Chapter 9

Cannon-Launched Projectile Tracking Problem

Introduction

IN ALL of the applications so far, there was only one sensor measurement from which we built our Kalman filters. In this chapter we will attempt to track a cannon-launched projectile with a sensor that measures both the range and angle to the projectile. We will first see how the extra sensor measurement is incorporated into an extended Kalman filter. In this example the real world is linear in the Cartesian coordinate system, but the measurements are nonlinear. The extended Kalman filter will initially be designed in the Cartesian coordinate system to establish a benchmark for tracking performance. We will then see if performance can be improved by redesigning the filter in the polar coordinate system, where the measurements are linear but the model of the real world is nonlinear. Finally, we will also see if linear coupled and decoupled polynomial Kalman filters also can be made to work for this problem.

Problem Statement

Consider another simple example in which a radar tracks a cannon-launched projectile traveling in a two-dimensional, drag-free environment, as shown in Fig. 9.1. After the projectile is launched at an initial velocity, only gravity acts on the projectile. The tracking radar, which is located at coordinates x_R, y_R of Fig. 9.1, measures the range r and angle θ from the radar to the projectile.

In this example the projectile is launched at an initial velocity of 3000 ft/s at a 45-deg angle with respect to the horizontal axis in a zero-drag environment (i.e., atmosphere is neglected). The radar is located on the ground and is 100,000 ft downrange from where the projectile is initially launched. The radar is considered to have an angular measurement accuracy of 0.01 rad and a range measurement accuracy of 100 ft.

Before we can proceed with the development of a filter for this application, it is important to first get a numerical and analytical feel for all aspects of the problem. Because only gravity acts on the projectile, we can say that in the downrange or x direction there is no acceleration, whereas in the altitude or y direction there is the acceleration of gravity g or[1]

$$\ddot{x}_T = 0$$
$$\ddot{y}_T = -g$$

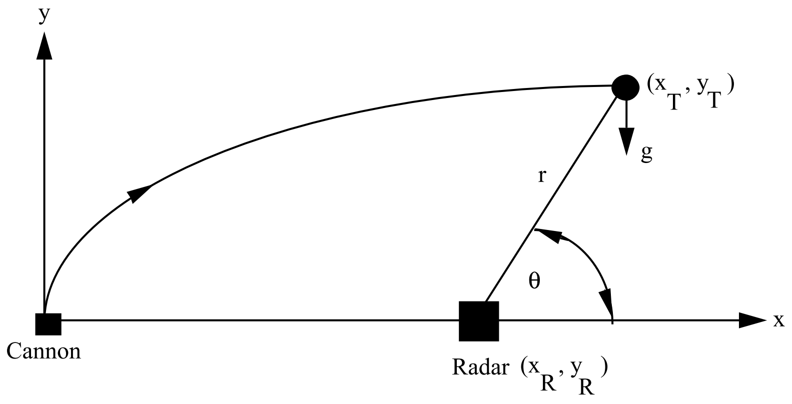

Fig. 9.1 Radar tracking cannon-launched projectile.

where the acceleration of gravity has a value of 32.2 ft/s^2 in the English system of units. Using trigonometry, the angle and range from the radar to the projectile can be obtained by inspection of Fig. 9.1 to be

$$\theta = \tan^{-1}\left(\frac{y_T - y_R}{x_T - x_R}\right)$$

$$r = \sqrt{(x_T - x_R)^2 + (y_T - y_R)^2}$$

Before we build an extended Kalman filter to track the projectile and estimate its position and velocity at all times based on noisy angle and range measurements, let us see what can be done without any filtering at all. By inspection of Fig. 9.1, we can see that we can express the location of the cannon-launched projectile (i.e., x_T, y_T) in terms of the radar range and angle as

$$x_T = r\cos\theta + x_R$$
$$y_T = r\sin\theta + y_R$$

If the radar coordinates are known precisely and r^* and θ^* are the noisy radar range and angle measurements, then the location of the projectile at any time can be calculated directly or estimated without any filtering at all as

$$\hat{x}_T = r^*\cos\theta^* + x_R$$
$$\hat{y}_T = r^*\sin\theta^* + y_R$$

The estimates of the velocity components of the projectile can be obtained from the noisy position estimates by using the definition of a derivative from calculus. In other words, we simply subtract the old position estimate from the new position estimate and divide by the sampling time or the time between measurements to get the estimated projectile velocity components or

$$\hat{\dot{x}}_{T_k} = \frac{\hat{x}_{T_k} - \hat{x}_{t_{k-1}}}{T_s}$$

$$\hat{\dot{y}}_{T_k} = \frac{\hat{y}_{T_k} - \hat{y}_{t_{k-1}}}{T_s}$$

Figure 9.2 displays the true projectile trajectory and the estimated projectile trajectory based on the raw radar measurements. We can see that the estimated trajectory appears to be quite close to the actual trajectory, so one might wonder why filtering techniques are required in this example. On the other hand, Figs. 9.3 and 9.4 show that when the raw measurements are used to estimate the projectile velocity the results are very poor. For example, the actual downrange velocity is constant and is approximately 2000 ft/s. However, we can see from Fig. 9.3 that the estimated downrange velocity varies from 500 to 5000 ft/s. Similarly, Fig. 9.4 shows that the actual altitude velocity varies from 2000 ft/s at cannon ball launch to −2000 ft/s near impact. However, the figure also shows that the estimated altitude velocity is often in error by several thousand feet per second. Clearly, filtering is required if we desire a better way of estimating the projectile's velocity.

It might appear from Fig. 9.2 that the position estimates of the projectile were satisfactory because they appeared to be so close to the actual trajectory. We have seen in the earlier filtering examples of this text that we often compared the error in the estimates to the theoretical predictions of the covariance matrix. Although in this example there is no covariance matrix, we can look at the errors in the estimates of downrange and altitude. For this problem the initial downrange and altitude estimates of the projectile were in error by 1000 ft. We can see from Figs. 9.5 and 9.6 that the errors in the estimates of the projectile's downrange and altitude are never substantially below 1000 ft. In other words, we are not reducing

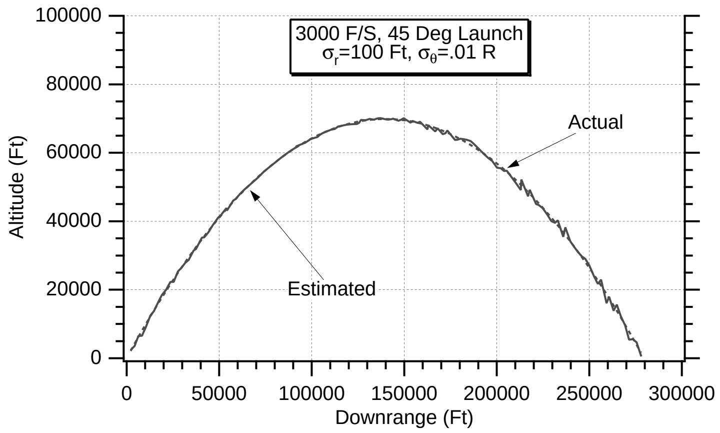

Fig. 9.2 Using raw measurements to estimate trajectory appears to be satisfactory.

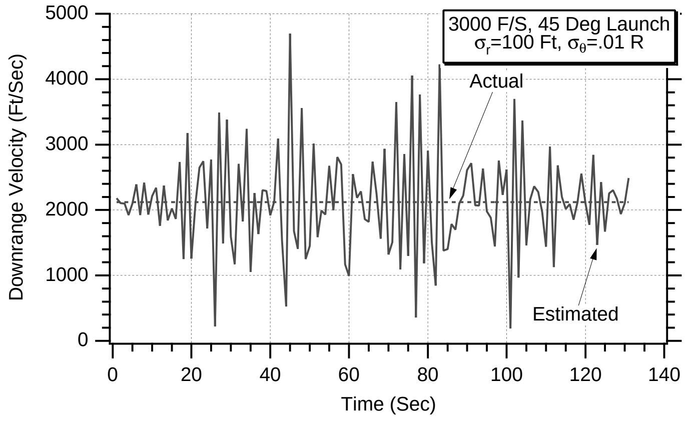

Fig. 9.3 Using raw measurements yields terrible downrange velocity estimates.

the initial errors. This is to be expected because we are not really doing any filtering but are simply using the raw measurements to obtain the location of the projectile.

Extended Cartesian Kalman Filter

If we desire to keep the model describing the real world linear in the cannon-launched projectile tracking problem, we can choose as states projectile location and velocity in the downrange or x direction and projectile location and velocity in the altitude or y direction. Thus, the proposed states are given by

$$\boldsymbol{x} = \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix}$$

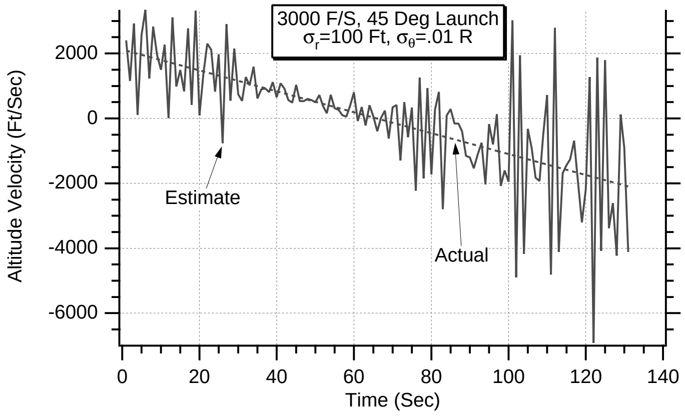

Fig. 9.4 Using raw measurements also yields very poor altitude velocity estimates.

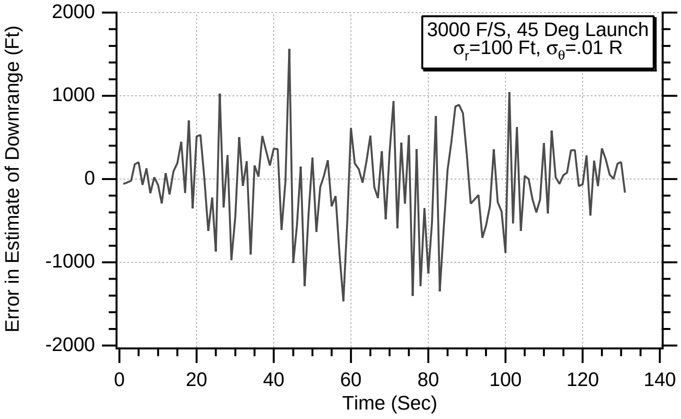

Fig. 9.5 Error in estimate of downrange is often greater than 1000 ft.

Therefore, when the preceding Cartesian states are chosen the state-space differential equation describing projectile motion becomes

$$\begin{bmatrix} \dot{x}_T \\ \ddot{x}_T \\ \dot{y}_T \\ \ddot{y}_T \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} + \left[\begin{bmatrix} 0 \\ u_s \\ 0 \\ u_s \end{bmatrix} \right]$$

Notice that in the preceding equation gravity g is not a state that has to be estimated but is assumed to be known in advance. We have also added process noise u_s to the acceleration portion of the equations as protection for effects that may not be considered by the Kalman filter. However, in our initial experiments

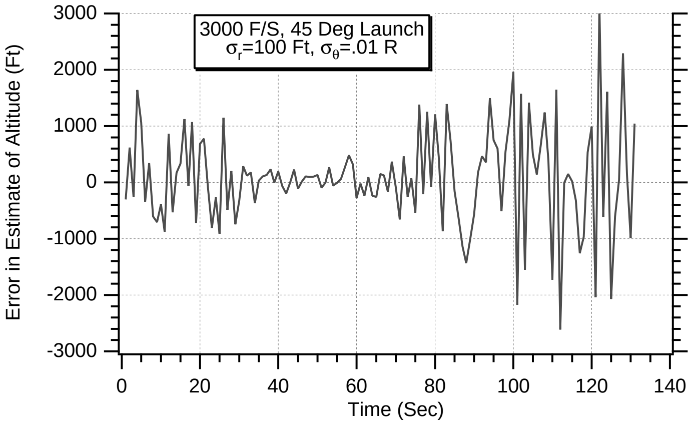

Fig. 9.6 Error in estimate of altitude is often greater than 1000 ft.

we will assume the process noise to be zero because our state-space equations upon which the Kalman filter is based will be a perfect match to the real world.

From the preceding state-space equation we can see that systems dynamics matrix is given by

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Because the fundamental matrix for a time-invariant system is given by

$$\Phi(\boldsymbol{t}) = \mathscr{L}^{-1}[(\boldsymbol{sI} - \boldsymbol{F})^{-1}]$$

we could find the fundamental matrix from the preceding expression. However, the required inversion of a 4 × 4 matrix might prove very tedious. A faster way to compute the fundamental matrix in this example is to simply use the Taylor-series approximation, which is given by

$$\Phi(t) = \boldsymbol{I} + \boldsymbol{F}t + \frac{\boldsymbol{F}^2 t^2}{2!} + \frac{\boldsymbol{F}^3 t^3}{3!} + \cdots$$

Because

$$\boldsymbol{F}^2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

all of the higher-order terms of the Taylor-series expansion must be zero, and the fundamental matrix becomes

$$\Phi(t) = \boldsymbol{I} + \boldsymbol{F}t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} t$$

or, more simply,

$$\Phi(t) = \begin{bmatrix} 0 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore, the discrete fundamental matrix can be found by substituting the sampling time T_s for time t and is given by

$$\Phi_k = \begin{bmatrix} 0 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Because our states have been chosen to be Cartesian, the radar measurements r and θ will automatically be nonlinear functions of those states. Therefore, we must write the linearized measurement equation as

$$\begin{bmatrix} \Delta\theta^* \\ \Delta r^* \end{bmatrix} = \begin{bmatrix} \dfrac{\partial\theta}{\partial x_T} & \dfrac{\partial\theta}{\partial \dot{x}_T} & \dfrac{\partial\theta}{\partial y_T} & \dfrac{\partial\theta}{\partial \dot{y}_T} \\ \dfrac{\partial r}{\partial x_T} & \dfrac{\partial r}{\partial \dot{x}_T} & \dfrac{\partial r}{\partial y_T} & \dfrac{\partial r}{\partial \dot{y}_T} \end{bmatrix} \begin{bmatrix} \Delta x_T \\ \Delta \dot{x}_T \\ \Delta y_T \\ \Delta \dot{y}_T \end{bmatrix} + \begin{bmatrix} v_\theta \\ v_r \end{bmatrix}$$

where v_θ and v_r represent the measurement noise on angle and range, respectively. Because the angle from the radar to the projectile is given by

$$\theta = \tan^{-1}\left(\frac{y_T - y_R}{x_T - x_R}\right)$$

the four partial derivatives of the angle with respect to each of the states can be computed as

$$\frac{\partial\theta}{\partial x_T} = \frac{1}{1 + [(y_T - y_R)^2/(x_T - x_R)^2]} \frac{(x_T - x_R)^*0 - (y_T - y_R)^*1}{(x_T - x_R)^2} = \frac{-(y_T - y_R)}{r^2}$$

$$\frac{\partial\theta}{\partial \dot{x}_T} = 0$$

$$\frac{\partial\theta}{\partial y_T} = \frac{1}{1 + [(y_T - y_R)^2/(x_T - x_R)^2]} \frac{(x_T - x_R)^*1 - (y_T - y_R)^*0}{(x_T - x_R)^2} = \frac{(x_T - x_R)}{r^2}$$

$$\frac{\partial\theta}{\partial \dot{y}_T} = 0$$

Similarly, because the range from the radar to the projectile is given by

$$r = \sqrt{(x_T - x_R)^2 + (y_T - y_R)^2}$$

the four partial derivatives of the range with respect to each of the states can be computed as

$$\frac{\partial r}{\partial x_T} = \frac{1}{2}[(x_T - x_R)^2 + (y_T - y_R)^2]^{-1/2} 2(x_T - x_R) = \frac{(x_T - x_R)}{r}$$

$$\frac{\partial r}{\partial \dot{x}_T} = 0$$

$$\frac{\partial r}{\partial y_T} = \frac{1}{2}[(x_T - x_R)^2 + (y_T - y_R)^2]^{-1/2} 2(y_T - y_R) = \frac{(y_T - y_R)}{r}$$

$$\frac{\partial r}{\partial \dot{y}_T} = 0$$

Because the linearized measurement matrix is given by

$$\boldsymbol{H} = \begin{bmatrix} \dfrac{\partial\theta}{\partial x_T} & \dfrac{\partial\theta}{\partial \dot{x}_T} & \dfrac{\partial\theta}{\partial y_T} & \dfrac{\partial\theta}{\partial \dot{y}_T} \\ \dfrac{\partial r}{\partial x_T} & \dfrac{\partial r}{\partial \dot{x}_T} & \dfrac{\partial r}{\partial y_T} & \dfrac{\partial r}{\partial \dot{y}_T} \end{bmatrix}$$

we can use substitution of the already derived partial derivatives to yield the linearized measurement matrix for this example to be

$$\boldsymbol{H} = \begin{bmatrix} \dfrac{-(y_T - y_R)}{r^2} & 0 & \dfrac{x_T - x_R}{r^2} & 0 \\ \dfrac{x_T - x_R}{r} & 0 & \dfrac{y_T - y_R}{r} & 0 \end{bmatrix}$$

For this problem it is assumed that we know where the radar is so that x_R and y_R are known and do not have to be estimated. The states required for the discrete linearized measurement matrix will be based on the projected state estimate or

$$\boldsymbol{H}_k = \begin{bmatrix} \dfrac{-(\bar{y}_{T_k} - y_R)}{\bar{r}_k^2} & 0 & \dfrac{\bar{x}_{T_k} - x_R}{\bar{r}^2} & 0 \\ \dfrac{\bar{x}_{T_k} - x_R}{\bar{r}_k} & 0 & \dfrac{\bar{y}_{T_k} - y_R}{\bar{r}_k} & 0 \end{bmatrix}$$

Because the discrete measurement noise matrix is given by

$$\boldsymbol{R}_k = E(\boldsymbol{v}_k \boldsymbol{v}_k^T)$$

we can say that for this problem the discrete measurement noise matrix is

$$\boldsymbol{R}_k = \begin{bmatrix} \sigma_\theta^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix}$$

where σ_θ^2 and σ_r^2 are the variances of the angle noise and range noise measurements, respectively. Similarly because the continuous process-noise matrix is given by

$$\boldsymbol{Q} = E(\boldsymbol{w}\boldsymbol{w}^T)$$

we can easily show from the linear state-space equation for this example that the continuous process-noise matrix is

$$\boldsymbol{Q}(t) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \Phi_s & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Phi_s \end{bmatrix}$$

where Φ_s is the spectral density of the white noise sources assumed to be on the downrange and altitude accelerations acting on the projectile. The discrete process-noise matrix can be derived from the continuous process-noise matrix according to

$$\boldsymbol{Q}_k = \int_0^{T_s} \Phi(\tau)\boldsymbol{Q}\Phi^T(\tau)\,\mathrm{d}t$$

Therefore, substitution of the appropriate matrices into the preceding expression yields

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} 1 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \Phi_s & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Phi_s \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ \tau & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \tau & 1 \end{bmatrix} \mathrm{d}\tau$$

After some algebra we get

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} \tau^2\Phi_s & \tau\Phi_s & 0 & 0 \\ \tau\Phi_s & \Phi_s & 0 & 0 \\ 0 & 0 & \tau^2\Phi_s & \tau\Phi_s \\ 0 & 0 & \tau\Phi_s & \Phi_s \end{bmatrix} \mathrm{d}\tau$$

Finally, after integration we obtain the final expression for the discrete process-noise matrix to be

$$\boldsymbol{Q}_k = \begin{bmatrix} \frac{T_s^3\Phi_s}{3} & \frac{T_s^2\Phi_s}{2} & 0 & 0 \\ \frac{T_s^2\Phi_s}{2} & T_s\Phi_s & 0 & 0 \\ 0 & 0 & \frac{T_s^3\Phi_s}{2} & \frac{T_s^2\Phi_s}{2} \\ 0 & 0 & \frac{T_s^2\Phi_s}{2} & T_s\Phi_s \end{bmatrix}$$

We now have defined all of the matrices required to solve the Riccati equations. The next step is to write down the equations for the Kalman-filter section. First, we must be able to propagate the states from the present sampling time to the next sampling time. This can be done in this example precisely with the fundamental matrix because the fundamental matrix is exact. In other nonlinear examples, where the fundamental matrix was approximate, we have already shown that the numerical integration of the nonlinear differential equations until the next update (i.e., using Euler integration) was required to eliminate potential filtering problems (i.e., hangoff error or even filter divergence).

