# Discrete-Time Implementation of a Two-State Kalman Observer in a Scan-Based Real-Time Control Environment

Taylor Turner
*Independent Controls and Software Engineering*
*resume.tturner@gmail.com*

(Dated: February 16, 2026)

A two-state discrete Kalman observer is implemented as a Structured Text Add-On Instruction (AOI) within a Rockwell Logix programmable logic controller (PLC). The observer estimates the position and velocity from a position-only measurement under a constant-velocity model. The discrete-time formulation is explicitly mapped to scalar procedural computation consistent with scan-based real-time execution. The effects of variable timestep, deterministic safeguards, and commissioning-oriented tuning are analyzed. Experimental validation using controlled signal injection demonstrates alignment between theoretical prediction and observed behavior. The implementation approach generalizes to other embedded real-time systems with similar computational constraints.

## INTRODUCTION

State estimation is foundational to modern control systems. Kalman filtering is traditionally presented in continuous-time or fixed-rate discrete-time frameworks, often assuming availability of matrix libraries and linear algebra primitives. In such treatments, sampling intervals are typically constant, and computational resources are implicitly sufficient to support generalized operations. Industrial control systems, however, frequently operate under markedly different execution semantics.

Programmable logic controllers (PLCs) execute logic cyclically in a deterministic scan model. Each scan performs the following procedural sequence: inputs are sampled, user logic executes top-to-bottom, outputs are updated, and the cycle repeats. State variables persist between scans, but no implicit concurrency or background execution exists. The program therefore evolves strictly through sequential evaluation. Each scan corresponds directly to a discrete-time step of duration $\Delta t_k$, determined by task scheduling and runtime load rather than by an externally enforced fixed-rate scheduler.
Under this execution model:

- Computation must complete within bounded scan time,
- Dynamic memory allocation is unavailable,
- Matrix libraries are absent,
- Floating-point precision is limited,
- Deterministic behavior under enable/disable transitions is required.

The business of this work is to formalize a two-state constant-velocity Kalman observer and explicitly relate each theoretical component to its deterministic scalar implementation within such a procedural real-time environment. The emphasis is not on applying Kalman theory, but on demonstrating how the discrete-time equations map directly to bounded, scan-synchronous execution.

## DISCRETE-TIME STATE-SPACE MODEL

The observer state is defined as

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix}, \tag{1}$$

where $x_k$ represents position and $\dot{x}_k$ represents velocity.

### Prediction Model

Assuming locally constant velocity over scan interval $\Delta t_k$,

$$x_{k+1} = x_k + \Delta t_k \dot{x}_k, \tag{2}$$
$$\dot{x}_{k+1} = \dot{x}_k. \tag{3}$$

In matrix form,

$$\mathbf{x}_{k+1} = F_k \mathbf{x}_k + \mathbf{w}_k, \quad F_k = \begin{bmatrix} 1 & \Delta t_k \\ 0 & 1 \end{bmatrix}. \tag{4}$$

Within the PLC, $\Delta t_k$ is computed per scan:

$$\Delta t_k = \frac{\texttt{dt (ms)}}{1000}.$$

The prediction step executes sequentially:

```
x_pred := x_pred + dt_s * x_dot_pred;
```

Velocity remains unchanged during prediction, consistent with the constant-velocity assumption.

### Measurement Model

Only position is measured:

$$z_k = H\mathbf{x}_k + v_k, \quad H = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{5}$$

Velocity is inferred through covariance coupling.

## COVARIANCE PROPAGATION IN SCAN TIME

The predicted covariance is

$$P_k^- = F_k P_k F_k^T + Q. \tag{6}$$

Expanding explicitly,

$$P_{00}^- = P_{00} + \Delta t_k (P_{01} + P_{10}) + \Delta t_k^2 P_{11}, \tag{7}$$
$$P_{01}^- = P_{01} + \Delta t_k P_{11}, \tag{8}$$
$$P_{10}^- = P_{10} + \Delta t_k P_{11}, \tag{9}$$
$$P_{11}^- = P_{11}. \tag{10}$$

In the implementation, these expressions are expanded into scalar arithmetic using temporary variables prior to committing to persistent covariance memory. This avoids partial updates during a scan.
Process noise is introduced as per-scan variance increments:

$$P_{00}^- \leftarrow P_{00}^- + q_x, \tag{11}$$
$$P_{11}^- \leftarrow P_{11}^- + q_{\dot{x}}. \tag{12}$$

These parameters represent engineering tuning values rather than continuous-time spectral densities. As a consequence, filter behavior implicitly scales with scan rate.

