CHAPTER 4
Extended and Decentralized Kalman Filtering

We complete our discussion of Kalman filtering with an outline of two popular extensions developed for nonlinear problems and distributed applications. Nonlinear state space models are often encountered, for example, in target tracking [6] and inertial navigation systems [8, 9]. Distributed estimation tasks arise frequently in the context of sensor networks [28] and multisensor tracking [29, 30]. In this chapter we examine the extended and decentralized Kalman filters and illustrate their utility through examples.

4.1 EXTENDED KALMAN FILTER

When the state evolution and measurement processes are nonlinear and Gaussian, the *extended Kalman filter* [6, 12] can be utilized for estimation of the states. The state space model for this case can be written as

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}) + \mathbf{v}_n, \tag{4.1a}$$
$$\mathbf{y}_n = \mathbf{h}(\mathbf{x}_n) + \mathbf{w}_n, \tag{4.1b}$$

where $\mathbf{f} : \mathbb{R}^D \mapsto \mathbb{R}^D$ is the state-transition function, $\mathbf{h} : \mathbb{R}^D \mapsto \mathbb{R}^M$ is the measurement function, and $\mathbf{v}_n$ and $\mathbf{w}_n$ are independent Gaussian noise vectors as defined earlier. Assuming that $\mathbf{f}$ and $\mathbf{h}$ are differentiable, the model can be linearized by making use of the Jacobian matrices:

$$\tilde{\mathbf{F}}_n = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{m}_{n-1|n-1}} = \left.\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_D} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_D}{\partial x_1} & \frac{\partial f_D}{\partial x_2} & \cdots & \frac{\partial f_D}{\partial x_D} \end{bmatrix}\right|_{\mathbf{m}_{n-1|n-1}}, \tag{4.2a}$$

$$\tilde{\mathbf{H}}_n = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{m}_{n|n-1}} = \left.\begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_D} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \cdots & \frac{\partial h_2}{\partial x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_M}{\partial x_1} & \frac{\partial h_M}{\partial x_2} & \cdots & \frac{\partial h_M}{\partial x_D} \end{bmatrix}\right|_{\mathbf{m}_{n|n-1}}. \tag{4.2b}$$

The extended Kalman filter equations are given by [6, 12]

$$\begin{aligned}
\mathbf{m}_{n|n-1} &= \mathbf{f}(\mathbf{m}_{n-1|n-1}), &\quad (4.3a)\\
\mathbf{P}_{n|n-1} &= \tilde{\mathbf{F}}_n\,\mathbf{P}_{n-1|n-1}\,\tilde{\mathbf{F}}_n^T + \mathbf{Q}_n, &\quad (4.3b)\\
\mathbf{S}_n &= \tilde{\mathbf{H}}_n\,\mathbf{P}_{n|n-1}\,\tilde{\mathbf{H}}_n^T + \mathbf{R}_n, &\quad (4.3c)\\
\mathbf{K}_n &= \mathbf{P}_{n|n-1}\,\tilde{\mathbf{H}}_n^T\,\mathbf{S}_n^{-1}, &\quad (4.3d)\\
\mathbf{m}_{n|n} &= \mathbf{m}_{n|n-1} + \mathbf{K}_n\,(\mathbf{y}_n - \mathbf{h}(\mathbf{m}_{n|n-1})), &\quad (4.3e)\\
\mathbf{P}_{n|n} &= \mathbf{P}_{n|n-1} - \mathbf{K}_n\,\tilde{\mathbf{H}}_n\,\mathbf{P}_{n|n-1}, &\quad (4.3f)
\end{aligned}$$

for $n = 1, 2, \ldots$, with Eqs. (4.3a)–(4.3b) representing the prediction step and Eqs. (4.3c)–(4.3f) comprising the update step. Note that the extended Kalman filter equations (4.3) reduce to the Kalman filter equations (3.5) when the state-transition function $\mathbf{f}$ and measurement function $\mathbf{h}$ are linear. The extended Kalman filtering algorithm is summarized below.

Algorithm 2 Extended Kalman filtering

Input: Nonlinear Gaussian state space model (4.1), and a set of measurements $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$.
Output: Estimates of the states $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ at time steps $n = 1, 2, \ldots, N$.

Initialize the mean and covariance of the Gaussian state distribution $\mathcal{N}(\mathbf{x}_0; \mathbf{m}_{0|0}, \mathbf{P}_{0|0})$ at time step $n = 0$.

For time steps $n = 1, 2, \ldots, N$, perform the following:

1. Carry out the prediction step

$$\begin{aligned}
\mathbf{m}_{n|n-1} &= \mathbf{f}(\mathbf{m}_{n-1|n-1}),\\
\mathbf{P}_{n|n-1} &= \tilde{\mathbf{F}}_n\,\mathbf{P}_{n-1|n-1}\,\tilde{\mathbf{F}}_n^T + \mathbf{Q}_n.
\end{aligned}$$

2. Compute the estimate $\hat{\mathbf{x}}_n = \mathbf{m}_{n|n}$ of the state $\mathbf{x}_n$ using the update step

$$\begin{aligned}
\mathbf{S}_n &= \tilde{\mathbf{H}}_n\,\mathbf{P}_{n|n-1}\,\tilde{\mathbf{H}}_n^T + \mathbf{R}_n,\\
\mathbf{K}_n &= \mathbf{P}_{n|n-1}\,\tilde{\mathbf{H}}_n^T\,\mathbf{S}_n^{-1},\\
\mathbf{m}_{n|n} &= \mathbf{m}_{n|n-1} + \mathbf{K}_n\,(\mathbf{y}_n - \mathbf{h}(\mathbf{m}_{n|n-1})),\\
\mathbf{P}_{n|n} &= \mathbf{P}_{n|n-1} - \mathbf{K}_n\,\tilde{\mathbf{H}}_n\,\mathbf{P}_{n|n-1}.
\end{aligned}$$

The MATLAB function `extended_kalman_filter.m` shown in Listing 4.1 performs one iteration of the extended Kalman filter prediction and update steps.