Recall that for the linear filtering problem the real world was represented by the state-space equation

$$\dot{\boldsymbol{x}} = \boldsymbol{F}\boldsymbol{x} + \boldsymbol{G}\boldsymbol{u} + \boldsymbol{w}$$

where $\boldsymbol{G}$ is a matrix multiplying a known disturbance or control vector $\boldsymbol{u}$ that does not have to be estimated. We can show that the discrete linear Kalman-filtering equation is given by

$$\hat{\boldsymbol{x}}_k = \Phi_k\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{G}_k\boldsymbol{u}_{k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}\Phi_k\hat{x}_{k-1} - \boldsymbol{H}\boldsymbol{G}_k\boldsymbol{u}_{k-1})$$

where $\boldsymbol{G}_k$ is obtained from

$$\boldsymbol{G}_k = \int_0^{T_s} \Phi(\tau)\boldsymbol{G}\,\mathrm{d}\tau$$

If we forget about the gain times the residual portion of the filtering equation, we can see that the projected state is simply

$$\bar{\boldsymbol{x}}_k = \Phi_k\hat{\mathbf{x}}_{k-1} + \boldsymbol{G}_k\boldsymbol{u}_{k-1}$$

For this problem

$$\boldsymbol{G} = \boldsymbol{G}\boldsymbol{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix}$$

Therefore, the $\boldsymbol{G}_k$ becomes

$$\boldsymbol{G}_k = \int_0^{T_s} \begin{bmatrix} 1 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} \mathrm{d}\tau = \begin{bmatrix} 0 \\ 0 \\ -\dfrac{gT_s^2}{2} \\ -gT_s \end{bmatrix}$$

and our projected state is determined from

$$\bar{\boldsymbol{x}}_k = \begin{bmatrix} 1 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix} \hat{\boldsymbol{x}}_{k-1} + \begin{bmatrix} 0 \\ 0 \\ -\dfrac{gT_s^2}{2} \\ -gT_s \end{bmatrix}$$

When we convert the preceding matrix equation for the projected states to four scalar equations, we obtain

$$\bar{x}_{T_k} = \hat{x}_{T_{k-1}} + T_s \hat{\dot{x}}_{T_{k-1}}$$

$$\overline{\dot{x}_{T_k}} = \hat{\dot{x}}_{T_{k-1}}$$

$$\bar{y}_{T_k} = \hat{y}_{T_{k-1}} + T_s \hat{\dot{y}}_{T_{k-1}} - 0.5gT_s^2$$

$$\overline{\dot{y}_{T_k}} = \hat{\dot{y}}_{T_{k-1}} - gT_s$$

The next portion of the Kalman filter uses gains times residuals. Because the measurements are nonlinear, the residuals are simply the measurements minus the projected values of the measurements (i.e., we do not want to use the linearized measurement matrix). Therefore, the projected value of the angle and range from the radar to the projectile must be based upon the projected state estimates or

$$\bar{\theta}_k = \tan^{-1}\left(\frac{\bar{y}_{T_{k-1}} - y_R}{\bar{x}_{T_{k-1}} - x_R}\right)$$

$$\bar{r}_k = \sqrt{(\bar{x}_{T_{k-1}} - x_R)^2 + (\bar{y}_{T_{k-1}} - y_R)^2}$$

Now the extended Kalman-filtering equations can be written simply as

$$\hat{x}_{T_k} = \bar{x}_{T_k} + K_{11_k}(\theta_k^* - \bar{\theta}_k) + K_{12_k}(r_k^* - \bar{r}_k)$$

$$\hat{\dot{x}}_{T_k} = \bar{\dot{x}}_{T_k} + K_{21_k}(\theta_k^* - \bar{\theta}_k) + K_{22_k}(r_k^* - \bar{r}_k)$$

$$\hat{y}_{T_k} = \bar{y}_{T_k} + K_{31_k}(\theta_k^* - \bar{\theta}_k) + K_{32_k}(r_k^* - \bar{r}_k)$$

$$\hat{\dot{y}}_{T_k} = \bar{\dot{y}}_{T_k} + K_{41_k}(\theta_k^* - \bar{\theta}_k) + K_{42_k}(r_k^* - \bar{r}_k)$$

where θ_k^* and r_k^* are the noisy measurements of radar angle and range. Again, notice that we are using the actual nonlinear measurement equations in the extended Kalman filter.

The preceding equations for the Kalman filter and Riccati equations were programmed and are shown in Listing 9.1, along with a simulation of the real world. We can see that the process-noise matrix is set to zero in this example (i.e., $\Phi_s = 0$). In addition, we have initialized the states of the filter close to the true values. The filter's position states are in error by 1000 ft, and the velocity states are in error by 100 ft/s. The initial covariance matrix reflects those errors. Also, because we have two independent measurements, the Riccati equation requires the inverse of a 2×2 matrix. This inverse is done exactly using the matrix inverse formula for a 2×2 matrix from Chapter 1.

The extended Kalman filter of Listing 9.1 was run for the nominal conditions described. Figure 9.7 compares the error in the estimate of the projectile's

Listing 9.1 Cartesian extended Kalman-filter simulation for projectile tracking problem

```
C THE FIRST THREE STATEMENTS INVOKE THE ABSOFT RANDOM
NUMBER GENERATOR ON THE MACINTOSH
      GLOBAL DEFINE
            INCLUDE 'quickdraw.inc'
      END
      IMPLICIT REAL*8(A-H,O-Z)
      REAL*8 P(4,4),Q(4,4),M(4,4),PHI(4,4),HMAT(2,4),HT(4,2),PHIT(4,4)
      REAL*8 RMAT(2,2),IDN(4,4),PHIP(4,4),PHIPPHIT(4,4),HM(2,4)
      REAL*8 HMHT(2,2),HMHTR(2,2),HMHTRINV(2,2),MHT(4,2),K(4,2)
      REAL*8 KH(4,4),IKH(4,4)
      INTEGER ORDER
      TS=1.
      ORDER=4
      PHIS=0.
      SIGTH=.01
      SIGR=100.
      VT=3000.
      GAMDEG=45.
      G=32.2
      XT=0.
      YT=0.
```

(*continued*)

Listing 9.1 (*Continued*)

```
        XTD=VT*COS(GAMDEG/57.3)
        YTD=VT*SIN(GAMDEG/57.3)
        XR=100000.
        YR=0.
        OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
        OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
        T=0.
        S=0.
        H=.001
        DO 14 I=1,ORDER
        DO 14 J=1,ORDER
        PHI(I,J)=0.
        P(I,J)=0.
        Q(I,J)=0.
        IDN(I,J)=0.
14      CONTINUE
        TS2=TS*TS
        TS3=TS2*TS
        Q(1,1)=PHIS*TS3/3.
        Q(1,2)=PHIS*TS2/2.
        Q(2,1)=Q(1,2)
        Q(2,2)=PHIS*TS
        Q(3,3)=Q(1,1)
        Q(3,4)=Q(1,2)
        Q(4,3)=Q(3,4)
        Q(4,4)=Q(2,2)
        PHI(1,1)=1.
        PHI(1,2)=TS
        PHI(2,2)=1.
        PHI(3,3)=1.
        PHI(3,4)=TS
        PHI(4,4)=1.
        CALL MATTRN(PHI,ORDER,ORDER,PHIT)
        RMAT(1,1)=SIGTH**2
        RMAT(1,2)=0.
        RMAT(2,1)=0.
        RMAT(2,2)=SIGR**2
        IDN(1,1)=1.
        IDN(2,2)=1.
        IDN(3,3)=1.
        IDN(4,4)=1.
        P(1,1)=1000.**2
        P(2,2)=100.**2
        P(3,3)=1000.**2
        P(4,4)=100.**2
        XTH=XT+1000.
        XTDH=XTD-100.
        YTH=YT-1000.
        YTDH=YTD+100.
```

(*continued*)

Listing 9.1 (*Continued*)

```
          WHILE(YT>=0.)
               XTOLD=XT
               XTDOLD=XTD
               YTOLD=YT
               YTDOLD=YTD
               XTDD=0.
               YTDD=-G
               XT=XT+H*XTD
               XTD=XTD+H*XTDD
               YT=YT+H*YTD
               YTD=YTD+H*YTDD
               T=T+H
               XTDD=0.
               YTDD=-G
               XT=.5*(XTOLD+XT+H*XTD)
               XTD=.5*(XTDOLD+XTD+H*XTDD)
               YT=.5*(YTOLD+YT+H*YTD)
               YTD=.5*(YTDOLD+YTD+H*YTDD)
               S=S+H
               IF(S>=(TS-.00001))THEN
                    S=0.
                    XTB=XTH+TS*XTDH
                    XTDB=XTDH
                    YTB=YTH+TS*YTDH-.5*G*TS*TS
                    YTDB=YTDH-G*TS
                    RTB=SQRT((XTB-XR)**2+(YTB-YR)**2)
                    HMAT(1,1)=-(YTB-YR)/RTB**2
                    HMAT(1,2)=0.
                    HMAT(1,3)=(XTB-XR)/RTB**2
                    HMAT(1,4)=0.
                    HMAT(2,1)=(XTB-XR)/RTB
                    HMAT(2,2)=0.
                    HMAT(2,3)=(YTB-YR)/RTB
                    HMAT(2,4)=0.
                    CALL MATTRN(HMAT,2,ORDER,HT)
                    CALL MATMUL(PHI,ORDER,ORDER,P,ORDER,
                      ORDER,PHIP)
                    CALL MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
     1                 ORDER,PHIPPHIT)
                    CALL MATADD(PHIPPHIT,ORDER,ORDER,Q,M)
                    CALL MATMUL(HMAT,2,ORDER,M,ORDER,
                      ORDER,HM)
                    CALL MATMUL(HM,2,ORDER,HT,ORDER,2,HMHT)
                    CALL MATADD(HMHT,ORDER,ORDER,RMAT,
                      HMHTR)
                    DET=HMHTR(1,1)*HMHTR(2,2)-HMHTR(1,2)*
                      HMHTR(2,1)
                    HMHTRINV(1,1)=HMHTR(2,2)/DET
                    HMHTRINV(1,2)=-HMHTR(1,2)/DET
```

(*continued*)

Listing 9.1 (*Continued*)

```
                    HMHTRINV(2,1)=-HMHTR(2,1)/DET
                    HMHTRINV(2,2)=HMHTR(1,1)/DET
                    CALL MATMUL(M,ORDER,ORDER,HT,ORDER,2,MHT)
                    CALL MATMUL(MHT,ORDER,2,HMHTRINV,2,2,K)
                    CALL MATMUL(K,ORDER,2,HMAT,2,ORDER,KH)
                    CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
                    CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
1                     ORDER,P)
                    CALL GAUSS(THETNOISE,SIGTH)
                    CALL GAUSS(RTNOISE,SIGR)
                    THET=ATAN2((YT-YR),(XT-XR))
                    RT=SQRT((XT-XR)**2+(YT-YR)**2)
                    THETMEAS=THET+THETNOISE
                    RTMEAS=RT+RTNOISE
                    THETB=ATAN2((YTB-YR),(XTB-XR))
                    RTB=SQRT((XTB-XR)**2+(YTB-YR)**2)
                    RES1=THETMEAS-THETB
                    RES2=RTMEAS-RTB
                    XTH=XTB+K(1,1)*RES1+K(1,2)*RES2
                    XTDH=XTDB+K(2,1)*RES1+K(2,2)*RES2
                    YTH=YTB+K(3,1)*RES1+K(3,2)*RES2
                    YTDH=YTDB+K(4,1)*RES1+K(4,2)*RES2
                    ERRX=XT-XTH
                    SP11=SQRT(P(1,1))
                    ERRXD=XTD-XTDH
                    SP22=SQRT(P(2,2))
                    ERRY=YT-YTH
                    SP33=SQRT(P(3,3))
                    ERRYD=YTD-YTDH
                    SP44=SQRT(P(4,4))
                    WRITE(9,*)T,XT,YT,THET*57.3,RT
                    WRITE(1,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
                    WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
1                     ERRY,SP33,-SP33,ERRYD,SP44,-SP44
                ENDIF
        END DO
        PAUSE
        CLOSE(1)
        END

C SUBROUTINE GAUSS IS SHOWN IN LISTING 1.8
C SUBROUTINE MATTRN IS SHOWN IN LISTING 1.3
C SUBROUTINE MATMUL IS SHOWN IN LISTING 1.4
C SUBROUTINE MATADD IS SHOWN IN LISTING 1.1
C SUBROUTINE MATSUB IS SHOWN IN LISTING 1.2
```

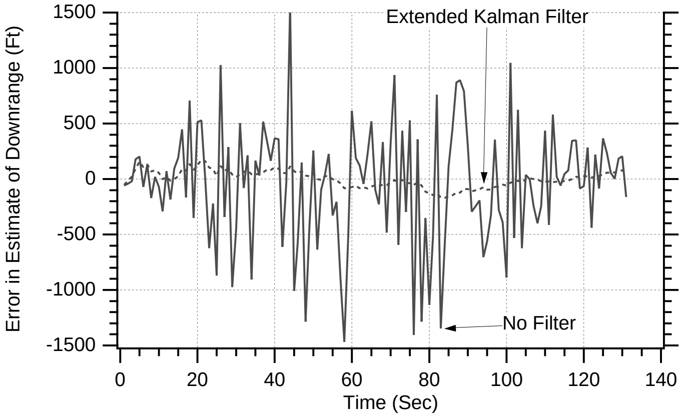

Fig. 9.7 Filtering dramatically reduces position error over using raw measurements.

downrange position for a single-run in which the extended Kalman filter is used in the preceding example in which there was no filtering at all (i.e., projectile's position reconstructed directly from range and angle measurements). The solid curve represents the case in which no filtering is used, whereas the dashed curve represents the error in the estimate when the extended Kalman filter is used. We can see that filtering dramatically reduces the error in the estimate of the projectile's position from more than 1000 ft to less than 100 ft.

To get a better intuitive feel for how the extended Kalman filter is performing, it is best to compare errors in the estimate of the projectile's position and velocity to the theoretical predictions of the covariance matrix. Figures 9.8 and 9.9 show how the single simulation run errors in the estimates of downrange position (i.e.,

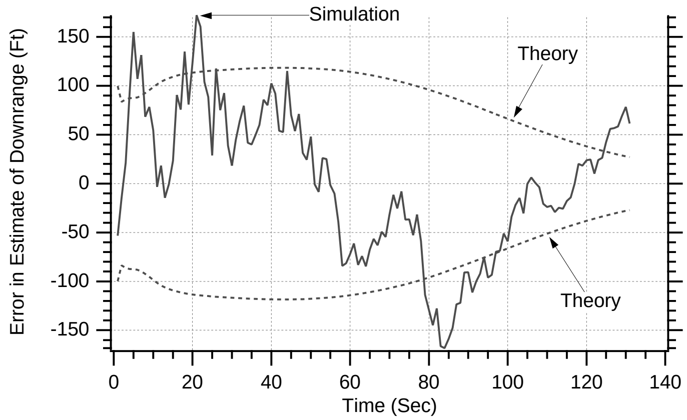

Fig. 9.8 Extended Cartesian Kalman filter's projectile's downrange estimates appear to agree with theory.

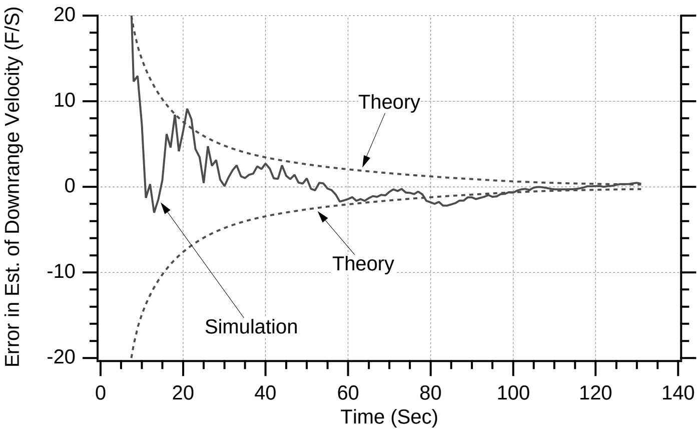

Fig. 9.9 Extended Cartesian Kalman filter's projectile's downrange velocity estimates appear to agree with theory.

x_T) and velocity (i.e., $\dot{x}_T$) compare with the theoretical predictions of the Riccati equation covariance matrix (i.e., square root of $P11$ and $P22$, respectively). We can see that the single flight simulation results lie within the theoretical bounds approximately 68% of the time, giving us a good feeling that the extended Cartesian Kalman filter is performing properly. Because there is no process noise in this example, the errors in the estimates continue to get smaller as more measurements are taken.

Similarly, Figs. 9.10 and 9.11 show how the single-run simulation errors in the estimates of the projectile's altitude (i.e., y_T) and altitude velocity (i.e., $\dot{y}_T$) compare with the theoretical predictions of the covariance matrix (i.e., square root of $P33$ and $P44$). Again, we can see that the single-run simulation results lie

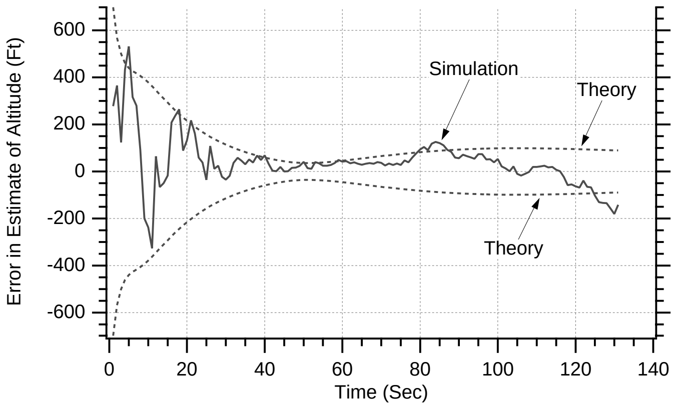

Fig. 9.10 Extended Cartesian Kalman filter's projectile's altitude estimates appear to agree with theory.