## MEASUREMENT UPDATE

The scan-synchronous Kalman correction can be expressed compactly as a prediction–correction recursion applied once per PLC scan. Let the predicted state and covariance be

$$\hat{\mathbf{x}}_k^- = F_k \hat{\mathbf{x}}_{k-1}, \qquad P_k^- = F_k P_{k-1} F_k^{\mathsf{T}} + Q, \tag{13}$$

where $F_k$ encodes the measured scan interval $\Delta t_k$ and $Q$ is the per-scan process noise covariance.
Given a position-only measurement $z_k$ with $H = \begin{bmatrix} 1 & 0 \end{bmatrix}$ and measurement noise variance $R = r_x$, the full correction step may be written as the single state update

$$\boxed{\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + K_k \left( z_k - H \hat{\mathbf{x}}_k^- \right)} \tag{14}$$

with gain

$$K_k = P_k^- H^{\mathsf{T}} \left( H P_k^- H^{\mathsf{T}} + R \right)^{-1}. \tag{15}$$

The corresponding covariance update in the computationally efficient form used by the AOI is

$$\boxed{P_k = (I - K_k H) \, P_k^- .} \tag{16}$$

To connect this formulation directly to the PLC implementation, note that $H$ selects the position component only. The innovation (residual) and its variance are therefore scalar:

$$y_k = z_k - \hat{x}_k^-, \qquad S_k = H P_k^- H^{\mathsf{T}} + R = P_{00}^- + r_x. \tag{17}$$

Thus the gain reduces to a two-element vector,

$$K_k = \frac{1}{S_k} \begin{bmatrix} P_{00}^- \\ P_{10}^- \end{bmatrix} \equiv \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}, \tag{18}$$

which yields the scalar state corrections used in Structured Text:

$$\hat{x}_k = \hat{x}_k^- + K_0 \, y_k, \tag{19}$$
$$\hat{v}_k = \hat{v}_k^- + K_1 \, y_k. \tag{20}$$

In the AOI, $S_k$ is guarded against non-positive values to prevent division instability, and the covariance symmetry condition $P_{10} = P_{01}$ is enforced after update to mitigate long-run floating-point drift.

## COVARIANCE UPDATE AND NUMERICAL SAFEGUARDS

The covariance update uses the computationally efficient form

$$P_k = (I - K_k H) P_k^-. \tag{21}$$

The Joseph stabilized form was not adopted due to increased computational cost relative to PLC execution constraints.
To mitigate floating-point asymmetry over long runtimes, covariance symmetry is explicitly enforced:

$$P_{10} = P_{01}.$$

This preserves practical positive semi-definiteness under finite precision arithmetic.

## EFFECT OF VARIABLE SCAN TIME

Because $\Delta t_k$ appears in both state and covariance propagation, scan jitter influences transient response. Larger $\Delta t_k$ increases position advance, covariance growth, and state coupling.
Since $Q$ is not scaled by $\Delta t_k$, gain evolution exhibits mild scan-rate dependence. This design choice prioritizes commissioning clarity over strict sampling invariance.

## VELOCITY DAMPING HEURISTIC

An optional multiplicative decay is applied when innovation magnitude falls below a threshold:

$$\hat{v}_k \leftarrow \alpha \hat{v}_k, \quad 0 < \alpha < 1. \tag{22}$$

While not part of optimal Kalman theory, this heuristic mitigates overshoot at motion discontinuities and improves practical signal behavior in industrial systems.