Other algorithms for nonlinear and/or non-Gaussian filtering include those based on unscented transforms [12, 31] and Monte Carlo techniques such as the particle filter [5, 16].

```
function [m_n,P_n] = extended_kalman_filter (m_n1,P_n1,y_n,f,Ft_n,Q_n...
    ,h,Ht_n,R_n)

% Extended Kalman filter prediction and update steps --- propagates
% Gaussian posterior state distribution from time step n-1 to time step n
% (for details, see Section 4.1 of the text)
%
% Inputs:
% m_n1 - (Dx1 vector) Gaussian posterior mean at time step n-1
% P_n1 - (DxD matrix) Gaussian posterior covariance at time step n-1
% y_n - (Mx1 vector) measurements at time step n
% f(x) - (Dx1 vector function of Dx1 vector x) state-transition function
% Ft_n - (DxD matrix) state-transition Jacobian matrix at time step n
% Q_n - (DxD matrix) Gaussian state noise covariance at time step n
% h(x) - (Mx1 vector function of Dx1 vector x) measurement function
% Ht_n - (MxD matrix) measurement Jacobian matrix at time step n
% R_n - (MxM matrix) Gaussian measurement noise covariance at time step n
%
% Outputs:
% m_n - (Dx1 vector) Gaussian posterior mean at time step n
% P_n - (DxD matrix) Gaussian posterior covariance at time step n

% Predict
m_nn1 = feval(f,m_n1);
P_nn1 = Ft_n*P_n1*Ft_n' + Q_n;

% Update
S_n = Ht_n*P_nn1*Ht_n' + R_n;
K_n = P_nn1*Ht_n'/S_n;
m_n = m_nn1 + K_n*(y_n-feval(h,m_nn1));
P_n = P_nn1 - K_n*Ht_n*P_nn1;
```

Listing 4.1: `extended_kalman_filter.m`

4.1.1 EXAMPLE: PREDATOR-PREY SYSTEM

We consider a bio-system occupied by a predator population and a prey population. This type of a system is modeled using the Lotka-Volterra equations [32]. In this model, the predator and prey interact based on the following equations:

$$x_{n,1} = \left[1 + \Delta t\left(1 - \frac{x_{n-1,2}}{c_2}\right)\right] x_{n-1,1} + v_{n,1}, \tag{4.4a}$$

$$x_{n,2} = \left[1 - \Delta t\left(1 - \frac{x_{n-1,1}}{c_1}\right)\right] x_{n-1,2} + v_{n,2}. \tag{4.4b}$$

Here, $x_{n,1}$ and $x_{n,2}$ denote respectively the prey and predator populations at time step n, Δt is the time interval between time steps $n-1$ and n, c_1 and c_2 are constant model parameters, and $v_{n,1}$ and $v_{n,2}$ are i.i.d. Gaussian with zero mean and variance σ_x^2. Clearly, the two populations do not live in isolation. For example, if left with an infinite food supply and no predators, a population of rabbits would experience normal population growth. Similarly, a population of foxes with no rabbit population would face decline. When together, these populations interact according to (4.4). Figure 4.1 depicts the cycle of predator-prey population dynamics.

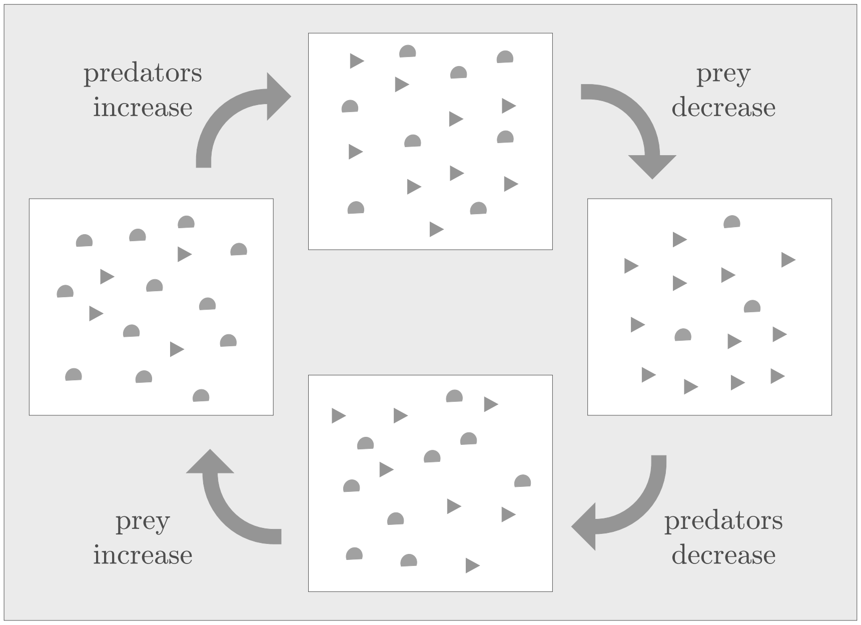


Figure 4.1: Cycle of predator-prey population dynamics. When there is an abundance of preys, the predator population increases. But this increase in predators drives the prey population down. With a scarcity of preys, the predator population is forced to decrease. The decrease in predators then results in prey population increasing, and the cycle continues.

In matrix notation, (4.4) can be expressed as

$$\begin{bmatrix} x_{n,1} \\ x_{n,2} \end{bmatrix} = \mathbf{f}\left(\begin{bmatrix} x_{n-1,1} \\ x_{n-1,2} \end{bmatrix}\right) + \begin{bmatrix} v_{n,1} \\ v_{n,2} \end{bmatrix}. \tag{4.5}$$

The state vector $\mathbf{x}_n = [x_{n,1}\ x_{n,2}]^T$ is two-dimensional ($D = 2$), $\mathbf{f} : \mathbb{R}^2 \mapsto \mathbb{R}^2$ is the nonlinear state-transition function, and the state noise vector $\mathbf{v}_n = [v_{n,1}\ v_{n,2}]^T$ is 2×1 and Gaussian with zero mean and covariance matrix $\mathbf{Q}_n = \sigma_x^2\,\mathbf{I}$.

We wish to estimate the predator and prey populations based on noisy measurements of the total population

$$y_n = x_{n,1} + x_{n,2} + w_n, \tag{4.6}$$

where w_n is Gaussian with zero mean and variance σ_y^2. In matrix notation,

$$y_n = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_{n,1} \\ x_{n,2} \end{bmatrix} + w_n. \tag{4.7}$$

Thus, the measurement vector $\mathbf{y}_n = y_n$ is one-dimensional ($M = 1$), the measurement matrix $\mathbf{H}_n = [1\ 1]$ is 1×2, and the measurement noise vector $\mathbf{w}_n = w_n$ is scalar and Gaussian with zero mean and variance $\mathbf{R}_n = \sigma_y^2$.