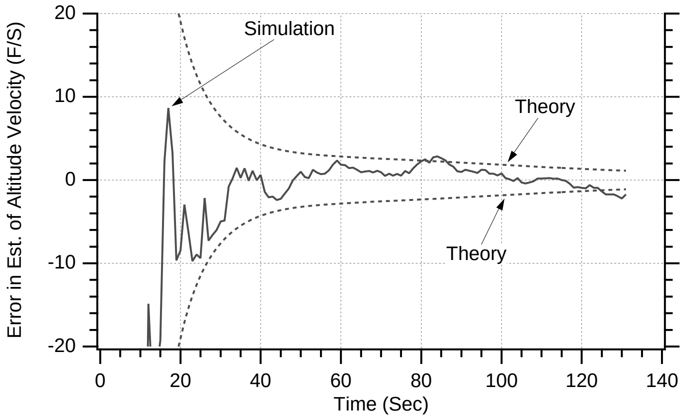

Fig. 9.11 Extended Cartesian Kalman filter's projectile's altitude velocity estimates appear to agree with theory.

within the theoretical bounds 68% of the time, giving us a good indication that the filter is performing properly.

In summary, we can say that for this example both the projectile's downrange and altitude position estimates appeared to be good to within 100 ft. The projectile's downrange and altitude velocity estimates appear to be good to within a few feet per second of the actual velocity. The results were for the case in which there was zero process noise. Adding process noise for practical reasons (i.e., prevent filter divergence when projectile does something that is not anticipated) will make the errors in the estimates larger. Recall that the process-noise matrix for the Cartesian extended Kalman filter of this example is given by

$$\boldsymbol{Q}_k = \begin{bmatrix} \frac{T_s^3\Phi_s}{3} & \frac{T_s^2\Phi_s}{2} & 0 & 0 \\ \frac{T_s^2\Phi_s}{2} & T_s\Phi_s & 0 & 0 \\ 0 & 0 & \frac{T_s^3\Phi_s}{3} & \frac{T_s^2\Phi_s}{2} \\ 0 & 0 & \frac{T_s^2\Phi_s}{2} & T_s\Phi_s \end{bmatrix}$$

Therefore, we can see that the amount of process noise is determined by Φ_s. Several runs were made in which the amount of process noise was varied, and the square root of the second diagonal element of the covariance matrix was saved. This covariance matrix element represents the theoretical error in the estimate of the projectile's downrange velocity. We can see from Fig. 9.12 that when there is no process noise (i.e., $\Phi_s = 0$) the projectile's velocity errors get smaller as time goes on (i.e., more measurements are taken). However, with process noise the

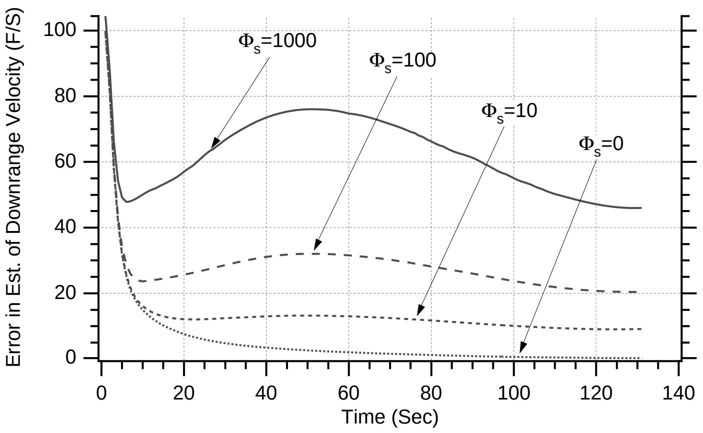

Fig. 9.12 Errors in estimate of velocity increase with increasing process noise.

error in the estimate of the projectile's downrange velocity approaches a steady state. In other words, errors in the estimate of the projectile's velocity do not decrease as more measurements are taken. We can see that the steady-state value of the error in the estimate of the projectile's velocity increases with increasing values of the process noise Φ_s.

Polar Coordinate System

In the preceding section we derived an extended Kalman filter, whose states were in a Cartesian coordinate system and whose measurements were in a polar coordinate system. With this methodology the model of the real world could easily be represented by linear differential equations, while the measurements were nonlinear functions of the states. In this section we will derive an extended Kalman filter based on a different set of states. These states will be for a polar coordinate system, which in turn will lead to measurements that are linear functions of the states. However, the resulting polar differential equations representing the real world will now be more complex and nonlinear. Eventually we want to see if an extended polar Kalman filter, based on this alternative representation, will perform better than the extended Cartesian Kalman filter. To accomplish this task, we must first derive and verify the polar equations for the cannon-launched projectile tracking problem.

Recall from Fig. 9.1 that the angle and range from the radar to the cannon ball is given by

$$\theta = \tan^{-1}\left(\frac{y_T - y_R}{x_T - x_R}\right)$$

$$r = \sqrt{(x_T - x_R)^2 + (y_T - y_R)^2}$$

Taking the first derivative of the angle yields

$$\dot{\theta} = \frac{1}{1 + (y_T - y_R)^2/(x_T - x_R)^2} \frac{(x_T - x_R)\dot{y}_T - (y_T - y_R)\dot{x}_T}{(x_T - x_R)^2}$$

After some simplification we can substitute the formula for range into the preceding expression to obtain

$$\dot{\theta} = \frac{(x_T - x_R)\dot{y}_T - (y_T - y_R)\dot{x}_T}{r^2}$$

Using calculus, we can also take the first derivative of range

$$\dot{r} = \frac{1}{2}[(x_T - x_R)^2 + (y_7 - y_R)^2]^{-1/2}[2(x_T - x_R)\dot{x}_T + 2(y_T - y_R)\dot{y}_T]$$

which simplifies to

$$\dot{r} = \frac{(x_T - x_R)\dot{x}_T + (y_T - y_R)\dot{y}_T}{r}$$

We can also find the angular acceleration by taking the derivative of the angular velocity or

$$\ddot{\theta} = \frac{r^2[\dot{x}_T\dot{y}_T + (x_T - x_R)\ddot{y}_t - \dot{x}_T\dot{y}_T - (y_T - y_R)\ddot{x}_T] - [(x_T - x_R)\dot{y}_T - (y_T - y_R)\dot{y}_T]2r\dot{r}}{r^4}$$

After some simplification we obtain

$$\ddot{\theta} = \frac{(x_T - x_R)\ddot{y}_T - (y_T - y_R)\ddot{x}_T - 2r\dot{r}\dot{\theta}}{r^2}$$

Recognizing that

$$\ddot{x}_T = 0$$

$$\ddot{y}_T = -g$$

$$\cos\theta = \frac{x_T - x_R}{r}$$

$$\sin\theta = \frac{y_T - y_R}{r}$$

the formula for angular acceleration simplifies even further to

$$\ddot{\theta} = \frac{-g\cos\theta - 2\dot{r}\dot{\theta}}{r}$$

Finally, the second derivative of range yields

$$\dot{r} = \frac{r[\dot{x}_T\dot{x}_T + (x_T - x_R)\ddot{x}_T + \dot{y}_T\dot{y}_T + (y_T - y_R)\ddot{y}_T] - [(x_T - x_R)\dot{x}_T + (y_T - y_R)\dot{y}_T]\dot{r}}{r^2}$$

After a great deal of algebraic manipulation and simplification, we obtain

$$\dot{r} = \frac{r^2\dot{\theta}^2 - gr\sin\theta}{r}$$

In summary, we are saying that the nonlinear polar differential equations representing the cannon-launched projectile are given by

$$\ddot{\theta} = \frac{-g\cos\theta - 2\dot{r}\dot{\theta}}{r}$$

$$\ddot{r} = \frac{r^2\dot{\theta}^2 - gr\sin\theta}{r}$$

The preceding two nonlinear differential equations are completely equivalent to the linear Cartesian differential equations

$$\ddot{x}_T = 0$$

$$\ddot{y}_T = -g$$

if they both have the same initial conditions. For example, if the initial conditions in the Cartesian equations are denoted

$$x_T(0), \qquad y_T(0), \qquad \dot{x}_T(0), \qquad \dot{y}_T(0)$$

then the initial conditions on the polar equations must be

$$\theta(0) = \tan^{-1}\left[\frac{y_T(0) - y_R}{x_T(0) - x_R}\right]$$

$$r(0) = \sqrt{[x_T(0) - x_R]^2 + [y_T(0) - y_R]^2}$$

$$\dot{\theta}(0) = \frac{[x_T(0) - x_R]\dot{y}_T(0) - [y_T(0) - y_R]\dot{x}_T(0)}{r(0)^2}$$

$$\dot{r}(0) = \frac{[x_T(0) - x_R]\dot{x}_T(0) + [y_T(0) - y_R]\dot{y}_T(0)}{r(0)}$$

When we have integrated the polar equations, we must also then find a way to convert the resultant answers back to Cartesian coordinates in order to perform a direct comparison with the Cartesian differential equations. The conversion can be easily accomplished by recognizing that

$$\hat{x}_T = r\cos\theta + x_R$$

$$\hat{y}_T = r\sin\theta + y_R$$

where the caret denotes the estimates based on the conversion. Taking the derivative of the preceding two equations yields the estimated velocity components, which are also based on the conversion or

$$\hat{\dot{x}}_T = \dot{r}\cos\theta - r\dot{\theta}\sin\theta$$

$$\hat{\dot{y}}_T = \dot{r}\sin\theta + r\dot{\theta}\cos\theta$$

The simulation that compares the flight of the cannon-launched projectile in both the polar and Cartesian coordinate systems appears in Listing 9.2. We can see that the simulation consists of both Cartesian and polar differential equations for the projectile. Both sets of differential equations have the same initial conditions and integration step size. The location and velocity of the projectile, based on the Cartesian and polar differential equations, are printed out every second.

The nominal case of Listing 9.2 was run, and we can see from Fig. 9.13 that the projectile locations for both the Cartesian and polar formulation of the

Listing 9.2 Simulation that compares polar and Cartesian differential equations of cannon-launched projectile

```
IMPLICIT REAL*8(A-H,O-Z)
VT=3000.
GAMDEG=45.
G=32.2
TS=1.
XT=0.
YT=0.
XTD=VT*COS(GAMDEG/57.3)
YTD=VT*SIN(GAMDEG/57.3)
XR=100000.
YR=0.
OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
T=0.
S=0.
H=.001
THET=ATAN2((YT-YR),(XT-XR))
RT=SQRT((XT-XR)**2+(YT-YR)**2)
THETD=((XT-XR)*YTD-(YT-YR)*XTD)/RT**2
RTD=((XT-XR)*XTD+(YT-YR)*YTD)/RT
WHILE(YT>=0.)
        XTOLD=XT
        XTDOLD=XTD
        YTOLD=YT
        YTDOLD=YTD
        THETOLD=THET
```

(*continued*)

Listing 9.2 (*Continued*)

```
          THETDOLD=THETD
          RTOLD=RT
          RTDOLD=RTD
          XTDD=0.
          YTDD=-G
          THETDD=(-G*COS(THET)-2.*RTD*THETD)/RT
          RTDD=((RT*THETD)**2-G*RT*SIN(THET))/RT
          XT=XT+H*XTD
          XTD=XTD+H*XTDD
          YT=YT+H*YTD
          YTD=YTD+H*YTDD
          THET=THET+H*THETD
          THETD=THETD+H*THETDD
          RT=RT+H*RTD
          RTD=RTD+H*RTDD
          T=T+H
          XTDD=0.
          YTDD=-G
          THETDD=(-G*COS(THET)-2.*RTD*THETD)/RT
          RTDD=((RT*THETD)**2-G*RT*SIN(THET))/RT
          XT=.5*(XTOLD+XT+H*XTD)
          XTD=.5*(XTDOLD+XTD+H*XTDD)
          YT=.5*(YTOLD+YT+H*YTD)
          YTD=.5*(YTDOLD+YTD+H*YTDD)
          THET=.5*(THETOLD+THET+H*THETD)
          THETD=.5*(THETDOLD+THETD+H*THETDD)
          RT=.5*(RTOLD+RT+H*RTD)
          RTD=.5*(RTDOLD+RTD+H*RTDD)
          S=S+H
          IF(S>=(TS-.00001))THEN
               S=0.
               XTH=RT*COS(THET)+XR
               YTH=RT*SIN(THET)+YR
               XTDH=RTD*COS(THET)-RT*SIN(THET)*THETD
               YTDH=RTD*SIN(THET)+RT*COS(THET)*THETD
               WRITE(9,*)T,XT,XTH,YT,YTH,XTD,XTDH,YTD,YTDH
               WRITE(1,*)T,XT,XTH,YT,YTH,XTD,XTDH,YTD,YTDH
          ENDIF
     END DO
     PAUSE
     CLOSE(1)
     END
```

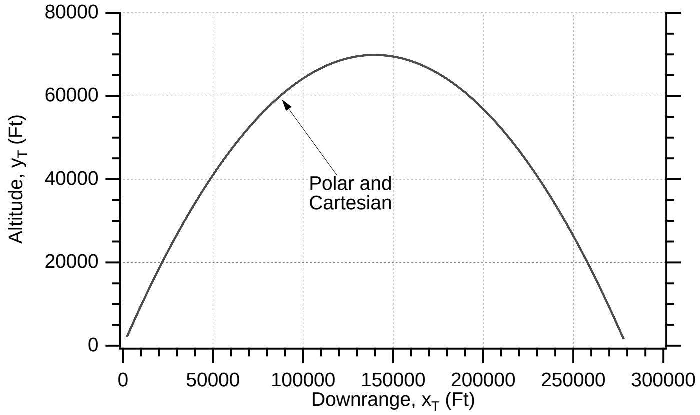

Fig. 9.13 Polar and Cartesian differential equations are identical in position.

differential equations are identical. In addition, Fig. 9.14 shows that the projectile component velocities are also identical in both coordinate systems, indicating the polar nonlinear differential equations have been derived correctly and are completely equivalent to the Cartesian linear differential equations.

Extended Polar Kalman Filter

We have already derived and validated the nonlinear polar differential equations for the cannon-launched projectile, which are given by

$$\ddot{\theta} = \frac{-g\cos\theta - 2\dot{r}\dot{\theta}}{r}$$

$$\ddot{r} = \frac{r^2\dot{\theta}^2 - gr\sin\theta}{r}$$

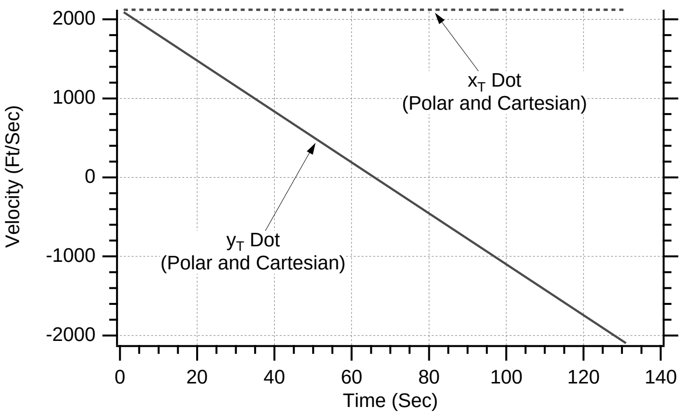

Fig. 9.14 Polar and Cartesian differential equations are identical in velocity.

In addition, we have already seen that, for this example, the extended Cartesian Kalman filter performed well without a process-noise matrix because the filter model of the real world was perfect. Let us assume, for purposes of comparison, that we also can neglect the process noise in polar coordinates. Under these circumstances we can express the nonlinear polar differential equations describing projectile motion in state-space form as

$$\begin{bmatrix} \Delta\dot{\theta} \\ \Delta\ddot{\theta} \\ \Delta\dot{r} \\ \Delta\ddot{r} \end{bmatrix} = \begin{bmatrix} \frac{\partial\dot{\theta}}{\partial\theta} & \frac{\partial\dot{\theta}}{\partial\dot{\theta}} & \frac{\partial\dot{\theta}}{\partial r} & \frac{\partial\dot{\theta}}{\partial\dot{r}} \\ \frac{\partial\ddot{\theta}}{\partial\theta} & \frac{\partial\ddot{\theta}}{\partial\dot{\theta}} & \frac{\partial\ddot{\theta}}{\partial r} & \frac{\partial\ddot{\theta}}{\partial\dot{r}} \\ \frac{\partial\dot{r}}{\partial\theta} & \frac{\partial\dot{r}}{\partial\dot{\theta}} & \frac{\partial\dot{r}}{\partial r} & \frac{\partial\dot{r}}{\partial\dot{r}} \\ \frac{\partial\ddot{r}}{\partial\theta} & \frac{\partial\ddot{r}}{\partial\dot{\theta}} & \frac{\partial\ddot{r}}{\partial r} & \frac{\partial\ddot{r}}{\partial\dot{r}} \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta\dot{\theta} \\ \Delta r \\ \Delta\dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{\partial\ddot{\theta}}{\partial\theta} & \frac{\partial\ddot{\theta}}{\partial\dot{\theta}} & \frac{\partial\ddot{\theta}}{\partial r} & \frac{\partial\ddot{\theta}}{\partial\dot{r}} \\ 0 & 0 & 0 & 1 \\ \frac{\partial\ddot{r}}{\partial\theta} & \frac{\partial\ddot{r}}{\partial\dot{\theta}} & \frac{\partial\ddot{r}}{\partial r} & \frac{\partial\ddot{r}}{\partial\dot{r}} \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta\dot{\theta} \\ \Delta r \\ \Delta\dot{r} \end{bmatrix}$$

In this case the system dynamics matrix, which is the preceding matrix of partial derivatives, can be written by inspection as

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{g\sin\theta}{r} & \frac{-2\dot{r}}{r} & \frac{g\cos\theta + 2\dot{\theta}\dot{r}}{r^2} & \frac{-2\dot{\theta}}{r} \\ 0 & 0 & 0 & 1 \\ -g\cos\theta & 2r\dot{\theta} & \dot{\theta}^2 & 0 \end{bmatrix}$$

Therefore, if we use a two-term Taylor-series approximation for the fundamental matrix, we get

$$\Phi(t) = \boldsymbol{I} + \boldsymbol{F}t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & t & 0 & 0 \\ \frac{g\sin\theta}{r}t & \frac{-2\dot{r}}{r}t & \frac{g\cos\theta + 2\dot{\theta}\dot{r}}{r^2}t & \frac{-2\dot{\theta}}{r}t \\ 0 & 0 & 0 & t \\ -g\cos\theta t & 2r\dot{\theta}t & \dot{\theta}^2 t & 0 \end{bmatrix}$$

After some simplification we finally obtain

$$\Phi(t) = \begin{bmatrix} 1 & t & 0 & 0 \\ \frac{g\sin\theta}{r}t & 1 - \frac{2\dot{r}}{r}t & \frac{g\cos\theta + 2\dot{\theta}\dot{r}}{r^2}t & \frac{-2\dot{\theta}}{r}t \\ 0 & 0 & 1 & t \\ -g\cos\theta t & 2r\dot{\theta}t & \dot{\theta}^2 t & 1 \end{bmatrix}$$

The discrete fundamental matrix can be found by substituting T_s for t in the preceding expression, thus obtaining

$$\Phi_k = \begin{bmatrix} 1 & T_s & 0 & 0 \\ \frac{g\sin\theta}{r}T_s & 1-\frac{2\dot{r}}{r}T_s & \frac{g\cos\theta + 2\dot{\theta}\dot{r}}{r^2}T_s & \frac{-2\dot{\theta}}{r}T_s \\ 0 & 0 & 1 & T_s \\ -g\cos\theta T_s & 2r\dot{\theta}T_s & \dot{\theta}^2 T_s & 1 \end{bmatrix}$$

Because the states are polar, the measurement equation is linear and can be expressed as

$$\begin{bmatrix} \theta^* \\ r^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ r \\ \dot{r} \end{bmatrix} + \begin{bmatrix} v_\theta \\ v_r \end{bmatrix}$$

where the measurement matrix is

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Because the discrete measurement noise matrix is given by

$$\boldsymbol{R}_k = E(\boldsymbol{v}_k \boldsymbol{v}_k^T)$$

we can say that for this problem the discrete measurement noise matrix turns out to be

$$\boldsymbol{R}_k = \begin{bmatrix} \sigma_\theta^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix}$$

where σ_θ^2 and σ_r^2 are the variances of the angle noise and range noise measurements, respectively.

