2017

clc
clear

% load identified parametric WEC model model
WEC = load('WEC_param_model_1DOF.mat'); % radiation impedance model
Gr = struct2array(load('Gr_model.mat')); % radiation pressure model
Ge = struct2array(load('Ge_model.mat')); % excitation pressure model

Tc = 1e-3;
N = 5e4;

% load excitation force
Fe = load('Exc_time_series.mat'); %(excitation force in kN)
t = (1:N)*Tc;
d = interp1(Fe.t_trim, Fe.Fexc_td, t, 'pchip')*1e-3; % control input (PTO force in kN). Open loop
, no control implemented
u = .5*sin(2*pi*0.75*t);

mass = 858.3987;
Ainf_hat = 822.3799;
K = 2.3981e+04;
Ar = WEC.rad_sys.a;
Br = WEC.rad_sys.b;
Cr = WEC.rad_sys.c;
Bf = WEC.B_eq_mat(9);

Am = [-Bf/(mass + Ainf_hat) -K/(mass + Ainf_hat) -Cr/(mass + Ainf_hat); 1 0 zeros(1,length(Ar));
2 zeros(length(Ar),1) Ar];
```

[21]
Cm = Am(:,1);
Dm = Bm(:,1);
Ai = blkdiag(Am, Gr.A);
Bi = [Bm; Gr.B];
Ci = blkdiag(Cm, Gr.C);
Di = [Dm; Gr.D];
ni = size(Ai,1);
ne = size(Ge.A,1);
Ac = [Ai, Bi*Ge.C; zeros(ne, ni) Ge.A];
Bc = [Bi, Bi*Ge.D; zeros(ne,1), Ge.B];
C = [Ci, Di*Ge.C];
D = [Di, Di*Ge.D + [0;1] ];
sys_c = ss(Ac, Bc, C, D);
sys_d = c2d(sys_c, Tc);
Ge_d = c2d(Ge, Tc);
Ce = Ge_d.C;
De = Ge_d.D;
n = size(C,2);
p = size(C,1);
% convert continuous time model to discrete time
A = expm(Ac*Tc);
B = Ac*(A - eye(n))'*Bc;
G = B(:,1);
B = B(:,1);
H = D(:,2);
D = D(:,1);
% process noise
w = .00001*(rand(N,ni+ne) -0.5); % measurements noise
v = (rand(N,p) -0.5) * diag([.05, 0.05]) ;
Q = cov(w);
R = cov(v);
% Time Varying filter
% initialize variables
P_x_k_k = 0.001*eye(n);
P_xd_k = 0.001*ones(n,1);
P_d_k = 0.001; % process noise
x = zeros(n,1);
x_k_k = x;
d_k = 0;
In = eye(n);
Ip = eye(p);
% preallocation
x_vec = zeros(n,N);
y_vec = zeros(p,N);
d_k_vec = zeros(N,1);
x_k_vec = zeros(n,N);
```matlab
Fe_est = zeros(N,1);
Fe = Fe_est;

tic
for ii = 1:N
    x = A*x + B*u(ii) + G*d(ii) + w(ii,:);
    y = C*x + D*u(ii) + H*d(ii) + v(ii,:);
    Fe(ii) = Ce*x(ni+1:end) + De*d(ii);
    x_vec(:,ii) = x;
    y_vec(:,ii) = y;
    x_k_k1 = A*x_k_k + B*u(ii) + G*d_k;
    P_x_k_k1 = A* P_x_k_k *A' + G*P_xd_k '*A' + A* P_xd_k *G' + G*P_d_k*G' + Q;
    R_t_k = C* P_x_k_k1 *C' + R;
    Kk = P_x_k_k1 *C'/ R_t_k;
    Lk = Kk*(Ip-{H/((H'/ R_t_k )*H)\H'/ R_t_k});
    x_k_k = x_k_k1 + Lk*(y - C*x_k_k1 - D*u(ii) );
    P_x_k_k = (In -Lk*C)* P_x_k_k1 *(In -Lk*C)';
    R_ts_k = (Ip -C*Lk)*R_t_k *(Ip -C*Lk)';
    d_k = M_k*(y - C*x_k_k - D*u(ii));
    x_k_vec(:,ii) = x_k_k;
    Fe_est(ii) = Ce*x_k_k(ni+1:end) + De*d_k;
end
t_tv = toc;
disp('done TV')

%% Steady State filter
L_inf = Lk;
M_inf = M_k;

% preallocation
d_k_vec_inf = zeros(N,1);
x_vec_inf = zeros(n,N);
y_vec_inf = zeros(p,N);
x_vec_est_inf = x_vec_inf;
Fe_est_inf = zeros(N,1);

% initialization
x = zeros(n,1);
x_k_k_inf = zeros(n,1);
d_k1_inf = 0;
tic
for ii = 1:N
    x = A*x + B*u(ii) + G*d(ii) + w(ii,:);
    y = C*x + D*u(ii) + H*d(ii) + v(ii,:);
    x_k_k1_inf = A*x_k_k_inf + B*u(ii) + G*d_k1_inf;
    P_x_k_k1_inf = A* P_x_k_k_inf *A' + G*P_xd_k '*A' + A* P_xd_k_inf *G' + G*P_d_k*G' + Q;
    R_t_k = C* P_x_k_k1_inf *C' + R;
    Kk = P_x_k_k1_inf *C'/ R_t_k;
    Lk = Kk*(Ip-{H/((H'/ R_t_k)*H)\H'/ R_t_k});
    x_k_k_inf = x_k_k1_inf + L_k*(y - C*x_k_k_inf - D*u(ii));
    d_k = M_k*(y - C*x_k_k_inf - D*u(ii));
    x_k_vec_inf(:,ii) = x_k_k_inf;
    Fe_est_inf(ii) = Ce*x_k_k_inf(ni+1:end) + De*d_k;
end
```

x_vec_est_inf(:,ii) = x_k_k_inf;
Fe_est_inf(ii) = Ce*x_k_k_inf(ni+1:end) + De*d_k1_inf;
end

t_ss = toc;
disp('done SS')

%%% plotting

disp(['Time to compute Time-Varying filter: ' num2str(t_tv) 's'])
disp(['Time to compute Steady-State filter: ' num2str(t_ss) 's'])

figure(1)
subplot 211
plot(t, d_k_vec, t, d)
xlabel('time (s)')
ylabel('(kPa)')
grid on
legend({'$\hat{P}e$', '$Pe$ ' }, 'Interpreter', 'latex')
title('Time-Varying filter: Unknown input (Excitation pressure Pe)')

subplot 212
plot(t, Fe_est, t, Fe)
grid on
xlabel('time (s)')
ylabel('(kN)')
title('Excitation force')
legend({'$\hat{F}e$', '$Fe$ '}, 'Interpreter', 'latex')

figure(2)

figure(3)

figure(4)

subplot 211
plot(t, y_vec(1,:))
xlabel('Time (s)')
ylabel('m/s^2')
title('Measured (noisy) Outputs')
legend({'$\ddot{z}$'}, 'Interpreter', 'latex')
grid on
subplot 212
plot(t, y_vec(2,:))
xlabel('Time (s)')
ylabel('kPa')
title('')
legend({'$P$ '}, 'Interpreter', 'latex')
grid on
State_and_unknown_input_estimator_pressure_acceleration.m
DISTRIBUTION:

1  MS 0899  Technical Library, 9536 (electronic copy)