The state space model above is nonlinear and Gaussian, and the extended Kalman filter can be used for estimation. In particular, the state-transition function Jacobian matrix (Eq. (4.2a)) is 2×2 and given by

$$\tilde{\mathbf{F}}_n = \begin{bmatrix} 1 + \Delta t\left(1 - \frac{m_{n-1|n-1,2}}{c_2}\right) & -\Delta t\,\frac{m_{n-1|n-1,1}}{c_2} \\ \Delta t\,\frac{m_{n-1|n-1,2}}{c_1} & 1 - \Delta t\left(1 - \frac{m_{n-1|n-1,1}}{c_1}\right) \end{bmatrix}, \tag{4.8}$$

where $\mathbf{m}_{n-1|n-1} = \left[m_{n-1|n-1,1}\ m_{n-1|n-1,2}\right]^T$ is the mean of the Gaussian posterior pdf at time step $n-1$.

The MATLAB program `predator_prey.m` shown in Listing 4.2 performs $N = 2000$ time steps of predator-prey population estimation using the extended Kalman filter. The state space model parameters were $\Delta t = 0.01$, $c_1 = 300$, $c_2 = 200$, $\sigma_x = 1$, $\sigma_y = 10$, the initial prey and predator populations were set as $\mathbf{x}_0 = [400\ 100]^T$ (these values were selected to generate a plot similar to that in [33, Ch. 16]), and the extended Kalman filter was initialized with Gaussian mean $\mathbf{m}_{0|0} = \mathbf{x}_0$ and covariance $\mathbf{P}_{0|0} = 100\,\mathbf{I}$. The MATLAB function `extended_kalman_filter.m` shown in Listing 4.1 is utilized to carry out the extended Kalman filter prediction and update steps (Eq. (4.3)). Figure 4.2b shows a plot of the actual and estimated predator-prey populations. It can be seen that the extended Kalman filter is able to track the predator-prey dynamics with good accuracy.

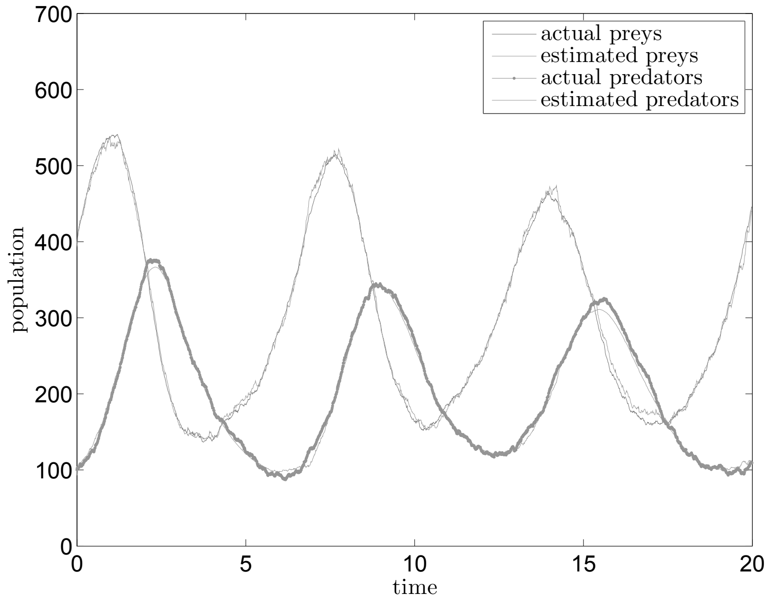
```
% Solution of Predator-Prey Equations using the Extended Kalman Filter

function predator_prey

% State space model
delt = 0.01;                 % Time step interval
c1 = 300; c2 = 200;          % Predator-Prey model parameters
sigma_x = 1;                 % State noise standard deviation
Q_n = sigma_x^2*eye(2);      % State noise covariance matrix
H_n = [1 1];                 % Measurement matrix
sigma_y = 10;                % Measurement noise standard deviation
R_n = sigma_y^2;             % Measurement noise covariance matrix

% Initialization
x(1:2,1) = [400; 100];       % Initial prey and predator populations
m(1:2,1) = x(1:2,1);         % Gaussian posterior mean at time step 1
P(1:2,1:2,1) = 100*eye(2);   % Gaussian posterior covariance at time step 1

% Predator-Prey estimation using extended Kalman filter
for n = 2 : 2000,                     % Time steps

    % State propagation
    v_n = sigma_x*randn(2,1);         % State noise vector
    x(1:2,n) = f(x(1:2,n-1)) + v_n;   % Markov nonlinear Gaussian evolution

    % Generate measurements
    w_n = sigma_y*randn(1,1);         % Measurement noise vector
    y_n = H_n*x(1:2,n) + w_n;         % Linear Gaussian measurements

    % State-transition function Jacobian matrix
    Ft_n = [1+delt*(1-m(2,n-1)/c2) -delt*m(1,n-1)/c2;...
        delt*m(2,n-1)/c1 1-delt*(1-m(1,n-1)/c1)];

    % Compute Gaussian posterior mean and covariance at time step n
    [m(1:2,n),P(1:2,1:2,n)] = extended_kalman_filter (m(1:2,n-1),...
        P(1:2,1:2,n-1),y_n,@f,Ft_n,Q_n,@h,H_n,R_n);
end

```

```
% Plot actual and estimated predator-prey populations
% Evolution in time
t = [0:1999]'*delt;
figure, plot(t,x(1,:),'b'); hold on, plot(t,m(1,:),'r--');
hold on, plot(t,x(2,:),'g.-'); hold on, plot(t,m(2,:),'m-.');
xlabel('time'); ylabel('population');
title('Solution of predator-prey equations using extended Kalman filter');
legend('actual preys','estimated preys','actual predators',...
    'estimated predators');
% Phase plot
figure, plot(x(1,:),x(2,:),'b'); hold on, plot(m(1,:),m(2,:),'r--');
xlabel('prey population'); ylabel('predator population');
title('Solution of predator-prey equations using extended Kalman filter');
legend('actual','estimated');

% State-transition function
function x_n = f (x_n1)
    x_n = [(1+delt*(1-x_n1(2,1)/c2))*x_n1(1,1); ...
        (1-delt*(1-x_n1(1,1)/c1))*x_n1(2,1)];
end

% Measurement function
function y_n = h (x_n)
    y_n = x_n(1,1) + x_n(2,1);
end

end
```