Because we are assuming zero process noise, we now have defined all of the matrices required to solve the Riccati equations. However, if we want to compare the resultant polar covariance matrices obtained from the Riccati equations with the preceding Cartesian covariance matrices, then we must have a way of going between both systems. Recall that we have shown in the preceding section that the four equations relating the polar description of the projectile to the Cartesian description of the projectile are given by

$$\begin{aligned}
x_T &= r\cos\theta + x_R \\
y_T &= r\sin\theta + y_R \\
\dot{x}_T &= \dot{r}\cos\theta - r\dot{\theta}\sin\theta \\
\dot{y}_T &= \dot{r}\sin\theta + r\dot{\theta}\cos\theta
\end{aligned}$$

We can use the chain rule from calculus to get the total differentials

$$\begin{aligned}
\Delta x_T &= \frac{\partial x_T}{\partial\theta}\Delta\theta + \frac{\partial x_T}{\partial\dot{\theta}}\Delta\dot{\theta} + \frac{\partial x_T}{\partial r}\Delta r + \frac{\partial x_T}{\partial\dot{r}}\Delta\dot{r} \\
\Delta\dot{x}_T &= \frac{\partial\dot{x}_T}{\partial\theta}\Delta\theta + \frac{\partial\dot{x}_T}{\partial\dot{\theta}}\Delta\dot{\theta} + \frac{\partial\dot{x}_T}{\partial r}\Delta r + \frac{\partial\dot{x}_T}{\partial\dot{r}}\Delta\dot{r} \\
\Delta y_T &= \frac{\partial y_T}{\partial\theta}\Delta\theta + \frac{\partial y_T}{\partial\dot{\theta}}\Delta\dot{\theta} + \frac{\partial y_T}{\partial r}\Delta r + \frac{\partial y_T}{\partial\dot{r}}\Delta\dot{r} \\
\Delta\dot{y}_T &= \frac{\partial\dot{y}_T}{\partial\theta}\Delta\theta + \frac{\partial\dot{y}_T}{\partial\dot{\theta}}\Delta\dot{\theta} + \frac{\partial\dot{y}_T}{\partial r}\Delta r + \frac{\partial\dot{y}_T}{\partial\dot{r}}\Delta\dot{r}
\end{aligned}$$

Evaluating each of the partial derivatives yields the following expressions for each of the total differentials:

$$\begin{gathered}
\Delta x_T = -r\sin\theta\Delta\theta + \cos\theta\Delta r \\
\Delta\dot{x}_T = (-\dot{r}\sin\theta - r\dot{\theta}\cos\theta)\Delta\theta - r\sin\theta\Delta\dot{\theta} - \dot{\theta}\sin\theta\Delta r + \cos\theta\Delta\dot{r} \\
\Delta y_T = r\cos\theta\Delta\theta + \sin\theta\Delta r \\
\Delta\dot{y}_T = (\dot{r}\cos\theta - r\dot{\theta}\sin\theta)\Delta\theta + r\cos\theta\Delta\dot{\theta} + \dot{\theta}\cos\theta\Delta r + \sin\theta\Delta\dot{r}
\end{gathered}$$

Placing the preceding set of equations into state-space form yields a more compact expression for the total differential

$$\begin{bmatrix} \Delta x_T \\ \Delta\dot{x}_T \\ \Delta y_T \\ \Delta\dot{y}_T \end{bmatrix} = \begin{bmatrix} -r\sin\theta & 0 & \cos\theta & 0 \\ -\dot{r}\sin\theta - r\dot{\theta}\cos\theta & -r\sin\theta & -\dot{\theta}\sin\theta & \cos\theta \\ r\cos\theta & 0 & \sin\theta & 0 \\ \dot{r}\cos\theta & r\cos\theta & \dot{\theta}\cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta\dot{\theta} \\ \Delta r \\ \Delta\dot{r} \end{bmatrix}$$

If we define the transformation matrix $\boldsymbol{A}$ relating the polar total differentials to the Cartesian total differentials, we get

$$\boldsymbol{A} = \begin{bmatrix} -r\sin\theta & 0 & \cos\theta & 0 \\ -\dot{r}\sin\theta - r\dot{\theta}\cos\theta & -r\sin\theta & -\dot{\theta}\sin\theta & \cos\theta \\ r\cos\theta & 0 & \sin\theta & 0 \\ \dot{r}\cos\theta & r\cos\theta & \dot{\theta}\cos\theta & \sin\theta \end{bmatrix}$$

It is easy to show that the Cartesian covariance matrix $\boldsymbol{P}_{\text{CART}}$ is related to the polar covariance matrix $\boldsymbol{P}_{\text{POL}}$ according to

$$\boldsymbol{P}_{CART} = \boldsymbol{A}\boldsymbol{P}_{POL}\boldsymbol{A}^T$$

The next step is to write down the equations for the Kalman-filter section. First, we must be able to propagate the states from the present sampling time to the next sampling time. This could be done exactly with the fundamental matrix when we were working in the Cartesian system because the fundamental matrix was exact. In the polar system the fundamental matrix is approximate, and so it is best to numerically integrate the nonlinear differential equations for a sampling interval to get the projected states. Now the Kalman-filtering equations can be written as

$$\hat{\theta}_k = \bar{\theta}_k + K_{11_k}(\theta_k^* - \bar{\theta}_k) + K_{12_k}(r_k^* - \bar{r}_k)$$

$$\hat{\dot{\theta}}_k = \bar{\dot{\theta}}_k + K_{21_k}(\theta_k^* - \bar{\theta}_k) + K_{22_k}(r_k^* - \bar{r}_k)$$

$$\hat{r}_k = \bar{r}_k + K_{31_k}(\theta_k^* - \bar{\theta}_k) + K_{32_k}(r_k^* - \bar{r}_k)$$

$$\hat{\dot{r}}_k = \bar{\dot{r}}_k + K_{41_k}(\theta_k^* - \bar{\theta}_k) + K_{42_k}(r_k^* - \bar{r}_k)$$

where θ_k^* and r_k^* are the noisy measurements of radar angle and range and the bar indicates the projected states that are obtained directly by numerical integration.

The preceding equations for the polar Kalman filter and Riccati equations were programmed and are shown in Listing 9.3, along with a simulation of the real world. There is no process-noise matrix in this formulation. We have initialized the states of the polar filter to have the same initialization of the Cartesian filter. The initial covariance matrix reflects the initial state estimate errors. The subroutine PROJECT handles the numerical integration of the nonlinear polar differential equations to get the projected value of the states T_s seconds in the future.

The nominal case of Listing 9.3 was run, and the polar and Cartesian errors in the estimates were computed. By comparing Fig. 9.15 with Fig. 9.8, we can see that the errors in the estimates of downrange are comparable for both filters. Initially, the error in the estimate of x is 100 ft, and after 130 s the error reduces to approximately 25 ft. Figure 9.15 also shows that the single-run simulation error in the estimate of x appears to be correct when compared to the theoretical

Listing 9.3 Polar extended Kalman filter simulation for projectile tracking problem

```
C THE FIRST THREE STATEMENTS INVOKE THE ABSOFT RANDOM
  NUMBER GENERATOR ON THE MACINTOSH
        GLOBAL DEFINE
                INCLUDE ‘quickdraw.inc’
        END
        IMPLICIT REAL*8 (A-H)
        IMPLICIT REAL*8 (O-Z)
        REAL*8 P(4,4),M(4,4),GAIN(4,2),PHI(4,4),PHIT(4,4),PHIP(4,4)
        REAL*8 Q(4,4),HMAT(2,4),HM(2,4),MHT(4,2),PHIPPHIT(4,4),AT(4,4)
        REAL*8 RMAT(2,2),HMHTR(2,2),HMHTRINV(2,2),A(4,4),PNEW(4,4)
        REAL*8 HMHT(2,2),HT(4,2),KH(4,4),IDN(4,4),IKH(4,4),FTS(4,4)
        REAL*8 AP(4,4)
        INTEGER ORDER
 106    CONTINUE
        OPEN(1,STATUS=‘UNKNOWN’,FILE=‘DATFIL’)
        OPEN(2,STATUS=‘UNKNOWN’,FILE=‘COVFIL’)
        OPEN(3,STATUS=‘UNKNOWN’,FILE=‘COVFIL2’)
        SIGTH=.01
        SIGR=100.
        TS=1.
        G=32.2
        VT=3000.
        GAMDEG=45.
        XT=0.
        YT=0.
        XTD=VT*COS(GAMDEG/57.3)
        YTD=VT*SIN(GAMDEG/57.3)
        XR=100000.
        YR=0.
        XTH=XT+1000.
        YTH=YT-1000.
        XTDH=XTD-100.
        YTDH=YTD+100.
        TH=ATAN2((YT-YR),(XT-XR)+.001)
        R=SQRT((XT-XR)**2+(YT-YR)**2)
        THD=((XT-XR)*YTD-(YT-YR)*XTD)/R**2
        RD=((XT-XR)*XTD+(YT-YR)*YTD)/R
        THH=ATAN2((YTH-YR),(XTH-XR)+.001)
        RH=SQRT((XTH-XR)**2+(YTH-YR)**2)
        THDH=((XTH-XR)*YTDH-(YTH-YR)*XTDH)/RH**2
        RDH=((XTH-XR)*XTDH+(YTH-YR)*YTDH)/RH
        ORDER=4
        TF=100.
        T=0.
        S=0.
        H=.001
        HP=.001
        DO 1000 I=1,ORDER
        DO 1000 J=1,ORDER
```

(*continued*)

Listing 9.3 (*Continued*)

```
                    P(I,J)=0.
                    Q(I,J)=0.
                    IDN(I,J)=0.
                    PHI(I,J)=0.
                    FTS(I,J)=0.
                    A(I,J)=0.
 1000       CONTINUE
            IDN(1,1)=1.
            IDN(2,2)=1.
            IDN(3,3)=1.
            IDN(4,4)=1.
            P(1,1)=(TH-THH)**2
            P(2,2)=(THD-THDH)**2
            P(3,3)=(R-RH)**2
            P(4,4)=(RD-RDH)**2
            RMAT(1,1)=SIGTH**2
            RMAT(1,2)=0.
            RMAT(2,1)=0.
            RMAT(2,2)=SIGR**2
            HMAT(1,1)=1.
            HMAT(1,2)=0.
            HMAT(1,3)=0.
            HMAT(1,4)=0.
            HMAT(2,1)=0.
            HMAT(2,2)=0.
            HMAT(2,3)=1.
            HMAT(2,4)=0.
            WHILE(YT>=0.)
                  THOLD=TH
                  THDOLD=THD
                  ROLD=R
                  RDOLD=RD
                  THDD=(-G*R*COS(TH)-2.*THD*R*RD)/R**2
                  RDD=(R*R*THD*THD-G*R*SIN(TH))/R
                  TH=TH+H*THD
                  THD=THD+H*THDD
                  R=R+H*RD
                  RD=RD+H*RDD
                  T=T+H
                  THDD=(-G*R*COS(TH)-2.*THD*R*RD)/R**2
                  RDD=(R*R*THD*THD-G*R*SIN(TH))/R
                  TH=.5*(THOLD+TH+H*THD)
                  THD=.5*(THDOLD+THD+H*THDD)
                  R=.5*(ROLD+R+H*RD)
                  RD=.5*(RDOLD+RD+H*RDD)
                  S=S+H
                  IF(S>=(TS-.00001))THEN
                        S=0.
                        FTS(1,2)=1.*TS
                        FTS(2,1)=G*SIN(THH)*TS/RH
```

(*continued*)

Listing 9.3 (*Continued*)

```
FTS(2,2)=-2.*RDH*TS/RH
FTS(2,3)=(G*COS(THH)+2.*THDH*RDH)*TS/RH**2
FTS(2,4)=-2.*THDH*TS/RH
FTS(3,4)=1.*TS
FTS(4,1)=-G*COS(THH)*TS
FTS(4,2)=2.*RH*THDH*TS
FTS(4,3)=(THDH**2)*TS
CALL MATADD(FTS,ORDER,ORDER,IDN,PHI)
CALL MATTRN(HMAT,2,ORDER,HT)
CALL MATTRN(PHI,ORDER,ORDER,PHIT)
CALL MATMUL(PHI,ORDER,ORDER,P,ORDER,
  ORDER,PHIP)
CALL MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
  ORDER, PHIPPHIT)
CALL MATADD(PHIPPHIT,ORDER,ORDER,Q,M)
CALL MATMUL(HMAT,2,ORDER,M,ORDER,
  ORDER,HM)
CALL MATMUL(HM,2,ORDER,HT,ORDER,2,HMHT)
CALL MATADD(HMHT,2,2,RMAT,HMHTR)
DET=HMHTR(1,1)*HMHTR(2,2)-HMHTR(1,2)*
  HMHTR(2,1)
HMHTRINV(1,1)=HMHTR(2,2)/DET
HMHTRINV(1,2)=-HMHTR(1,2)/DET
HMHTRINV(2,1)=-HMHTR(2,1)/DET
HMHTRINV(2,2)=HMHTR(1,1)/DET
CALL MATMUL(M,ORDER,ORDER,HT,ORDER,
  2,MHT)
CALL MATMUL(MHT,ORDER,2,HMHTRINV,2,
  2,GAIN)
CALL MATMUL(GAIN,ORDER,2,HMAT,2,ORDER,KH)
CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
  ORDER,P)
CALL GAUSS(THNOISE,SIGTH)
CALL GAUSS(RNOISE,SIGR)
CALL PROJECT(T,TS,THH,THDH,RH,RDH,THB,
  THDB,RB,RDB,HP)
RES1=TH+THNOISE-THB
RES2=R+RNOISE-RB
THH=THB+GAIN(1,1)*RES1+GAIN(1,2)*RES2
THDH=THDB+GAIN(2,1)*RES1+GAIN(2,2)*RES2
RH=RB+GAIN(3,1)*RES1+GAIN(3,2)*RES2
RDH=RDB+GAIN(4,1)*RES1+GAIN(4,2)*RES2
ERRTH=TH-THH
SP11=SQRT(P(1,1))
ERRTHD=THD-THDH
SP22=SQRT(P(2,2))
ERRR=R-RH
```

(*continued*)

Listing 9.3 (*Continued*)

```
                        SP33=SQRT(P(3,3))
                        ERRRD=RD-RDH
                        SP44=SQRT(P(4,4))
                        XT=R*COS(TH)+XR
                        YT=R*SIN(TH)+YR
                        XTD=RD*COS(TH)-R*THD*SIN(TH)
                        YTD=RD*SIN(TH)+R*THD*COS(TH)
                        XTH=RH*COS(THH)+XR
                        YTH=RH*SIN(THH)+YR
                        XTDH=RDH*COS(THH)-RH*THDH*SIN(THH)
                        YTDH=RDH*SIN(THH)+RH*THDH*COS(THH)
                        A(1,1)=-RH*SIN(THH)
                        A(1,3)=COS(THH)
                        A(2,1)=-RDH*SIN(THH)-RH*THDH*COS(THH)
                        A(2,2)=-RH*SIN(THH)
                        A(2,3)=-THDH*SIN(THH)
                        A(2,4)=COS(THH)
                        A(3,1)=RH*COS(THH)
                        A(3,3)=SIN(THH)
                        A(4,1)=RDH*COS(THH)-RH*SIN(THH)*THDH
                        A(4,2)=RH*COS(THH)
                        A(4,3)=THDH*COS(THH)
                        A(4,4)=SIN(THH)
                        CALL  MATTRN(A,ORDER,ORDER,AT)
                        CALL  MATMUL(A,ORDER,ORDER,P,ORDER,
                          ORDER,AP)
                        CALL  MATMUL(AP,ORDER,ORDER,AT,ORDER,
                          ORDER,PNEW)
                        ERRXT=XT-XTH
                        SP11P=SQRT(PNEW(1,1))
                        ERRXTD=XTD-XTDH
                        SP22P=SQRT(PNEW(2,2))
                        ERRYT=YT-YTH
                        SP33P=SQRT(PNEW(3,3))
                        ERRYTD=YTD-YTDH
                        SP44P=SQRT(PNEW(4,4))
                        WRITE(9,*)T,R,RH,RD,RDH,TH,THH,THD,THDH
                        WRITE(1,*)T,R,RH,RD,RDH,TH,THH,THD,THDH
                        WRITE(2,*)T,ERRTH,SP11,-SP11,ERRTHD,SP22,-SP22,
  1                       ERRR,SP33,-SP33,ERRRD,SP44,-SP44
                        WRITE(3,*)T,ERRXT,SP11P,-SP11P,ERRXTD,SP22P,
  1                       -SP22P,ERRYT,SP33P,-SP33P,ERRYTD,
  2                       SP44P,-SP44P
                   ENDIF
      END  DO
      PAUSE
      CLOSE(1)
      CLOSE(2)
      CLOSE(3)
      END
```

(*continued*)

Listing 9.3 (*Continued*)

```
        SUBROUTINE PROJECT(TP,TS,THP,THDP,RP,RDP,THH,THDH,RH,
          RDH,HP)
        IMPLICIT REAL*8 (A-H)
        IMPLICIT REAL*8 (O-Z)
        T=0.
        G=32.2
        TH=THP
        THD=THDP
        R=RP
        RD=RDP
        H=HP
        WHILE(T<=(TS-.0001))
                THDD=(-G*R*COS(TH)-2.*THD*R*RD)/R**2
                RDD=(R*R*THD*THD-G*R*SIN(TH))/R
                THD=THD+H*THDD
                TH=TH+H*THD
                RD=RD+H*RDD
                R=R+H*RD
                T=T+H
        END DO
        RH=R
        RDH=RD
        THH=TH
        THDH=THD
        RETURN
        END

C SUBROUTINE GAUSS IS SHOWN IN LISTING 1.8
C SUBROUTINE MATTRN IS SHOWN IN LISTING 1.3
C SUBROUTINE MATMUL IS SHOWN IN LISTING 1.4
C SUBROUTINE MATADD IS SHOWN IN LISTING 1.1
C SUBROUTINE MATSUB IS SHOWN IN LISTING 1.2
C SUBROUTINE MATSCA IS SHOWN IN LISTING 1.5
```

predictions from the transformed covariance matrix in the sense that it appears to be within the theoretical error bounds approximately 68% of the time.