Listing 4.2: `predator_prey.m`

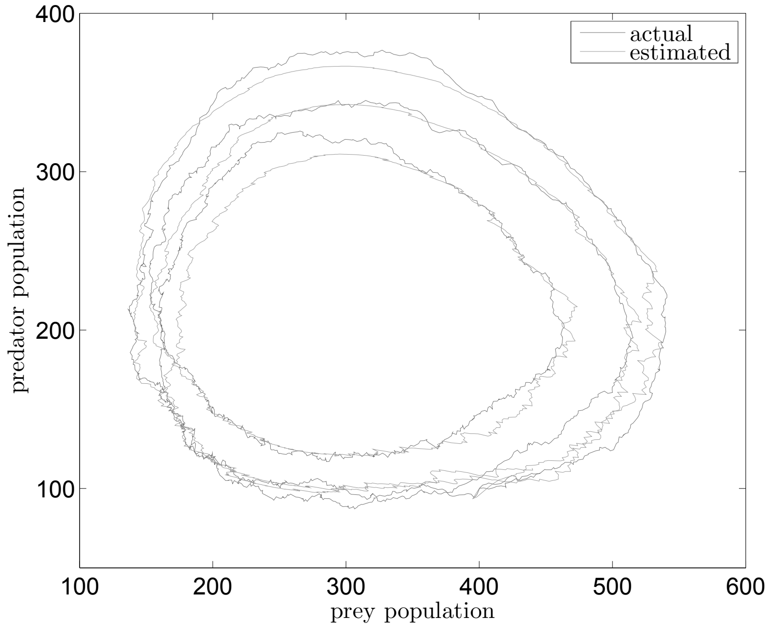
4.2 DECENTRALIZED KALMAN FILTERING

In distributed multisensor network configurations, the *decentralized Kalman filter* [34] enables estimation for linear Gaussian state space models without the need for a central communication and processing facility. The resulting estimation framework is robust to failure of one or more nodes and can yield performance benefits such as faster processing capability. In this scheme, each sensing node first collects measurements about the system's state and communicates a small amount of information with the other nodes. Next, each node locally processes the information received from the other nodes in conjunction with its own data to compute an overall estimate



(a) Evolution in time

Figure 4.2a: Solution of the Predator-Prey equations using the extended Kalman filter. The initial prey and predator populations are 400 and 100, respectively, and evolve according the dynamical law given in (4.4), with $c_1 = 300$ and $c_2 = 200$. Noisy total population measurements are collected as specified in (4.6), with $\sigma_y = 10$. $N = 2000$ time steps of extended Kalman filtering are performed and the actual and estimated predator-prey populations are shown in the plots.



(b) Phase plot

Figure 4.2b: Solution of the Predator-Prey equations using the extended Kalman filter. The initial prey and predator populations are 400 and 100, respectively, and evolve according the dynamical law given in (4.4), with $c_1 = 300$ and $c_2 = 200$. Noisy total population measurements are collected as specified in (4.6), with $\sigma_y = 10$. $N = 2000$ time steps of extended Kalman filtering are performed and the actual and estimated predator-prey populations are shown in the plots.

of the state. In the following brief description, it is assumed for simplicity that measurement validation is not used and that the communication between the nodes is synchronized; the reader is referred to [34] for the general method.

Using matrix inversion lemmas and some algebra, the Kalman filter update equations (3.5e)–(3.5f) for the mean $\mathbf{m}_{n|n}$ and covariance $\mathbf{P}_{n|n}$ of the (Gaussian) posterior pdf $p(\mathbf{x}_n|\mathbf{Y}_n)$ at time step n can be rewritten in *information form* as

$$\begin{aligned} \mathbf{P}_{n|n}^{-1} &= \mathbf{P}_{n|n-1}^{-1} + \mathbf{H}_n^T\,\mathbf{R}_n^{-1}\,\mathbf{H}_n, & (4.9a)\\ \mathbf{P}_{n|n}^{-1}\,\mathbf{m}_{n|n} &= \mathbf{P}_{n|n-1}^{-1}\,\mathbf{m}_{n|n-1} + \mathbf{H}_n^T\,\mathbf{R}_n^{-1}\,\mathbf{y}_n. & (4.9b)\end{aligned}$$

Let K be the number of distributed nodes used to collect measurements, with the measurement equation for each of the nodes expressed as

$$\mathbf{y}_{n,k} = \mathbf{H}_{n,k}\,\mathbf{x}_n + \mathbf{w}_{n,k}, \quad k = 1, \ldots, K, \tag{4.10}$$

where $\mathbf{y}_{n,k}$ denotes the $M_k \times 1$ vector of measurements collected by sensor node k at time step n, $\mathbf{H}_{n,k}$ is the $M_k \times D$ measurement matrix for node k, $\mathbf{x}_n$ is the $D \times 1$ state vector, and $\mathbf{w}_{n,k}$ is a $M_k \times 1$ Gaussian random measurement noise vector with zero mean and covariance matrix $\mathbf{R}_{n,k}$. In block matrix notation,

$$\begin{bmatrix} \mathbf{y}_{n,1} \\ \mathbf{y}_{n,2} \\ \vdots \\ \mathbf{y}_{n,K} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{n,1} \\ \mathbf{H}_{n,2} \\ \vdots \\ \mathbf{H}_{n,K} \end{bmatrix} \mathbf{x}_n + \begin{bmatrix} \mathbf{w}_{n,1} \\ \mathbf{w}_{n,2} \\ \vdots \\ \mathbf{w}_{n,K} \end{bmatrix}, \tag{4.11}$$

or

$$\mathbf{y}_n = \mathbf{H}_n\,\mathbf{x}_n + \mathbf{w}_n, \tag{4.12}$$

where $\mathbf{y}_n = [\mathbf{y}_{n,1}^T\ \mathbf{y}_{n,2}^T\ \ldots\ \mathbf{y}_{n,K}^T]^T$ is the combined $M \times 1$ measurement vector ($M = \sum_{k=1}^{K} M_k$), $\mathbf{H}_n = [\mathbf{H}_{n,1}^T\ \mathbf{H}_{n,2}^T\ \ldots\ \mathbf{H}_{n,K}^T]^T$ is the combined $M \times D$ measurement matrix, and $\mathbf{w}_n = [\mathbf{w}_{n,1}^T\ \mathbf{w}_{n,2}^T\ \ldots\ \mathbf{w}_{n,K}^T]^T$ is the combined $M \times 1$ Gaussian random measurement noise vector with zero mean and covariance matrix

$$\mathbf{R}_n = \begin{bmatrix} \mathbf{R}_{n,1} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_{n,2} & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{R}_{n,K} \end{bmatrix}. \tag{4.13}$$

Here, the measurement noise vectors of the different sensor nodes have been assumed to be uncorrelated. Due to the block structure of the measurement model, we can write

$$\mathbf{H}_n^T \mathbf{R}_n^{-1} \mathbf{H}_n = \sum_{k=1}^{K} \mathbf{H}_{n,k}^T \mathbf{R}_{n,k}^{-1} \mathbf{H}_{n,k}, \tag{4.14a}$$

$$\mathbf{H}_n^T \mathbf{R}_n^{-1} \mathbf{y}_n = \sum_{k=1}^{K} \mathbf{H}_{n,k}^T \mathbf{R}_{n,k}^{-1} \mathbf{y}_{n,k}. \tag{4.14b}$$

Substituting into (4.9) yields

$$\mathbf{P}_{n|n}^{-1} = \mathbf{P}_{n|n-1}^{-1} + \sum_{k=1}^{K} \mathbf{H}_{n,k}^T \mathbf{R}_{n,k}^{-1} \mathbf{H}_{n,k}, \tag{4.15a}$$

$$\mathbf{P}_{n|n}^{-1} \mathbf{m}_{n|n} = \mathbf{P}_{n|n-1}^{-1} \mathbf{m}_{n|n-1} + \sum_{k=1}^{K} \mathbf{H}_{n,k}^T \mathbf{R}_{n,k}^{-1} \mathbf{y}_{n,k}, \tag{4.15b}$$

which represent the decentralized Kalman filter update equations [34]. The quantities inside the summation terms in (4.15) need to be communicated between the nodes in order to perform the state update step.

Note that the decentralized Kalman filter update step (4.15) is mathematically equivalent to the (centralized) Kalman filter update step in Eqs. (3.5c)–(3.5f); no approximation has been used. Assuming that the same global state evolution model of Eq. (3.1a) (and therefore the same prediction step in Eqs. (3.5a)–(3.5b)) is used by all nodes, with identical initialization, the decentralized Kalman filter provides exactly the same state estimates as the centralized Kalman filter.

Computationally, the decentralized Kalman filter can be viewed as a parallelization of the centralized Kalman filter, in which the central measurement process has been divided between the K distributed sensor nodes. Since (a) the local measurement model at each node is smaller (dimension M_k for node k) than the global measurement model (dimension M), (b) the overhead associated with the combination stage (summations in (4.15)) is small, and (c) the nodes perform their computations in parallel, significant savings are afforded in terms of processing speed (ignoring communication delays).

The decentralized Kalman filtering algorithm is summarized below.

4.2.1 EXAMPLE: DISTRIBUTED OBJECT TRACKING

We consider the tracking of a moving object based on information collected using multiple sensor nodes. Such a task can arise during satellite and cellular network-based navigation, as depicted in Figure 4.3. We are interested in estimating the object's position and velocity in a distributed manner, without making use of a central processor.

Let (x_n, y_n) denote the object's position at time step n and $(\dot{x}_n, \dot{y}_n)$ its velocity at time step n, so that the state vector $\mathbf{x}_n = [x_n \; y_n \; \dot{x}_n \; \dot{y}_n]^T$ is four-dimensional ($D = 4$). Assume that

Algorithm 3 Decentralized Kalman filtering

Input: Linear Gaussian state evolution model (3.1a), linear Gaussian measurement model (4.10), and a set of measurements $\{\mathbf{y}_{1,k}, \mathbf{y}_{2,k}, \dots, \mathbf{y}_{N,k}\}$ at each of K distributed nodes ($k = 1, 2, \dots, K$).
Output: Estimates of the states $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ at time steps $n = 1, 2, \dots, N$, at each node k.

Initialize the mean and covariance of the Gaussian state distribution $\mathcal{N}(\mathbf{x}_0; \mathbf{m}_{0|0}, \mathbf{P}_{0|0})$ at time step $n = 0$, at each node k.

For time steps $n = 1, 2, \dots, N$, at each node k perform the following:

1. Carry out the prediction step

$$\begin{aligned} \mathbf{m}_{n|n-1} &= \mathbf{F}_n\, \mathbf{m}_{n-1|n-1}, \\ \mathbf{P}_{n|n-1} &= \mathbf{F}_n\, \mathbf{P}_{n-1|n-1}\, \mathbf{F}_n^T + \mathbf{Q}_n. \end{aligned}$$

2. Calculate the quantities $\mathbf{H}_{n,k}^T\, \mathbf{R}_{n,k}^{-1}\, \mathbf{H}_{n,k}$ and $\mathbf{H}_{n,k}^T\, \mathbf{R}_{n,k}^{-1}\, \mathbf{y}_{n,k}$ and communicate to all other nodes.
3. Compute the estimate $\hat{\mathbf{x}}_n = \mathbf{m}_{n|n}$ of the state $\mathbf{x}_n$ using the update step

$$\begin{aligned} \mathbf{P}_{n|n} &= \left(\mathbf{P}_{n|n-1}^{-1} + \sum_{k=1}^{K} \mathbf{H}_{n,k}^T\, \mathbf{R}_{n,k}^{-1}\, \mathbf{H}_{n,k} \right)^{-1}, \\ \mathbf{m}_{n|n} &= \mathbf{P}_{n|n} \left(\mathbf{P}_{n|n-1}^{-1}\, \mathbf{m}_{n|n-1} + \sum_{k=1}^{K} \mathbf{H}_{n,k}^T\, \mathbf{R}_{n,k}^{-1}\, \mathbf{y}_{n,k} \right). \end{aligned}$$

the object moves according to the approximately constant velocity model given in (3.12) (see Subsection 3.3.1).

In this example, the measurement system is comprised of $K = 3$ sensor nodes, with node 1 measuring the object's x-position, node 2 measuring the object's y-position, and node 3 measuring the object's x-position and x-velocity. Thus, the dimensionality of the local measurements is $M_1 = 1$, $M_2 = 1$, and $M_3 = 2$, and the measurement vectors are

$$\begin{aligned} \mathbf{y}_{n,1} &= \mathbf{H}_{n,1}\, \mathbf{x}_n + \mathbf{w}_{n,1}, && (4.16a) \\ \mathbf{y}_{n,2} &= \mathbf{H}_{n,2}\, \mathbf{x}_n + \mathbf{w}_{n,2}, && (4.16b) \\ \mathbf{y}_{n,3} &= \mathbf{H}_{n,3}\, \mathbf{x}_n + \mathbf{w}_{n,3}, && (4.16c) \end{aligned}$$

with the measurement matrices $\mathbf{H}_{n,1} = [1\ 0\ 0\ 0]$, $\mathbf{H}_{n,2} = [0\ 1\ 0\ 0]$, and $\mathbf{H}_{n,3} = \left[\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{smallmatrix}\right]$, and the measurement noise covariance matrices $\mathbf{R}_{n,1} = \sigma_y^2$, $\mathbf{R}_{n,2} = \sigma_y^2$, and $\mathbf{R}_{n,3} = \sigma_y^2\, \mathbf{I}$. Note that the object's y-velocity is not measured by any of the sensing nodes.