Figure 9.16 displays the error in the estimate of downrange velocity. By comparing Fig. 9.16 with Fig. 9.9, we can see that the errors in the estimate of downrange velocity are comparable for both the polar and Cartesian extended Kalman filters. Initially, the error in the estimate of velocity is approximately 20 ft/s at 10 s and diminishes to approximately 1 ft/s after 130 s. Figure 9.16 also shows that the single flight error in the estimate of $\dot{x}$ appears to be correct when compared to the theoretical predictions from the transformed covariance matrix.

Figure 9.17 displays the error in the estimate of altitude. By comparing Fig. 9.17 with Fig. 9.10, we can see that the errors in the estimate of altitude are comparable for both the polar and Cartesian extended Kalman filters. Initially, the

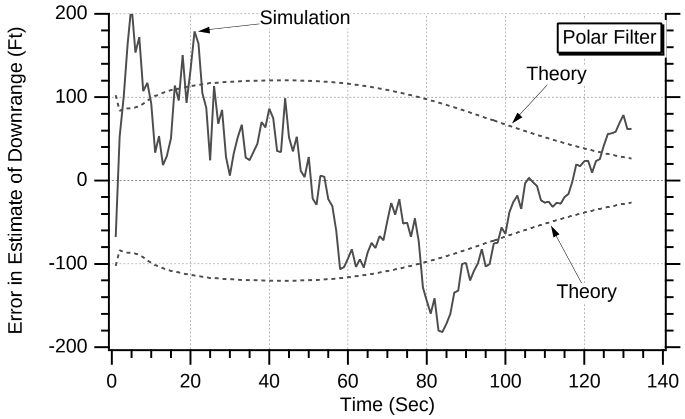

Fig. 9.15 Polar and Cartesian extended Kalman filters yield similar results for downrange estimates.

error in the estimate of altitude is approximately 600 ft and diminishes to approximately 100 ft after 130 s. Figure 9.17 also shows that the single-run simulation error in the estimate of y also appears to be correct when compared to the theoretical predictions from the transformed covariance matrix.

Figure 9.18 displays the error in the estimate of velocity in the altitude direction. By comparing Fig. 9.18 with Fig. 9.11, we can see that the errors in the estimate of the altitude velocity component are comparable for both the polar and Cartesian extended Kalman filters. Initially, the error in the estimate of altitude velocity is approximately 20 ft/s at 20 s and diminishes to approximately 2 ft/s after 130 s. Figure 9.18 also shows that the single-run simulation error in the estimate of $\dot{y}$ appears to be correct when compared to the theoretical predictions from the transformed covariance matrix.

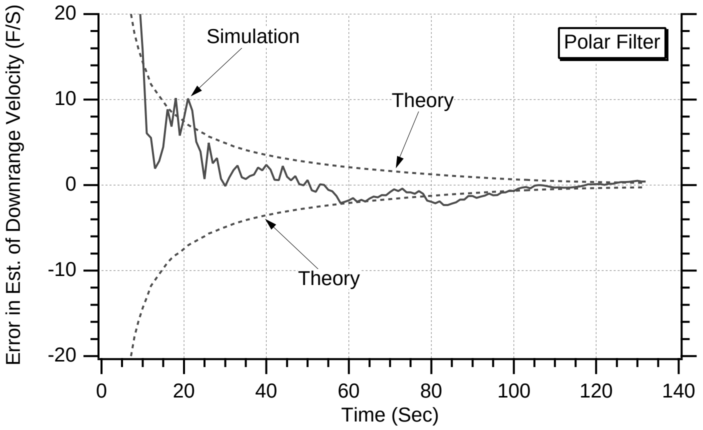

Fig. 9.16 Polar and Cartesian extended Kalman filters yield similar results for downrange velocity estimates.

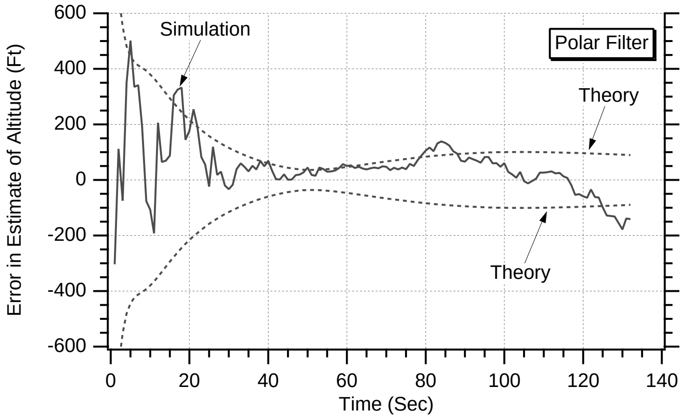

Fig. 9.17 Polar and Cartesian extended Kalman filters yield similar results for altitude estimates.

Recall that the polar filter states are angle, angle rate, distance, and distance rate or

$$\begin{bmatrix} \theta \\ \dot{\theta} \\ r \\ \dot{r} \end{bmatrix}$$

The covariance matrix resulting from the polar extended-Kalman-filter Riccati equations represents errors in the estimates of those states. Figures 9.19–9.22 display the single simulation run and theoretical (i.e., square root of appropriate diagonal elements of covariance matrix) error in the estimate of angle, angle rate,

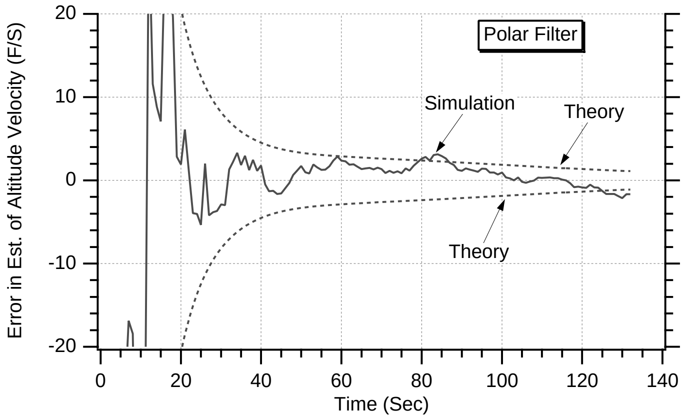

Fig. 9.18 Polar and Cartesian extended Kalman filters yield similar results for altitude velocity estimates.

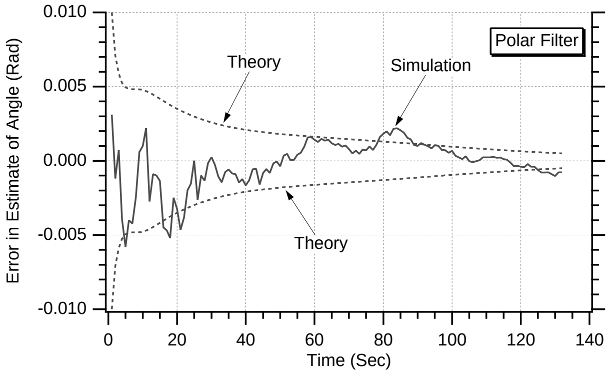

Fig. 9.19 Error in the estimate of angle indicates that extended polar Kalman filter appears to be working properly.

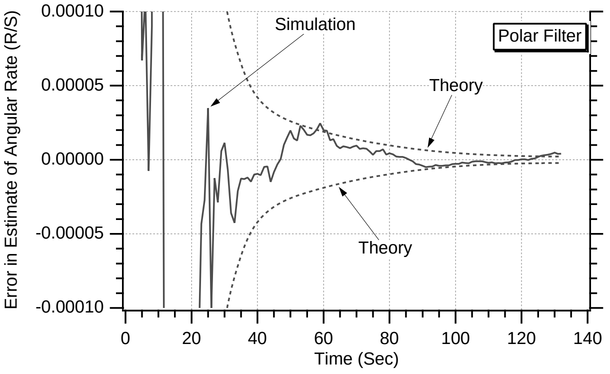

Fig. 9.20 Error in the estimate of angle rate indicates that extended polar Kalman filter appears to be working properly.

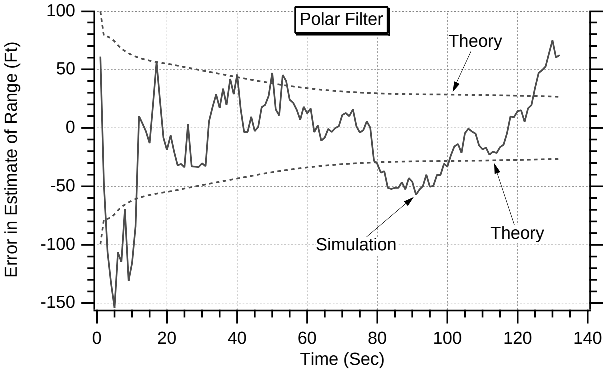

Fig. 9.21 Error in the estimate of range indicates that extended polar Kalman filter appears to be working properly.

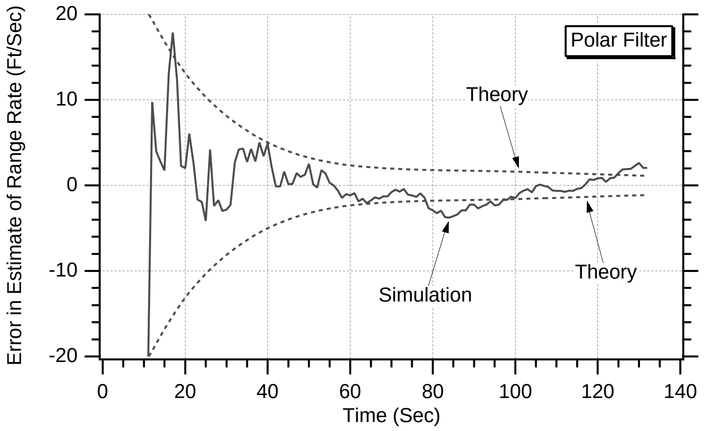

Fig. 9.22 Error in the estimate of range rate indicates that extended polar Kalman filter appears to be working properly.

range, and range rate, respectively. We can see from each of the plots that the single-run simulation error in the estimate is within the theoretical predictions of the covariance matrix approximately 68% of the time, indicating that the extended polar Kalman filter appears to be working properly.

In summary, we can say that there does not appear to be any advantage in using a polar extended Kalman filter for this example. The performance results of the filter are comparable to those of the Cartesian extended Kalman filter, although the computational burden is significantly greater. The increase in computational burden for the polar filter is primarily caused by the states having to be propagated forward at each sampling instant by numerical integration. With the Cartesian extended Kalman filter the fundamental matrix was exact, and numerical integration was not required for state propagation (i.e., multiplying the states by the fundamental matrix propagates the states).

Using Linear Decoupled Polynomial Kalman Filters

The cannon-launched projectile problem was ideal for an extended Kalman filter because either the measurements were nonlinear (if viewed in a Cartesian frame) or the system equations were nonlinear (if viewed in a polar frame). The problem also could have been manipulated to make it appropriate to apply two linear decoupled two-state polynomial Kalman filters—one in the downrange direction and the other in the altitude direction. This can be accomplished by pretending that the filter measures downrange and altitude (i.e., x_T^* and y_T^*), rather than angle and range (i.e., θ^* and r^*).

To find the equivalent noise on downrange and altitude, we have to go through the following procedure. We have already shown that downrange and altitude can be expressed in terms of range and angle according to

$$x_T = r\cos\theta + x_R$$
$$y_T = r\sin\theta + y_R$$

Using calculus, we can find the total differential of the preceding two equations as

$$\Delta x_T = \frac{\partial x_T}{\partial r}\Delta r + \frac{\partial x_T}{\partial \theta}\Delta\theta = \cos\theta\Delta r - r\sin\theta\Delta\theta$$

$$\Delta y_T = \frac{\partial y_T}{\partial r}\Delta r + \frac{\partial y_T}{\partial \theta}\Delta\theta = \sin\theta\Delta r + r\cos\theta\Delta\theta$$

We can square each of the preceding equations to obtain

$$\Delta x_T^2 = \cos^2\theta\Delta r^2 - 2r\sin\theta\cos\theta\Delta r\Delta\theta + r^2\sin^2\theta\Delta\theta^2$$
$$\Delta y_T^2 = \sin^2\theta\Delta r^2 + 2r\sin\theta\cos\theta\Delta r\Delta\theta + r^2\cos^2\theta\Delta\theta^2$$

To find the variance of the pseudonoise on downrange and altitude, we can take expectations of both sides of the equations, assuming that the actual range and angle noise are not correlated [i.e., $E(\Delta r\Delta\theta) = 0$). Therefore, we can say that

$$E(\Delta x_T^2) = \cos^2\theta E(\Delta r^2) + r^2\sin^2\theta E(\Delta\theta^2)$$
$$E(\Delta y_T^2) = \sin^2\theta E(\Delta r^2) + r^2\cos^2\theta E(\Delta\theta^2)$$

Because

$$\sigma_{x_T}^2 = E(\Delta x_T^2)$$
$$\sigma_{y_T}^2 = E(\Delta y_T^2)$$
$$\sigma_r^2 = E(\Delta r^2)$$
$$\sigma_\theta^2 = E(\Delta\theta^2)$$

we can also say that

$$\sigma_{x_T}^2 = \cos^2\theta\sigma_r^2 + r^2\sin^2\theta\sigma_\theta^2$$
$$\sigma_{y_T}^2 = \sin^2\theta\sigma_r^2 + r^2\cos^2\theta\sigma_\theta^2$$

In other words, we are pretending that we are measuring downrange and altitude with measurement accuracies described by the two preceding equations. We now have enough information for the design of the filters in each channel. In downrange there is no acceleration acting on the projectile. The model of the real world in this channel can be represented in state-space form as

$$\begin{bmatrix} \dot{x}_T \\ \ddot{x}_T \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_T \\ \dot{x}_T \end{bmatrix} + \begin{bmatrix} 0 \\ u_s \end{bmatrix}$$

where u_s is white noise, which has been added to acceleration for possible protection as a result of uncertainties in modeling. Therefore, the continuous process-noise matrix is given by

$$\boldsymbol{Q} = \begin{bmatrix} 0 & 0 \\ 0 & \Phi_s \end{bmatrix}$$

where Φ_s is the spectral density of the white process noise. The systems dynamics matrix is obtained from the state-space equation by inspection and is given by

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

We have already shown in Chapter 4 that the fundamental matrix can be obtained exactly from this systems dynamics matrix and is given by

$$\Phi(\boldsymbol{t}) = \begin{bmatrix} 0 & t \\ 0 & 1 \end{bmatrix}$$

which means that the discrete fundamental matrix is

$$\Phi_k = \begin{bmatrix} 0 & T_s \\ 0 & 1 \end{bmatrix}$$

The discrete process-noise matrix is related to the continuous process-noise matrix according to

$$\boldsymbol{Q}_k \int_0^{T_s} \Phi(\tau)\boldsymbol{Q}\Phi^T(\tau)\,\mathrm{d}t$$

We have already derived the discrete process-noise matrix for this type of example in Chapter 4, and the results are given by

$$\boldsymbol{Q}_k = \Phi_s \begin{bmatrix} \dfrac{T_s^3}{3} & \dfrac{T_s^2}{2} \\ \dfrac{T_s^2}{2} & T_s \end{bmatrix}$$

The measurement equation is linear because we have assumed a pseudomeasurement and it is given by

$$x_T^* = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_T \\ \dot{x}_T \end{bmatrix} + v_x$$

Therefore, the measurement matrix is given by

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

In this case the measurement noise matrix is simply a scalar and is given by

$$\boldsymbol{R}_k = \sigma^2_{x_T}$$

We now have enough information to solve the Riccati equations for the Kalman gains in the downrange channel. Because the fundamental matrix is exact in this example, we can also use the preceding matrices in the linear Kalman-filtering equation

$$\hat{\boldsymbol{x}}_k = \Phi_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}\Phi_k \hat{\boldsymbol{x}}_{k-1})$$

to obtain

$$\begin{bmatrix} x_{T_k} \\ \hat{\dot{x}}_{T_k} \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{T_{k-1}} \\ \hat{\dot{x}}_{T_{k-1}} \end{bmatrix} + \begin{bmatrix} K_{1_k} \\ K_{2_k} \end{bmatrix} \left[x^*_{T_k} - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{T_{k-1}} \\ \hat{\dot{x}}_{T_{k-1}} \end{bmatrix} \right]$$

Multiplying out the terms of the preceding matrix equation yields the two scalar equations, which represent the linear polynomial Kalman filter in the downrange channel or

$$\hat{x}_{T_k} = \hat{x}_{t_{k-1}} + T_s \hat{\dot{x}}_{T_{k-1}} + K_{1_k}(x^*_{T_k} - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}})$$
$$\hat{\dot{x}}_{T_k} = \hat{\dot{x}}_{T_{k-1}} + K_{2_k}(x^*_{T_k} - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}})$$

In the altitude channel there is gravity, and so the equations will be slightly different. The state-space equation describing the real world is now given by

$$\begin{bmatrix} \dot{y}_T \\ \ddot{y}_T \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_T \\ \dot{y}_T \end{bmatrix} + \begin{bmatrix} 0 \\ -g \end{bmatrix} + \begin{bmatrix} 0 \\ u_s \end{bmatrix}$$

where g is gravity and is assumed to be known in advance. Therefore, the continuous known disturbance or control vector in this equation is given by

$$\boldsymbol{G} = \begin{bmatrix} 0 \\ -g \end{bmatrix}$$

Because the systems dynamics matrix in the altitude channel is identical to the one in the downrange channel, the fundamental matrix is also the same. Therefore, the discrete control vector can be found from the continuous one from the relationship

$$\boldsymbol{G}_k = \int_0^{T_s} \Phi \boldsymbol{G}\,\mathrm{d}\tau = \int_0^{T_s} \begin{bmatrix} 0 & \tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -g \end{bmatrix} \mathrm{d}\tau = \begin{bmatrix} -0.5T_s^2 g \\ -T_s g \end{bmatrix}$$