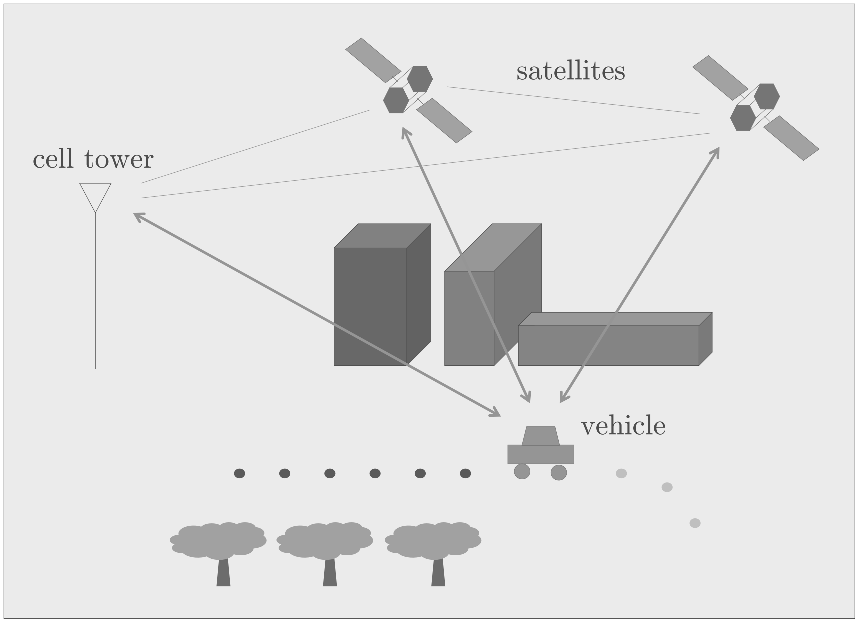


Figure 4.3: A satellite and cellular network-based navigation scenario. The automobile's position and velocity are estimated based on radio signals from navigation satellites within range and cellular network data where available.

The decentralized Kalman filter is now used to estimate the moving object's position and velocity. The MATLAB program `distributed_tracking.m` shown in Listing 4.3 performs $N = 100$ time steps of distributed tracking using the decentralized Kalman filter. The state space model parameters were $\Delta t = 0.5$, $\sigma_x = 0.1$, $\sigma_y = 0.25$, and the initial object position and velocity were set as $\mathbf{x}_0 = [1\ 1\ 0.1\ 0.1]^T$. Decentralized Kalman filtering is performed at each of the three nodes, and is initialized with Gaussian mean $\mathbf{m}_{0|0} = \mathbf{0}$ and covariance $\mathbf{P}_{0|0} = \mathbf{I}$. For time steps $n < 40$ and $n \geq 70$, perfect communication occurs between all nodes. For time steps $40 \leq n < 70$, nodes 1 and 2 have no communication with node 3. Figure 4.4c shows plots of the actual and estimated object position and the mean square error (MSE) in position and velocity. The MSE is computed using Monte Carlo simulation of 1 million trajectories, with object states initialized as $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{x}_0; \mathbf{0}, \mathbf{I})$. For comparison, the performance of the centralized Kalman filter (using the same data) implemented in the MATLAB function `kalman_filter.m` shown in Listing 3.1 is also given.

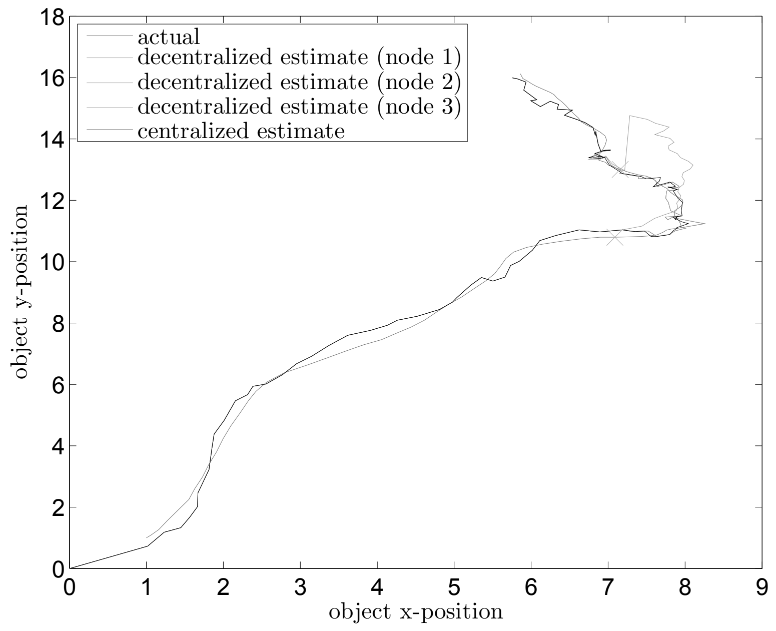
It can be seen that decentralized Kalman filtering is able to track the moving object accurately. For time steps $n < 40$, the decentralized Kalman filter estimates for all three nodes coincide with those from the centralized Kalman filter. This is expected, as the filters have the same information, initialization, prediction, and update. For time steps $40 \leq n < 70$, the decentralized

Kalman filter estimates for nodes 1 and 2 are different from those for node 3 (and also different from the centralized Kalman filter estimates). In particular, the decentralized estimates have a higher error than the centralized estimates. This is because nodes 1 and 2 communicate their local information to compute the decentralized estimates, while node 3 uses its local information to compute the decentralized estimates (and the centralized Kalman filter still uses all information). For time steps $n \geq 70$, with normal communications restored, the decentralized Kalman filter estimates for all three nodes converge back to those from the centralized Kalman filter.

```
% Distributed Object Tracking using the Decentralized Kalman Filter

% State space model
delt = 0.5;                     % Time step interval
F_n = eye(4); F_n(1,3) = delt;...
    F_n(2,4) = delt;            % State-transition matrix
sigma_x = 0.1; Q_n = sigma_x^2*[delt^3/3 0 delt^2/2 0;...
    0 delt^3/3 0 delt^2/2; delt^2/2 0 delt 0;...
    0 delt^2/2 0 delt];         % State noise covariance matrix
sigma_y = 0.25;                 % Measurement noise standard deviation
H_n1 = [1 0 0 0];               % Node 1 measurement matrix
R_n1 = sigma_y^2;               % Node 1 measurement noise covariance matrix
H_n2 = [0 1 0 0];               % Node 2 measurement matrix
R_n2 = sigma_y^2;               % Node 2 measurement noise covariance matrix
H_n3 = [1 0 0 0; 0 0 1 0];      % Node 3 measurement matrix
R_n3 = sigma_y^2*eye(2);        % Node 3 measurement noise covariance matrix
H_n = [H_n1; H_n2; H_n3];       % Central measurement matrix
R_n = blkdiag(R_n1,R_n2,R_n3);% Central measurement noise covariance matrix

% Initialization
x(1:4,1) = [1; 1; 0.1; 0.1];% Initial object position and velocity
m1(1:4,1) = zeros(4,1);     % Initial node 1 Gaussian posterior mean
P1(1:4,1:4,1) = eye(4);     % Initial node 1 Gaussian posterior covariance
m2(1:4,1) = zeros(4,1);     % Initial node 2 Gaussian posterior mean
P2(1:4,1:4,1) = eye(4);     % Initial node 2 Gaussian posterior covariance
m3(1:4,1) = zeros(4,1);     % Initial node 3 Gaussian posterior mean
P3(1:4,1:4,1) = eye(4);     % Initial node 3 Gaussian posterior covariance
m(1:4,1) = zeros(4,1);      % Initial central Gaussian posterior mean
P(1:4,1:4,1) = eye(4);      % Initial central Gaussian posterior covariance

% Track object using the decentralized Kalman filter
```

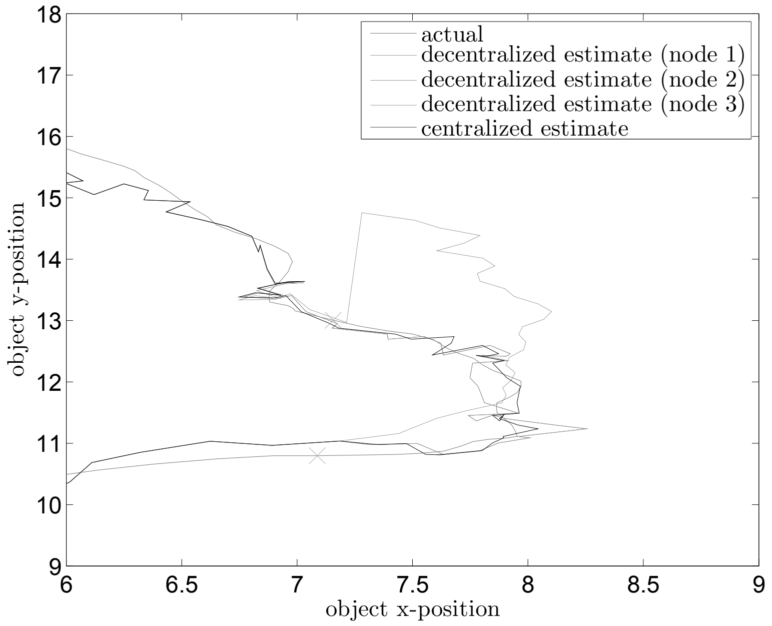
```
for n = 2 : 100,                     % Time steps

    % State propagation
    v_n = mvnrnd([0 0 0 0],Q_n)';    % State noise vector
    x(1:4,n) = F_n*x(1:4,n-1) + v_n; % Markov linear Gaussian evolution

    % Generate measurements
    w_n1 = sigma_y*randn(1,1);       % Node 1 measurement noise vector
    y_n1 = H_n1*x(1:4,n) + w_n1;     % Node 1 linear Gaussian measurements
    w_n2 = sigma_y*randn(1,1);       % Node 2 measurement noise vector
    y_n2 = H_n2*x(1:4,n) + w_n2;     % Node 2 linear Gaussian measurements
    w_n3 = sigma_y*randn(2,1);       % Node 3 measurement noise vector
    y_n3 = H_n3*x(1:4,n) + w_n3;     % Node 3 linear Gaussian measurements
    y_n = [y_n1; y_n2; y_n3];        % Central linear Gaussian measurements

    if (n<=40 || n>70)

        % Perfect communication between all nodes

        % Decentralized Kalman filter computations at node 1
        % Predict
        m1_nn1 = F_n*m1(1:4,n-1);
        P1_nn1 = F_n*P1(1:4,1:4,n-1)*F_n' + Q_n;
        % Update
        P1(1:4,1:4,n) = inv(inv(P1_nn1) + H_n1'*(R_n1\H_n1) +...
            H_n2'*(R_n2\H_n2) + H_n3'*(R_n3\H_n3));
        m1(1:4,n) = P1(1:4,1:4,n)*(P1_nn1\m1_nn1 + H_n1'*...
            (R_n1\y_n1) + H_n2'*(R_n2\y_n2) + H_n3'*(R_n3\y_n3));

        % Decentralized Kalman filter computations at node 2
        % Predict
        m2_nn1 = F_n*m2(1:4,n-1);
        P2_nn1 = F_n*P2(1:4,1:4,n-1)*F_n' + Q_n;
        % Update
        P2(1:4,1:4,n) = inv(inv(P2_nn1) + H_n1'*(R_n1\H_n1) +...
            H_n2'*(R_n2\H_n2) + H_n3'*(R_n3\H_n3));
        m2(1:4,n) = P2(1:4,1:4,n)*(P2_nn1\m2_nn1 + H_n1'*...
            (R_n1\y_n1) + H_n2'*(R_n2\y_n2) + H_n3'*(R_n3\y_n3));

        % Decentralized Kalman filter computations at node 3
        % Predict
        m3_nn1 = F_n*m3(1:4,n-1);
        P3_nn1 = F_n*P3(1:4,1:4,n-1)*F_n' + Q_n;
        % Update
        P3(1:4,1:4,n) = inv(inv(P3_nn1) + H_n1'*(R_n1\H_n1) +...
            H_n2'*(R_n2\H_n2) + H_n3'*(R_n3\H_n3));
        m3(1:4,n) = P3(1:4,1:4,n)*(P3_nn1\m3_nn1 + H_n1'*...
            (R_n1\y_n1) + H_n2'*(R_n2\y_n2) + H_n3'*(R_n3\y_n3));

    else

        % Nodes 1 and 2 have no communication with node 3

        % Decentralized Kalman filter computations at node 1
        % Predict
        m1_nn1 = F_n*m1(1:4,n-1);
        P1_nn1 = F_n*P1(1:4,1:4,n-1)*F_n' + Q_n;
        % Update
        P1(1:4,1:4,n) = inv(inv(P1_nn1) + H_n1'*(R_n1\H_n1) +...
            H_n2'*(R_n2\H_n2));
        m1(1:4,n) = P1(1:4,1:4,n)*(P1_nn1\m1_nn1 + H_n1'*...
            (R_n1\y_n1) + H_n2'*(R_n2\y_n2));

        % Decentralized Kalman filter computations at node 2
        % Predict
        m2_nn1 = F_n*m2(1:4,n-1);
        P2_nn1 = F_n*P2(1:4,1:4,n-1)*F_n' + Q_n;
        % Update
        P2(1:4,1:4,n) = inv(inv(P2_nn1) + H_n1'*(R_n1\H_n1) +...
            H_n2'*(R_n2\H_n2));
        m2(1:4,n) = P2(1:4,1:4,n)*(P2_nn1\m2_nn1 + H_n1'*...
            (R_n1\y_n1) + H_n2'*(R_n2\y_n2));

        % Decentralized Kalman filter computations at node 3
        % Predict
        m3_nn1 = F_n*m3(1:4,n-1);
        P3_nn1 = F_n*P3(1:4,1:4,n-1)*F_n' + Q_n;
        % Update
        P3(1:4,1:4,n) = inv(inv(P3_nn1) + H_n3'*(R_n3\H_n3));
        m3(1:4,n) = P3(1:4,1:4,n)*(P3_nn1\m3_nn1 + H_n3'*(R_n3\y_n3));

    end

    % Centralized Kalman filter computations
    [m(1:4,n),P(1:4,1:4,n)] = kalman_filter (m(1:4,n-1),...
        P(1:4,1:4,n-1),y_n,F_n,Q_n,H_n,R_n);
end

% Plot actual and estimated object position
figure, plot(x(1,:),x(2,:)); hold on, plot(m1(1,:),m1(2,:),'r');
plot(m2(1,:),m2(2,:),'g--'); plot(m3(1,:),m3(2,:),'m-.');
plot(m(1,:),m(2,:),'k:');
xlabel('object x-position'); ylabel('object y-position');
title('Distributed object tracking using the decentralized Kalman filter');
legend('actual','decentralized estimate (node 1)',...
    'decentralized estimate (node 2)','decentralized estimate (node 3)',...
    'centralized estimate');
```