The measurement equation is given by

$$y_T^* = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} y_T \\ \dot{y}_T \end{bmatrix} + v_y$$

so that the measurement matrix in the altitude channel is the same as it was in the downrange channel. The measurement noise matrix is also simply a scalar in the altitude channel and is given by

$$\boldsymbol{R}_k = \sigma_{y_T}^2$$

We now have enough information to solve the Riccati equations for the Kalman gains in the altitude channel. Because the fundamental matrix is exact in this example, we can also use the preceding matrices in the Kalman-filtering equation

$$\hat{\boldsymbol{x}}_k = \Phi_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{G}_k \boldsymbol{u}_{k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}\Phi_k \hat{\boldsymbol{x}}_{k-1} - \boldsymbol{H}\boldsymbol{G}_k \boldsymbol{u}_{k-1})$$

to obtain

$$\begin{bmatrix} y_{Y_k} \\ \hat{\dot{y}}_{T_k} \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_{T_{k-1}} \\ \hat{\dot{y}}_{T_{k-1}} \end{bmatrix} - \begin{bmatrix} 0.5gT_s^2 \\ gT_s \end{bmatrix} + \begin{bmatrix} K_{1_k} \\ K_{2_k} \end{bmatrix} \left[y_{T_k}^* - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_{T_{k-1}} \\ \hat{\dot{y}}_{T_{k-1}} \end{bmatrix} + \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0.5gT_s^2 \\ gT_s \end{bmatrix} \right]$$

Multiplying out the terms of the preceding matrix equation yields the two scalar equations, which represent the linear polynomial Kalman filter in the downrange channel or

$$\hat{y}_{T_k} = \hat{y}_{T_{k-1}} + T_s \hat{\dot{y}}_{T_{k-1}} - 0.5gT_s^2 + K_{1_k}(x_{T_k}^* - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}} + 0.5gT_s^2)$$
$$\hat{\dot{y}}_{T_k} = \hat{\dot{y}}_{T_{k-1}} - gT_s + K_{2_k}(x_{T_k}^* - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}} + 0.5gT_s^2)$$

The preceding equations for the two linear decoupled polynomial Kalman filters and associated Riccati equations were programmed and are shown in Listing 9.4, along with a simulation of the real world. Again, we can see that the process-noise matrix is set to zero in this example (i.e., $\Phi_s = 0$). As before, we have initialized the states of the filter close to the true values. The position states are in error by 1000 ft, and the velocity states are in error by 100 ft/s, while the

Listing 9.4 Two decoupled polynomial linear Kalman filters for tracking projectile

```
C THE FIRST THREE STATEMENTS INVOKE THE ABSOFT RANDOM
  NUMBER GENERATOR ON THE MACINTOSH
        GLOBAL DEFINE
                INCLUDE 'quickdraw.inc'
        END
        IMPLICIT REAL*8(A-H,O-Z)
        REAL*8 P(2,2),Q(2,2),M(2,2),PHI(2,2),HMAT(1,2),HT(2,1),PHIT(2,2)
        REAL*8 RMAT(1,1),IDN(2,2),PHIP(2,2),PHIPPHIT(2,2),HM(1,2)
        REAL*8 HMHT(1,1),HMHTR(1,1),HMHTRINV(1,1),MHT(2,1),K(2,1)
        REAL*8 KH(2,2),IKH(2,2)
        REAL*8 RMATY(1,1),PY(2,2),PHIPY(2,2),PHIPPHITY(2,2),MY(2,2)
        REAL*8 HMY(1,2),HMHTY(1,1),HMHTRY(1,1),HMHTRINVY(1,1)
        REAL*8 MHTY(2,1),KY(2,1),KHY(2,2),IKHY(2,2)
        INTEGER ORDER
        TS=1.
        ORDER=2
        PHIS=0.
        SIGTH=.01
        SIGR=100.
        VT=3000.
        GAMDEG=45.
        G=32.2
        XT=0.
        YT=0.
        XTD=VT*COS(GAMDEG/57.3)
        YTD=VT*SIN(GAMDEG/57.3)
        XR=100000.
        YR=0.
        OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
        OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
        T=0.
        S=0.
        H=.001
        DO 14 I=1,ORDER
        DO 14 J=1,ORDER
        PHI(I,J)=0.
        P(I,J)=0.
        Q(I,J)=0.
        IDN(I,J)=0.
 14     CONTINUE
        PHI(1,1)=1.
        PHI(1,2)=TS
        PHI(2,2)=1.
        HMAT(1,1)=1.
        HMAT(1,2)=0.
        CALL MATTRN(PHI,ORDER,ORDER,PHIT)
        CALL MATTRN(HMAT,1,ORDER,HT)
        IDN(1,1)=1.
        IDN(2,2)=1.
        Q(1,1)=PHIS*TS*TS*TS/3.
```

(*continued*)

Listing 9.4 (*Continued*)

```
      Q(1,2)=PHIS*TS*TS/2.
      Q(2,1)=Q(1,2)
      Q(2,2)=PHIS*TS
      P(1,1)=1000.**2
      P(2,2)=100.**2
      PY(1,1)=1000.**2
      PY(2,2)=100.**2
      XTH=XT+1000.
      XTDH=XTD-100.
      YTH=YT-1000.
      YTDH=YTD-100.
      WHILE(YT>=0.)
            XTOLD=XT
            XTDOLD=XTD
            YTOLD=YT
            YTDOLD=YTD
            XTDD=0.
            YTDD=-G
            XT=XT+H*XTD
            XTD=XTD+H*XTDD
            YT=YT+H*YTD
            YTD=YTD+H*YTDD
            T=T+H
            XTDD=0.
            YTDD=-G
            XT=.5*(XTOLD+XT+H*XTD)
            XTD=.5*(XTDOLD+XTD+H*XTDD)
            YT=.5*(YTOLD+YT+H*YTD)
            YTD=.5*(YTDOLD+YTD+H*YTDD)
            S=S+H
            IF(S>=(TS-.00001))THEN
                  S=0.
                  THETH=ATAN2((YTH-YR),(XTH-XR))
                  RTH=SQRT((XTH-XR)**2+(YTH-YR)**2)
                  RMAT(1,1)=(COS(THETH)*SIGR)**2+(RTH*SIN(THETH)
    1               *SIGTH)**2
                  CALL  MATMUL(PHI,ORDER,ORDER,P,ORDER,
                    ORDER,PHIP)
                  CALL  MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
    1                 ORDER,PHIPPHIT)
                  CALL  MATADD(PHIPPHIT,ORDER,ORDER,Q,M)
                  CALL  MATMUL(HMAT,1,ORDER,M,ORDER,
                    ORDER,HM)
                  CALL  MATMUL(HM,1,ORDER,HT,ORDER,1,HMHT)
                  CALL  MATADD(HMHT,1,1,RMAT,HMHTR)
                    HMHTRINV(1,1)=1./HMHTR(1,1)
                  CALL  MATMUL(M,ORDER,ORDER,HT,ORDER,1,MHT)
                  CALL  MATMUL(MHT,ORDER,1,HMHTRINV,1,1,K)
                  CALL  MATMUL(K,ORDER,1,HMAT,1,ORDER,KH)
```

(*continued*)

Listing 9.4 (*Continued*)

```
                  CALL  MATSUB(IDN,ORDER,ORDER,KH,IKH)
                  CALL  MATMUL(IKH,ORDER,ORDER,M,ORDER,
                    ORDER,P)
                  CALL  GAUSS(THETNOISE,SIGTH)
                  CALL  GAUSS(RTNOISE,SIGR)
                  THET=ATAN2((YT-YR),(XT-XR))
                  RT=SQRT((XT-XR)**2+(YT-YR)**2)
                  THETMEAS=THET+THETNOISE
                  RTMEAS=RT+RTNOISE
                  XTMEAS=RTMEAS*COS(THETMEAS)+XR
                  RES1=XTMEAS-XTH-TS*XTDH
                  XTH=XTH+TS*XTDH+K(1,1)*RES1
                  XTDH=XTDH+K(2,1)*RES1
                  RMATY(1,1)=(SIN(THETH)*SIGR)**2+
1                   (RTH*COS(THETH)*SIGTH)**2
                  CALL  MATMUL(PHI,ORDER,ORDER,PY,ORDER,
1                    ORDER,PHIPY)
                  CALL  MATMUL(PHIPY,ORDER,ORDER,PHIT,ORDER,
1                    ORDER,PHIPPHITY)
                  CALL  MATADD(PHIPPHITY,ORDER,ORDER,Q,MY)
                  CALL  MATMUL(HMAT,1,ORDER,MY,ORDER,ORDER,
                    HMY)
                  CALL  MATMUL(HMY,1,ORDER,HT,ORDER,1,HMHTY)
                  CALL  MATADD(HMHTY,1,1,RMATY,HMHTRY)
                    HMHTRINVY(1,1)=1./HMHTRY(1,1)
                  CALL  MATMUL(MY,ORDER,ORDER,HT,ORDER,1,
                    MHTY)
                  CALL  MATMUL(MHTY,ORDER,1,HMHTRINVY,1,
                    1,KY)
                  CALL  MATMUL(KY,ORDER,1,HMAT,1,ORDER,KHY)
                  CALL  MATSUB(IDN,ORDER,ORDER,KHY,IKHY)
                  CALL  MATMUL(IKHY,ORDER,ORDER,MY,ORDER,
                    ORDER,PY)
                  YTMEAS=RTMEAS*SIN(THETMEAS)+YR
                  RES2=YTMEAS-YTH-TS*YTDH+.5*TS*TS*G
                  YTH=YTH+TS*YTDH-.5*TS*TS*G+KY(1,1)*RES2
                  YTDH=YTDH-TS*G+KY(2,1)*RES2
                  ERRX=XT-XTH
                  SP11=SQRT(P(1,1))
                  ERRXD=XTD-XTDH
                  SP22=SQRT(P(2,2))
                  ERRY=YT-YTH
                  SP11Y=SQRT(PY(1,1))
                  ERRYD=YTD-YTDH
                  SP22Y=SQRT(PY(2,2))
                  WRITE(9,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
                  WRITE(1,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
                  WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
1                   ERRY,SP11Y,-SP11Y,ERRYD,SP22Y,-SP22Y
          ENDIF
```

(*continued*)

Listing 9.4 (*Continued*)

```
          END DO
          PAUSE
          CLOSE(1)
          CLOSE(2)
          END

C SUBROUTINE GAUSS IS SHOWN IN LISTING 1.8
C SUBROUTINE MATTRN IS SHOWN IN LISTING 1.3
C SUBROUTINE MATMUL IS SHOWN IN LISTING 1.4
C SUBROUTINE MATADD IS SHOWN IN LISTING 1.1
C SUBROUTINE MATSUB IS SHOWN IN LISTING 1.2
```

initial covariance matrix reflects those errors. The simulation running time of Listing 9.4 is much faster than that of the extended Cartesian Kalman filter of Listing 9.1 because there are effectively fewer equations with two two-state filters than there are for those with one four-state filter.

The nominal case was run, and the resultant errors in the estimates for position and velocity in the downrange and altitude direction appear in Figs. 9.23–9.26. We can see from the four figures that the linear Kalman filters appear to be working correctly because the single-run simulation errors in the estimates appear to lie between the theoretical predictions of the associated covariance matrices approximately 68% of the time. However, if we compare these results to the errors in the estimates of the Cartesian extended Kalman filter shown in Figs. 9.8–9.12, we can see that these new results appear to be slightly worse than before. In other words, the errors in the estimates of the two two-state linear

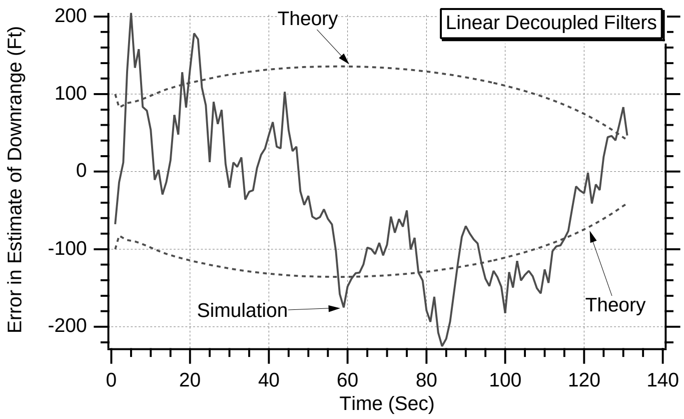

Fig. 9.23 Linear decoupled Kalman filter downrange error in the estimate of position is larger than that of extended Kalman filter.

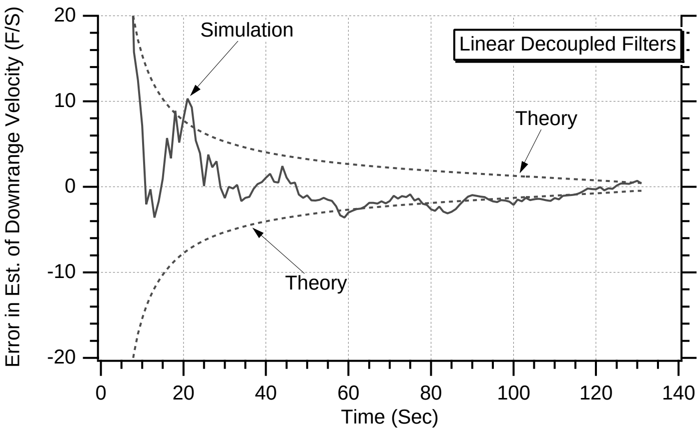

Fig. 9.24 Linear decoupled Kalman filter downrange error in the estimate of velocity is larger than that of extended Kalman filter.

decoupled polynomial Kalman filters appear to be larger than those of the four-state Cartesian extended Kalman filter.

Using Linear Coupled Polynomial Kalman Filters

In the preceding section we pretended that the noise in the altitude and downrange channels was not correlated, which allowed us to decouple the linear Kalman filters. Let us now remove that restriction in order to see what happens.

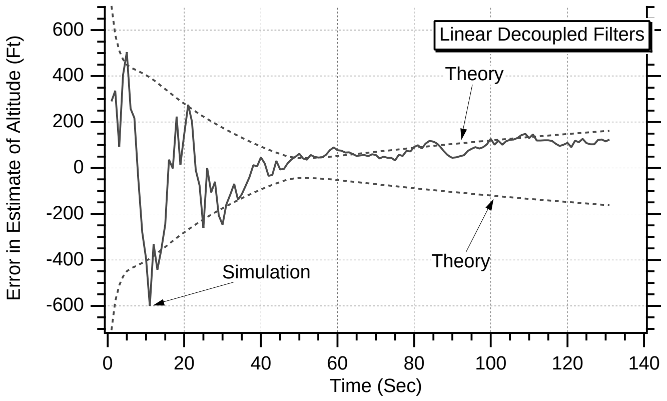

Fig. 9.25 Linear decoupled Kalman filter altitude error in the estimate of position is larger than that of extended Kalman filter.

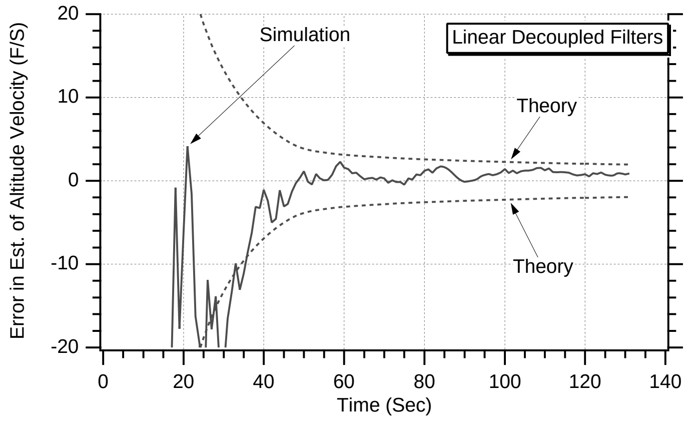

Fig. 9.26 Linear decoupled Kalman filter altitude error in the estimate of velocity is larger than that of extended Kalman filter.

Recall that we have already shown that downrange and altitude can be expressed in terms of range and angle according to

$$x_T = r\cos\theta + x_R$$
$$y_T = r\sin\theta + y_R$$

Using calculus, we can find the total differential of the preceding two equations as

$$\Delta x_T = \frac{\partial x_T}{\partial r}\Delta r + \frac{\partial x_T}{\partial \theta}\Delta\theta = \cos\theta\Delta r - r\sin\theta\Delta\theta$$
$$\Delta y_T = \frac{\partial y_T}{\partial r}\Delta r + \frac{\partial y_T}{\partial \theta}\Delta\theta = \sin\theta\Delta r + r\cos\theta\Delta\theta$$

Before, we squared each of the preceding equations to obtain

$$\Delta x_T^2 = \cos^2\theta\Delta r^2 - 2r\sin\theta\cos\theta\Delta r\Delta\theta + r^2\sin^2\theta\Delta\theta^2$$
$$\Delta y_T^2 = \sin^2\theta\Delta r^2 + 2r\sin\theta\cos\theta\Delta r\Delta\theta + r^2\cos^2\theta\Delta\theta^2$$

but now we also can find

$$\Delta x_T\Delta y_T = \sin\theta\cos\theta\Delta r^2 + r\sin^2\theta\Delta r\Delta\theta + r\cos^2\theta\Delta r\Delta\theta - r^2\sin\theta\cos\theta\Delta\theta^2$$

To find the variance of the pseudonoise, we can take expectations of both sides of the equations, assuming that the actual range and angle noise are not correlated [i.e., $E(\Delta r \Delta\theta) = 0$). Therefore, we can say that

$$E(\Delta x_T^2) = \cos^2\theta E(\Delta r^2) + r^2\sin^2\theta E(\Delta\theta^2)$$
$$E(\Delta y_T^2) = \sin^2\theta E(\Delta r^2) + r^2\cos^2\theta E(\Delta\theta^2)$$
$$E(\Delta x_T \Delta y_T) = \sin\theta\cos\theta E(\Delta r^2) - r^2\sin\theta\cos\theta E(\Delta\theta^2)$$

Because

$$\sigma_{x_T}^2 = E(\Delta x_T^2)$$
$$\sigma_{y_T}^2 = E(\Delta y_T^2)$$
$$\sigma_{x_T y_T}^2 = E(\Delta x_T \Delta y_T)$$
$$\sigma_r^2 = E(r^2)$$
$$\sigma_\theta^2 = E(\Delta\theta^2)$$

we can say that

$$\sigma_{x_T}^2 = \cos^2\theta\theta\sigma_r^2 + r^2\sin^2\theta\sigma_\theta^2$$
$$\sigma_{y_T}^2 = \sin^2\theta\theta\sigma_r^2 + r^2\cos^2\theta\sigma_\theta^2$$
$$\sigma_{x_T y_T}^2 = \sin\theta\cos\theta\sigma_r^2 - r^2\sin\theta\cos\theta\sigma_\theta^2$$

Therefore, unlike the preceding section, where we had a downrange and altitude scalar noise matrix, we now have a fully coupled 2×2 noise matrix given by

$$\boldsymbol{R}_k = \begin{bmatrix} \cos^2\theta\sigma_r^2 + r^2\sin^2\theta\sigma_\theta^2 & \sin\theta\cos\theta\sigma_r^2 - r^2\sin\theta\cos\theta\sigma_\theta^2 \\ \sin\theta\cos\theta\sigma_r^2 - r^2\sin\theta\cos\theta\sigma_\theta^2 & \sin^2\theta\sigma_r^2 + r^2\cos^2\theta\sigma_\theta^2 \end{bmatrix}$$

Because we are no longer decoupling the channels, the states upon which the new filter will be based are given by

$$\boldsymbol{x} = \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix}$$

and the pseudomeasurements are now linearly related to the states according to

$$\begin{bmatrix} x_T^* \\ y_T^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

From the preceding equation we can see that the measurement matrix is given by

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Therefore, when the preceding Cartesian states are chosen, we have already shown that the state-space differential equation describing projectile motion is also linear and given by

$$\begin{bmatrix} \dot{x}_T \\ \ddot{x}_T \\ \dot{y}_T \\ \ddot{y}_T \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} + \begin{bmatrix} 0 \\ u_s \\ 0 \\ u_s \end{bmatrix}$$

Again, notice that in the preceding equation gravity g is not a state that has to be estimated but is assumed to be known in advance. We also have added white process noise u_s to the acceleration portion of the equations as protection for effects that may not considered by the Kalman filter. The preceding equation is equivalent to the model used in deriving the extended Cartesian Kalman filter. The only difference in this formulation is that the actual measurements have been simplified to pseudomeasurements for purposes of making the filter linear.