Listing 4.3: `distributed_tracking.m`



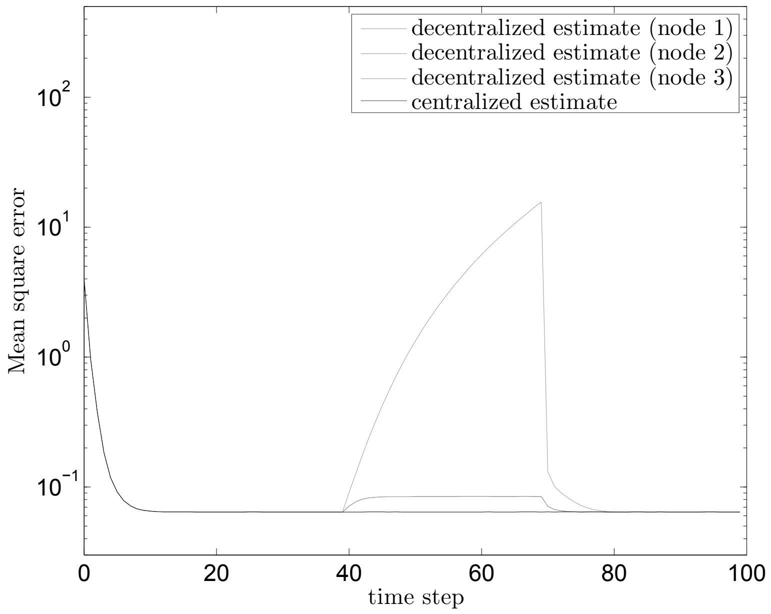
(a) Object trajectory and tracking

Figure 4.4a: Distributed object tracking using the decentralized Kalman filter. The object moves according the dynamical law given in (3.12), with $\Delta t = 0.5$ and $\sigma_x = 0.1$. Noisy measurements are collected using $K = 3$ sensor nodes as specified in (4.16), with $\sigma_y = 0.25$. $N = 100$ time steps of decentralized Kalman filtering are performed and the actual and estimated object position and the mean square error in position and velocity are shown in the plots. For time steps $n < 40$ and $n \geq 70$, perfect communication occurs between all nodes, and for time steps $40 \leq n < 70$, nodes 1 and 2 have no communication with node 3 (these time instants are indicated on the object's trajectory with cyan '$\times$'s). For comparison, the performance of a centralized Kalman filter (using the same data) is also given.



(b) Zoom of the region $6 \leq x \leq 9$ and $9 \leq y \leq 18$

Figure 4.4b: Distributed object tracking using the decentralized Kalman filter. The object moves according the dynamical law given in (3.12), with $\Delta t = 0.5$ and $\sigma_x = 0.1$. Noisy measurements are collected using $K = 3$ sensor nodes as specified in (4.16), with $\sigma_y = 0.25$. $N = 100$ time steps of decentralized Kalman filtering are performed and the actual and estimated object position and the mean square error in position and velocity are shown in the plots. For time steps $n < 40$ and $n \geq 70$, perfect communication occurs between all nodes, and for time steps $40 \leq n < 70$, nodes 1 and 2 have no communication with node 3 (these time instants are indicated on the object's trajectory with cyan '$\times$'s). For comparison, the performance of a centralized Kalman filter (using the same data) is also given.



(c) Mean square error (MSE) in estimation of object position and velocity

Figure 4.4c: Distributed object tracking using the decentralized Kalman filter. The object moves according the dynamical law given in (3.12), with $\Delta t = 0.5$ and $\sigma_x = 0.1$. Noisy measurements are collected using $K = 3$ sensor nodes as specified in (4.16), with $\sigma_y = 0.25$. $N = 100$ time steps of decentralized Kalman filtering are performed and the actual and estimated object position and the mean square error in position and velocity are shown in the plots. For time steps $n < 40$ and $n \geq 70$, perfect communication occurs between all nodes, and for time steps $40 \leq n < 70$, nodes 1 and 2 have no communication with node 3 (these time instants are indicated on the object's trajectory with cyan '$\times$'s). For comparison, the performance of a centralized Kalman filter (using the same data) is also given.