From the preceding state-space equation we can see that systems dynamics matrix is given by

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

and we have already shown in this chapter that the fundamental matrix for this system of equations is

$$\Phi(t) = \begin{bmatrix} 0 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore, the discrete fundamental matrix can be found by substituting T_s for t and is given by

$$\Phi_k = \begin{bmatrix} 0 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Because the continuous process-noise matrix is given by

$$\boldsymbol{Q} = E(\boldsymbol{w}\boldsymbol{w}^T)$$

we can say that, for this example, the continuous process-noise matrix is

$$\boldsymbol{Q}(t) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \Phi_s & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Phi_s \end{bmatrix}$$

where Φ_s is the spectral density of the white noise sources assumed to be on the downrange and altitude accelerations acting on the projectile. The discrete process-noise matrix can be derived from the continuous process-noise matrix according to

$$\boldsymbol{Q}_k = \int_0^{T_s} \Phi(\tau)\boldsymbol{Q}\Phi^T(\tau)\,\mathrm{d}t$$

and we have already shown that the discrete process-noise matrix turns out to be

$$\boldsymbol{Q}_k = \begin{bmatrix} \dfrac{T_s^3\Phi_s}{3} & \dfrac{T_s^2\Phi_s}{2} & 0 & 0 \\ \dfrac{T_s^2\Phi_s}{2} & T_s\Phi_s & 0 & 0 \\ 0 & 0 & \dfrac{T_s^3\Phi_s}{3} & \dfrac{T_s^2\Phi_s}{2} \\ 0 & 0 & \dfrac{T_s^2\Phi_s}{2} & T_s\Phi_s \end{bmatrix}$$

From the state-space equation we can see that the continuous control vector in this equation is given by

$$\boldsymbol{G} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix}$$

Therefore, the discrete control vector can be found from the continuous one from the relationship

$$\boldsymbol{G}_k = \int_0^{T_s} \Phi \boldsymbol{G}\,\mathrm{d}\tau = \int_0^{T_s} \begin{bmatrix} 0 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} \mathrm{d}\tau = \begin{bmatrix} 0 \\ 0 \\ -0.5gT_s^2 \\ -gT_s \end{bmatrix}$$

We now have defined all of the matrices required to solve the Riccati equations. Because the fundamental matrix is exact in this example, we also can use the preceding matrices in the Kalman-filtering equation

$$\hat{\boldsymbol{x}}_k = \Phi_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{G}_k \boldsymbol{u}_{k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}\Phi_k \hat{\boldsymbol{x}}_{k-1} - \boldsymbol{H}\boldsymbol{G}_k \boldsymbol{u}_{k-1})$$

to obtain

$$\begin{bmatrix} \hat{x}_{T_k} \\ \hat{\dot{x}}_{T_k} \\ \hat{y}_{T_k} \\ \hat{\dot{y}}_{T_k} \end{bmatrix} = \begin{bmatrix} 0 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_{T_{k-1}} \\ \hat{\dot{x}}_{T_{k-1}} \\ \hat{y}_{T_{k-1}} \\ \hat{\dot{y}}_{T_{k-1}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -0.5gT_s^2 \\ -gT_s \end{bmatrix}$$
$$+ \begin{bmatrix} K_{11_k} & K_{12_k} \\ K_{21_k} & K_{22_k} \\ K_{31_k} & K_{32_k} \\ K_{41_k} & K_{42_k} \end{bmatrix} \left[\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_{T-1} \\ \hat{\dot{x}}_{T_{k-1}} \\ \hat{y}_{T_{k-1}} \\ \hat{\dot{y}}_{T_{k-1}} \end{bmatrix} \right.$$
$$\left. - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -0.5gT_s^2 \\ -gT_s \end{bmatrix} \right]$$

Multiplying out the terms yields the following scalar equations:

$$\mathrm{Res}_1 = x^*_{T_k} - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}}$$

$$\mathrm{Res}_2 = y^*_{T_k} - \hat{y}_{T_{k-1}} - T_s \hat{\dot{y}}_{T_{k-1}} + 0.5gT_s^2$$

$$\hat{x}_{T_k} = \hat{x}_{T_{k-1}} + T_s \hat{\dot{x}}_{T_{k-1}} + K_{11_k}\mathrm{Res}_1 + K_{12_k}\mathrm{Res}_2$$

$$\hat{\dot{x}}_{T_k} = \hat{\dot{x}}_{T_{k-1}} + K_{21_k}\mathrm{Res}_1 + K_{22_k}\mathrm{Res}_2$$

$$\hat{y}_{T_k} = \hat{y}_{T_{k-1}} + T_s \hat{\dot{y}}_{T_{k-1}} - 0.5gT_s^2 + K_{31_k}\mathrm{Res}_1 + K_{32_k}\mathrm{Res}_2$$

$$\hat{\dot{y}}_{T_k} = \hat{\dot{y}}_{T_{k-1}} - gT_s + K_{41_k}\mathrm{Res}_1 + K_{42_k}\mathrm{Res}_2$$

The preceding equations for the four-state linear polynomial Kalman filters and associated Riccati equations were programmed and are shown in Listing 9.5, along with a simulation of the real world. Again, we can see that the process-noise matrix is set to zero in this example (i.e., $\Phi_s = 0$). As before, we have initialized the states of the filter close to the true values. The position states are in

Listing 9.5 Linear coupled four-state polynomial Kalman filter

```
C THE FIRST THREE STATEMENTS INVOKE THE ABSOFT RANDOM
  NUMBER GENERATOR ON THE MACINTOSH
        GLOBAL DEFINE
                INCLUDE 'quickdraw.inc'
        END
        IMPLICIT REAL*8(A-H,O-Z)
        REAL*8 P(4,4),Q(4,4),M(4,4),PHI(4,4),HMAT(2,4),HT(2,4),PHIT(4,4)
        REAL*8 RMAT(2,2),IDN(4,4),PHIP(4,4),PHIPPHIT(4,4),HM(2,4)
        REAL*8 HMHT(2,2),HMHTR(2,2),HMHTRINV(2,2),MHT(4,2),K(4,2)
        REAL*8 KH(4,4),IKH(4,4)
        INTEGER ORDER
        TS=1.
        ORDER=4
        PHIS=0.
        SIGTH=.01
        SIGR=100.
        VT=3000.
        GAMDEG=45.
        G=32.2
        XT=0.
        YT=0.
        XTD=VT*COS(GAMDEG/57.3)
        YTD=VT*SIN(GAMDEG/57.3)
        XR=100000.
        YR=0.
        OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
        OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
        T=0.
        S=0.
        H=.001
        DO 14 I=1,ORDER
        DO 14 J=1,ORDER
        PHI(I,J)=0.
        P(I,J)=0.
        Q(I,J)=0.
        IDN(I,J)=0.
 14     CONTINUE
        PHI(1,1)=1.
        PHI(1,2)=TS
        PHI(2,2)=1.
        PHI(3,3)=1.
```

(*continued*)

Listing 9.5 (*Continued*)

```
PHI(3,4)=TS
PHI(4,4)=1.
HMAT(1,1)=1.
HMAT(1,2)=0.
HMAT(1,3)=0.
HMAT(1,4)=0.
HMAT(2,1)=0.
HMAT(2,2)=0.
HMAT(2,3)=1.
HMAT(2,4)=0.
CALL  MATTRN(PHI,ORDER,ORDER,PHIT)
CALL  MATTRN(HMAT,2,ORDER,HT)
IDN(1,1)=1.
IDN(2,2)=1.
IDN(3,3)=1.
IDN(4,4)=1.
Q(1,1)=PHIS*TS*TS*TS/3.
Q(1,2)=PHIS*TS*TS/2.
Q(2,1)=Q(1,2)
Q(2,2)=PHIS*TS
Q(3,3)=PHIS*TS*TS*TS/3.
Q(3,4)=PHIS*TS*TS/2.
Q(4,3)=Q(3,4)
Q(4,4)=PHIS*TS
P(1,1)=1000.**2
P(2,2)=100.**2
P(3,3)=1000.**2
P(4,4)=100.**2
XTH=XT+1000.
XTDH=XTD-100.
YTH=YT-1000.
YTDH=YTD+100.
WHILE(YT>=0.)
          XTOLD=XT
          XTDOLD=XTD
          YTOLD=YT
          YTDOLD=YTD
          XTDD=0.
          YTDD=-G
          XT=XT+H*XTD
          XTD=XTD+H*XTDD
          YT=YT+H*YTD
          YTD=YTD+H*YTDD
          T=T+H
          XTDD=0.
          YTDD=-G
          XT=.5*(XTOLD+XT+H*XTD)
          XTD=.5*(XTDOLD+XTD+H*XTDD)
          YT=.5*(YTOLD+YT+H*YTD)
```

(*continued*)

Listing 9.5 (*Continued*)

```
              YTD=.5*(YTDOLD+YTD+H*YTDD)
              S=S+H
              IF(S>=(TS-.00001))THEN
                   S=0.
                   THETH=ATAN2((YTH-YR),(XTH-XR))
                   RTH=SQRT((XTH-XR)**2+(YTH-YR)**2)
                   RMAT(1,1)=(COS(THETH)*SIGR)**2+(RTH*SIN
     1               (THETH)*SIGTH)**2
                   RMAT(2,2)=(SIN(THETH)*SIGR)**2+(RTH*COS
     1               (THETH)*SIGTH)**2
                   RMAT(1,2)=SIN(THETH)*COS(THETH)*(SIGR**2-
     1               (RTH*SIGTH)**2)
                   RMAT(2,1)=RMAT(1,2)
                   CALL  MATMUL(PHI,ORDER,ORDER,P,ORDER,
                     ORDER,PHIP)
                   CALL  MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
     1               ORDER,PHIPPHIT)
                   CALL  MATADD(PHIPPHIT,ORDER,ORDER,Q,M)
                   CALL  MATMUL(HMAT,2,ORDER,M,ORDER,
                     ORDER,HM)
                   CALL  MATMUL(HM,2,ORDER,HT,ORDER,2,HMHT)
                   CALL  MATADD(HMHT,2,2,RMAT,HMHTR)
                     DET=HMHTR(1,1)*HMHTR(2,2)-
                     HMHTR(1,2)*HMHTR(2,1)
                   HMHTRINV(1,1)=HMHTR(2,2)/DET
                   HMHTRINV(1,2)=-HMHTR(1,2)/DET
                   HMHTRINV(2,1)=-HMHTR(2,1)/DET
                   HMHTRINV(2,2)=HMHTR(1,1)/DET
                   CALL  MATMUL(M,ORDER,ORDER,HT,ORDER,
                     2,MHT)
                   CALL  MATMUL(MHT,ORDER,2,HMHTRINV,2,2,K)
                   CALL  MATMUL(K,ORDER,2,HMAT,2,ORDER,KH)
                   CALL  MATSUB(IDN,ORDER,ORDER,KH,IKH)
                   CALL  MATMUL(IKH,ORDER,ORDER,M,ORDER,
                     ORDER,P)
                   CALL  GAUSS(THETNOISE,SIGTH)
                   CALL  GAUSS(RTNOISE,SIGR)
                   THET=ATAN2((YT-YR),(XT-XR))
                   RT=SQRT((XT-XR)**2+(YT-YR)**2)
                   THETMEAS=THET+THETNOISE
                   RTMEAS=RT+RTNOISE
                   XTMEAS=RTMEAS*COS(THETMEAS)+XR
                   YTMEAS=RTMEAS*SIN(THETMEAS)+YR
                   RES1=XTMEAS-XTH-TS*XTDH
                   RES2=YTMEAS-YTH-TS*YTDH+.5*TS*TS*G
                   XTH=XTH+TS*XTDH+K(1,1)*RES1+K(1,2)*RES2
                   XTDH=XTDH+K(2,1)*RES1+K(2,2)*RES2
                   YTH=YTH+TS*YTDH-.5*TS*TS*G+K(3,1)*RES1+
     1                 K(3,2)*RES2
                   YTDH=YTDH-TS*G+K(4,1)*RES1+K(4,2)*RES2
```

(*continued*)

Listing 9.5 (*Continued*)

```
                         ERRX=XT-XTH
                         SP11=SQRT(P(1,1))
                         ERRXD=XTD-XTDH
                         SP22=SQRT(P(2,2))
                         ERRY=YT-YTH
                         SP33=SQRT(P(3,3))
                         ERRYD=YTD-YTDH
                         SP44=SQRT(P(4,4))
                         WRITE(9,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
                         WRITE(1,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
                         WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
     1                     ERRY,SP33,-SP33,ERRYD,SP44,-SP44
                 ENDIF
          END DO
          PAUSE
          CLOSE(1)
          CLOSE(2)
          END

C SUBROUTINE GAUSS IS SHOWN IN LISTING 1.8
C SUBROUTINE MATTRN IS SHOWN IN LISTING 1.3
C SUBROUTINE MATMUL IS SHOWN IN LISTING 1.4
C SUBROUTINE MATADD IS SHOWN IN LISTING 1.1
C SUBROUTINE MATSUB IS SHOWN IN LISTING 1.2
```

error by 1000 ft, and the velocity states are in error by 100 ft/s, while the initial covariance matrix reflects those errors.

The nominal case of Listing 9.5 was run in which there was no process noise. Figures 9.27–9.30 display the errors in the estimates of the downrange and altitude states. By comparing these figures with those of the Cartesian extended Kalman filter (see Figs. 9.8–9.11), we can see that the results are now virtually identical. In other words, in the cannon-launched projectile example there is no degradation in performance by using a linear coupled filter with pseudomeasurements as opposed to the extended Cartesian Kalman filter with the actual measurements. However, there was some degradation in performance when we used decoupled linear filters, as was done in the preceding section.

Robustness Comparison of Extended and Linear Coupled Kalman Filters

So far we have seen that both the extended and linear coupled Kalman filters appear to have approximately the same performance. However, these results were based on work in which the initialization errors in the filter states were relatively modest. Let us first see how the extended Kalman filter reacts to larger initialization errors.

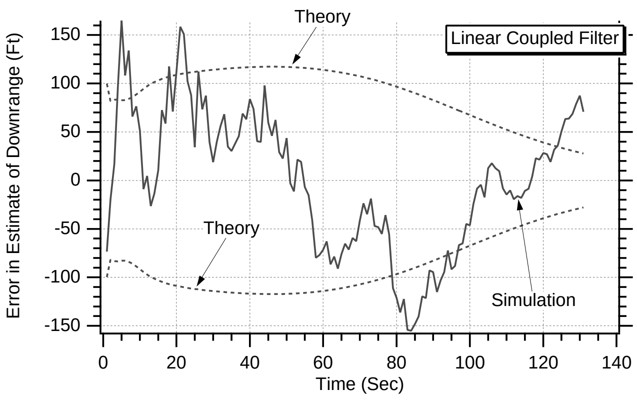

Fig. 9.27 Error in estimate of downrange is the same for the linear coupled and extended Kalman filters.

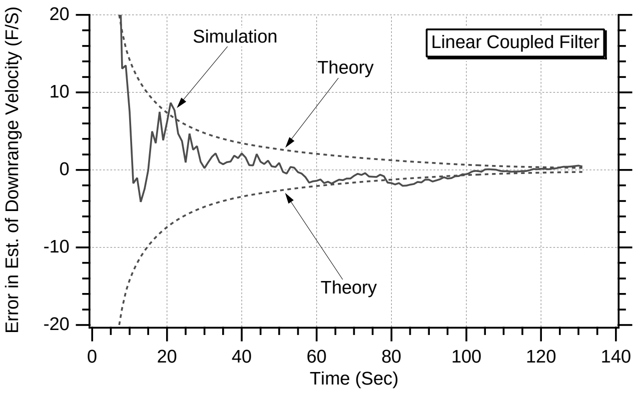

Fig. 9.28 Error in estimate of downrange velocity is the same for the linear coupled and extended Kalman filters.

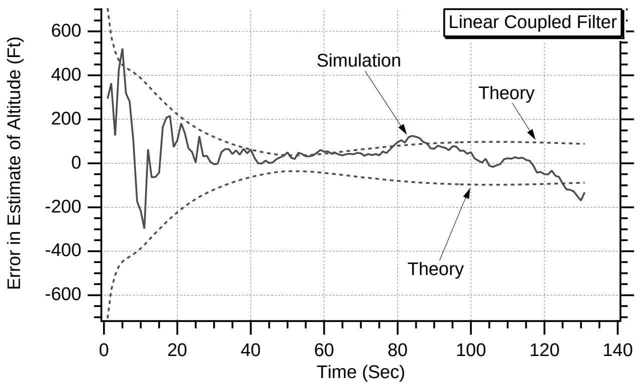

Fig. 9.29 Error in estimate of altitude is the same for the linear coupled and extended Kalman filters.

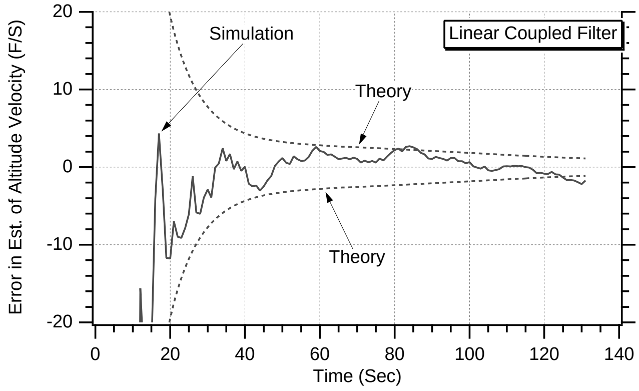

Fig. 9.30 Error in estimate of altitude velocity is the same for the linear coupled and extended Kalman filters.

In all of the experiments of this chapter, the initial state estimates of the filter were the true states plus an error of 1000 ft on position and 100 ft/s on velocity, as indicated here:

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} x_T(0) \\ \dot{x}_T(0) \\ y_T(0) \\ \dot{y}_T(0) \end{bmatrix} + \begin{bmatrix} 1000 \\ -100 \\ -1000 \\ 100 \end{bmatrix}$$

The initial covariance matrix of both the linear and extended Kalman filters reflected the initialization errors by having each diagonal element be the square of the appropriate initialization error or

$$\boldsymbol{P}_0 = \begin{bmatrix} 1000^2 & 0 & 0 & 0 \\ 0 & 100^2 & 0 & 0 \\ 0 & 0 & 1000^2 & 0 \\ 0 & 0 & 0 & 100^2 \end{bmatrix}$$

To see how sensitive the extended Kalman filter is to initialization errors, let us first double those errors so that the initial estimates of the extended Kalman filter and its covariance matrix are now given by

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} x_T(0) \\ \dot{x}_T(0) \\ y_T(0) \\ \dot{y}_T(0) \end{bmatrix} + \begin{bmatrix} 2000 \\ -200 \\ -2000 \\ 200 \end{bmatrix}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} 2000^2 & 0 & 0 & 0 \\ 0 & 200^2 & 0 & 0 \\ 0 & 0 & 2000^2 & 0 \\ 0 & 0 & 0 & 200^2 \end{bmatrix}$$

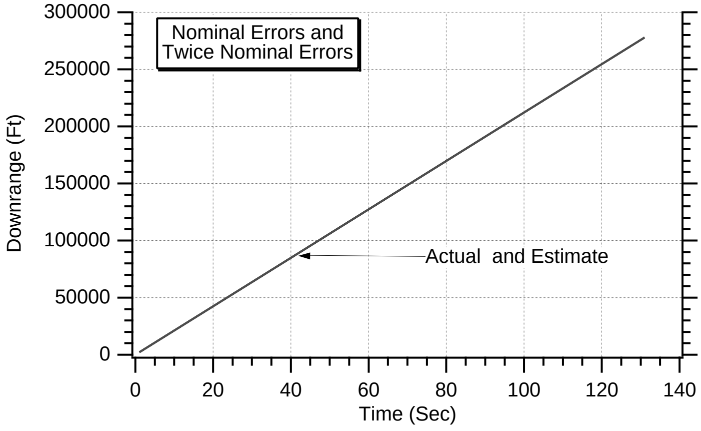

Fig. 9.31 Extended Kalman filter appears to yield good estimates even when initialization errors are twice as large as nominal.

Listing 9.1, which represents the extended Cartesian Kalman filter, was rerun with the preceding initialization errors. We can see from Fig. 9.31 that the estimate of the downrange location of the projectile appears to be independent of initialization error because the estimates do not change when the initialization errors are doubled. In addition, the estimates of the projectile location appear to be very close to the actual projectile location.

Next, the initialization errors were made five times larger than the nominal values so that the initial state estimates and covariance matrix are now given by

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} x_T(0) \\ \dot{x}_T(0) \\ y_T(0) \\ \dot{y}_T(0) \end{bmatrix} + \begin{bmatrix} 5000 \\ -500 \\ -5000 \\ 500 \end{bmatrix}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} 5000^2 & 0 & 0 & 0 \\ 0 & 500^2 & 0 & 0 \\ 0 & 0 & 5000^2 & 0 \\ 0 & 0 & 0 & 500^2 \end{bmatrix}$$

Listing 9.1 was rerun with the new initialization errors. We can see from Fig. 9.32 that with the larger initialization errors the extended Kalman filter's estimate of projectile downrange degrades severely. In addition, we can see that for the first 10 s there is a filter transient caused by the initialization errors. However, after the transient settles out the filter estimate of the projectile downrange position severely lags the actual downrange location.

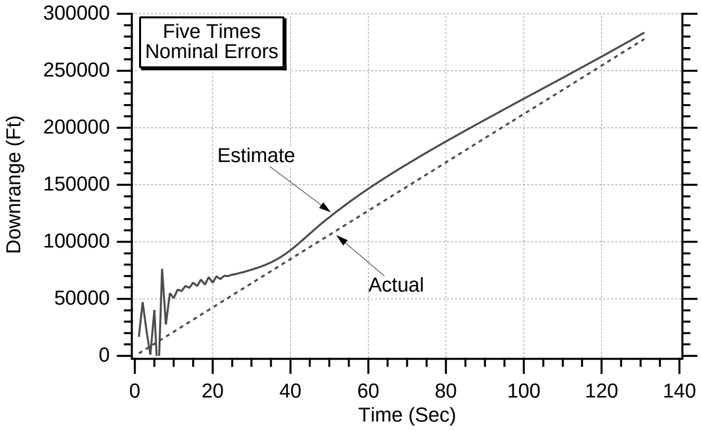

Fig. 9.32 Estimates from extended Kalman filter degrade severely when initialization errors are five time larger than nominal.

Next, the extended Kalman filter's initialization errors were made 10 times larger than the nominal values. The new initial state estimates and initial covariance matrix are now given by

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} x_T(0) \\ \dot{x}_T(0) \\ y_T(0) \\ \dot{y}_T(0) \end{bmatrix} + \begin{bmatrix} 10000 \\ -1000 \\ -10000 \\ 1000 \end{bmatrix}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} 10{,}000^2 & 0 & 0 & 0 \\ 0 & 1{,}000^2 & 0 & 0 \\ 0 & 0 & 10{,}000^2 & 0 \\ 0 & 0 & 0 & 1{,}000^2 \end{bmatrix}$$

Listing 9.1 was rerun with the preceding initialization conditions. We can see from Fig. 9.33 that when the initialization errors of the filter estimate and the initial covariance matrix are 10 times the nominal value the estimate of projectile, downrange is worthless. We can see that the filter estimate of the downrange location of the projectile is monotonically decreasing, whereas the actual location is monotonically increasing.

All of the initialization experiments conducted so far in this section had zero process noise (i.e., $\Phi_s = 0$). We know from Chapter 6 that the addition of process noise will widen the bandwidth of the Kalman filter, and hopefully this also will improve the extended Kalman filter's transient response. The case in which the initialization errors for the extended Kalman were 10 times larger than nominal and the filter's estimates were worthless was reexamined when process-noise was present. Figure 9.34 shows that the addition of process noise (i.e., $\Phi_s = 1$) now

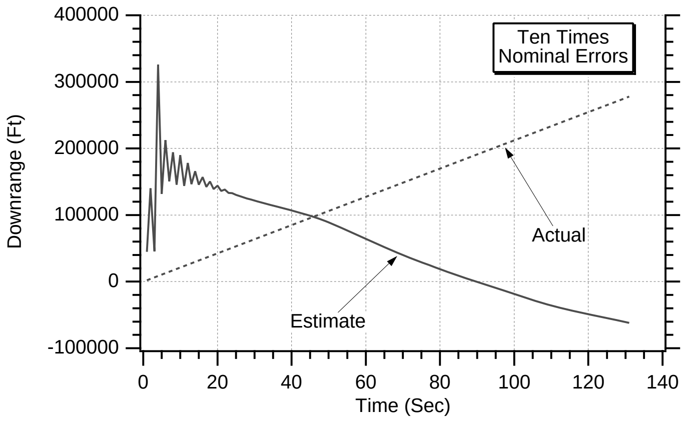

Fig. 9.33 Estimates from extended Kalman filter are worthless when initialization errors are 10 times larger than nominal.

enables the extended Kalman filter to estimate the downrange location of the projectile. Figures 9.34 and 9.35 show that increasing the process noise enables the filter to eliminate the estimation transient error associated with the large initialization errors more rapidly. This is not surprising because we already know from Chapter 6 that increasing the process noise will increase the bandwidth of the Kalman filter and thus improve its transient response.

For more precise work we must also see how the actual errors in the estimates compare to the covariance matrix predictions for the errors in the estimates. Figure 9.36 examines the error in the estimate of the projectile's downrange location. We can see that the extended Kalman filter with process noise (i.e., $\Phi_s = 1000$) now appears to be working correctly because, after an initial transient

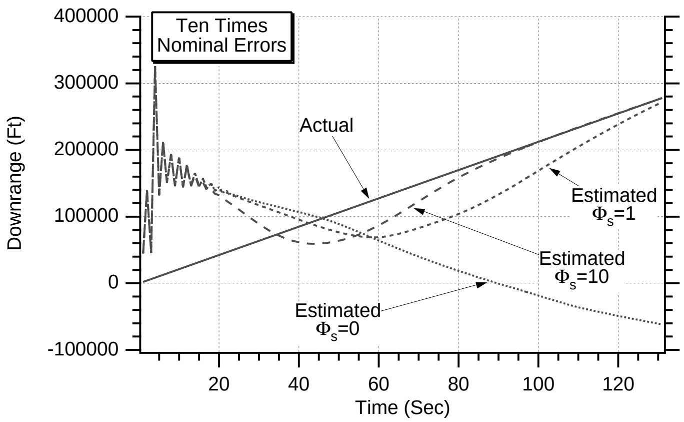

Fig. 9.34 Addition of process noise enables extended Kalman filter to better estimate downrange in presence of large initialization errors.

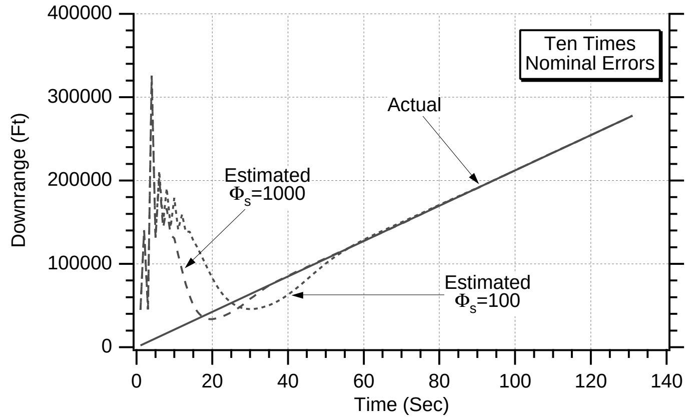

Fig. 9.35 Making process noise larger further improves extended Kalman filter's ability to estimate downrange in presence of large initialization errors.

period, the error in the estimate of projectile downrange position agrees with the results of the covariance matrix. In other words, the simulated error in the estimate appears to lie within the theoretical bounds (i.e., square root of P_{11}) approximately 68% of the time.

We now ask ourselves how the linear coupled polynomial Kalman filter would have performed under similar circumstances. Listing 9.5, which we have already shown has the linear coupled polynomial Kalman filter, was rerun for the case in which the initialization errors were 10 times their nominal value. The linear coupled polynomial Kalman filter was run *without* process-noise. We can see from Fig. 9.37 that under these adverse conditions the linear filter does not appear to have a problem in estimating the downrange location of the projectile. Again,

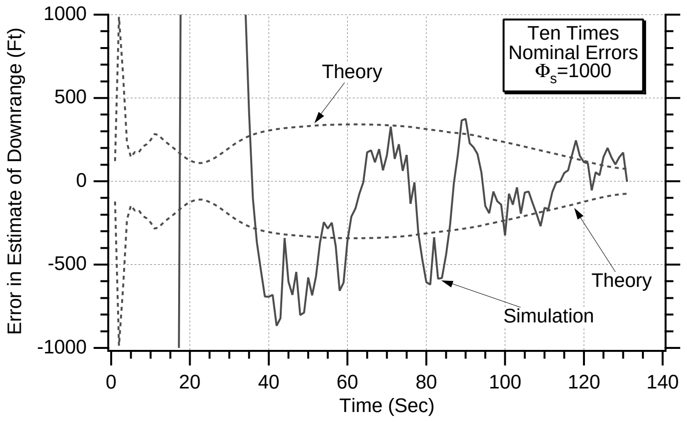

Fig. 9.36 After an initial transient period simulation results agree with covariance matrix predictions.

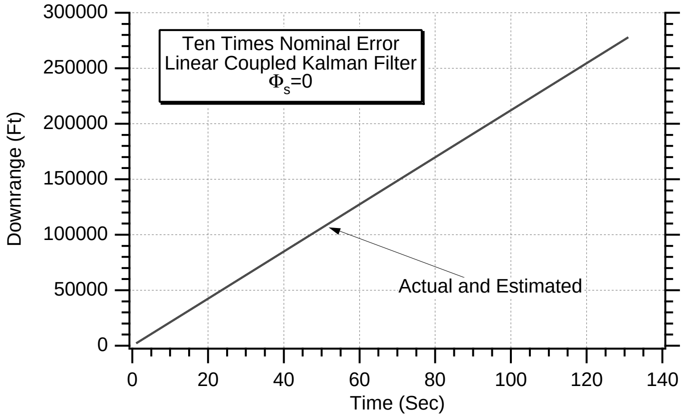

Fig. 9.37 Linear coupled polynomial Kalman filter does not appear to be sensitive to large initialization errors.

we can examine filter performance more critically by examining the error in the estimate from simulation and comparing it to the covariance matrix prediction. We can see from Fig. 9.38 that the single-run simulation results of the error in the estimate of the projectile downrange location appear to lie within the theoretical bounds approximately 68% of the time. Therefore, the linear coupled polynomial Kalman filter *without* process-noise appears to be working in the presence of large initialization errors. If we compare these results to the case in which the initialization errors were their nominal values (see Fig. 9.8), we can see that the performance of the linear filter does not appear to depend on the size of the initialization error.

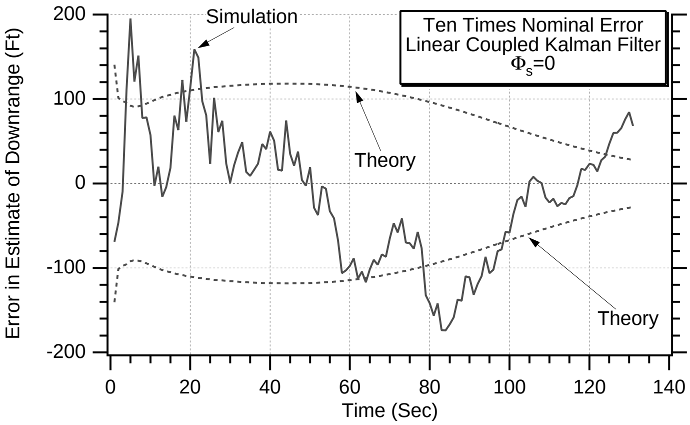

Fig. 9.38 Linear coupled polynomial Kalman filter appears to be working correctly in presence of large initialization errors.

If we compare the preceding results to those of the extended Kalman filter (see Fig. 9.36), we can see that the performance of the linear Kalman filter is far superior to that of the extended Kalman filter. When the initialization errors were 10 times the nominal value, the extended filter yielded estimates for projectile downrange location with errors on the order of 300 ft, while the linear filter errors were on the order of 100 ft.

To further demonstrate the robustness of the linear coupled polynomial Kalman filter to large initialization errors, another experiment was conducted. In this case the initial state estimates were set to zero or

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The diagonal elements of the initial covariance matrix were then set to infinity to account for the large uncertainties in the initial errors in the estimates or

$$\boldsymbol{P}_0 = \begin{bmatrix} \infty & 0 & 0 & 0 \\ 0 & \infty & 0 & 0 \\ 0 & 0 & \infty & 0 \\ 0 & 0 & 0 & \infty \end{bmatrix}$$

Figure 9.39 shows sample results for the linear coupled polynomial Kalman filter. We can see from Fig. 9.39 that the errors in the estimate of downrange are well behaved, even when the filter is extremely poorly initialized. Recall that when the diagonal elements of the initial covariance matrix are infinite and the process-noise matrix is zero we have a recursive least-squares filter. By comparing Fig 9.39 with Fig. 9.8, we can see that the filter performance is virtually identical, indicating once again that initialization is not an important issue for a linear Kalman filter. However, good initialization is critical for an extended Kalman filter.

Therefore, if we have a problem in which both a linear and extended Kalman filter work and yield comparable performance, it is safer to use the linear Kalman filter because it is more robust when large initialization errors are present.

Summary

In this chapter we have shown how an extended Kalman filter could be built in either the Cartesian or polar coordinate systems. For the cannon-launched projectile problem, in which the filter was in the Cartesian system, the state-space equation was linear, but the measurement equation was nonlinear. If the filter were in the polar system, the state-space equation was nonlinear, but the measurement equation was linear. We saw that for the example considered the performance of both extended Kalman filters was approximately the same. In other words, if computational requirements were not an issue the preferred

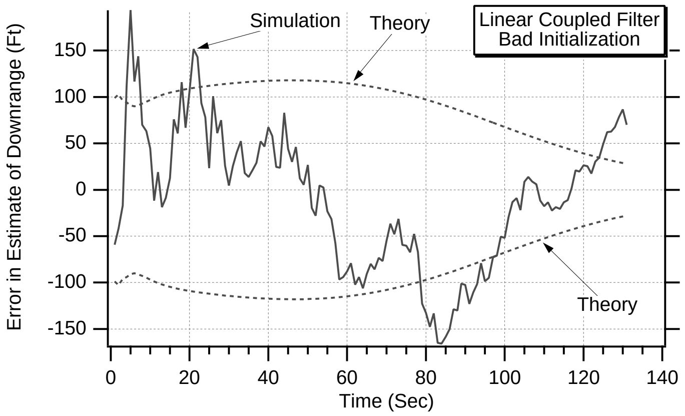

Fig. 9.39 Linear coupled Kalman filter performance appears to be independent of initialization errors.

coordinate system for the extended Kalman filter was a matter of personal preference and not related to performance issues. The polar filter was more computationally expensive than the Cartesian filter because numerical integration was required for state propagation. We also saw that a linear coupled Kalman filter that had near identical performance to that of the extended Kalman filters could also be built. The advantage of the linear filter was that it was more robust, as illustrated by an experiment in which the filter was improperly initialized. The linear coupled filter worked fine under such adverse conditions, whereas the extended Kalman filter failed. Process noise could be used to fix the extended Kalman filter, but the price paid was much larger estimation errors when compared to the linear Kalman filter. We also saw that if computation was an issue the linear filter also could be decoupled. The computer throughput requirements were better in the resultant filter pair because a pair of two-state linear filters require significantly less computation than one linear four-state filter. However, the price for reduced computation was a slight degradation in filter performance as measured by the estimation errors.

Reference

[1]Richards, J. A., Sears, F. W., Wehr, M. R., and Zemansky, M. W., *Modern University Physics, Part 1: Mechanics and Thermodynamics*, Addison Wesley Longman, Reading, MA, 1960, pp. 54–74.