## EXPERIMENTAL VALIDATION

A deterministic ramp–hold–ramp–hold waveform was injected into the PLC at $\Delta t = 50$ ms using PyLogix. Measurement noise was added as bounded uniform noise:

$$x_{\mathrm{meas}}(k) = x_{\mathrm{true}}(k) + \eta_k, \quad \eta_k \sim \mathcal{U}(-2, 2). \tag{23}$$

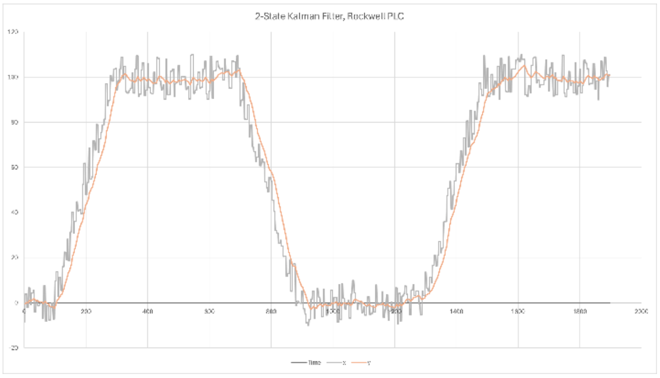

FIG. 1. Studio 5000 trend showing noisy injected signal (gray) and Kalman filtered estimate (orange).

The filtered output demonstrates noise attenuation during steady-state holds, minimal phase lag during ramps, and predictable transient correction at discontinuities. The empirical behavior aligns with theoretical expectations derived from the discrete-time propagation equations.

## DISCUSSION

Although implemented within a PLC, the scalar-expanded observer structure generalizes to other real-time embedded environments exhibiting similar constraints, including microcontrollers, servo drives, automotive control units, and edge industrial nodes.
In each case, estimation must execute deterministically within cyclic control loops, often without matrix libraries or dynamic memory allocation. The explicit scalar mapping of Riccati equations to bounded arithmetic enables portability across such platforms while preserving theoretical integrity.

## CONCLUSION

A two-state discrete Kalman observer has been rigorously mapped to a scan-based procedural execution environment using scalar arithmetic and deterministic safeguards. The implementation preserves theoretical structure while accommodating industrial constraints.
Experimental validation confirms noise attenuation, accurate slope tracking, and stable transient behavior. The approach demonstrates that state-space estimation can be executed transparently and robustly within constrained real-time control systems.
Future extensions include acceleration-state augmentation and timestep-scaled process noise modeling to improve sampling invariance.

## NOMENCLATURE

**$\mathbf{x}_k$:** State vector at discrete time step $k$, $\mathbf{x}_k = \begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix}$.

**$x_k$:** True position state at time step $k$.

**$\dot{x}_k$:** True velocity state at time step $k$.

**$\hat{\mathbf{x}}_k$:** Estimated (posterior) state vector after measurement update.

**$\hat{\mathbf{x}}_k^-$:** Predicted (prior) state estimate before measurement update.

**$\hat{x}_k$:** Estimated position component of $\hat{\mathbf{x}}_k$.

**$\hat{v}_k$:** Estimated velocity component of $\hat{\mathbf{x}}_k$.

**$k$:** Discrete time index corresponding to PLC scan number.

**$\Delta t_k$:** Scan interval duration at time step $k$ (seconds).

**$F_k$:** Discrete-time state transition matrix, $F_k = \begin{bmatrix} 1 & \Delta t_k \\ 0 & 1 \end{bmatrix}$.

**$\mathbf{w}_k$:** Process noise vector at time step $k$.

**$z_k$:** Measured position at time step $k$.

**$H$:** Measurement matrix, $H = \begin{bmatrix} 1 & 0 \end{bmatrix}$.

**$v_k$:** Measurement noise at time step $k$.

**$P_k$:** Posterior state covariance matrix at time step $k$.

**$P_k^-$:** Predicted (prior) covariance matrix before measurement update.

**$P_{ij}$:** Element of covariance matrix $P$, where $i, j \in \{0, 1\}$.

**$P_{00}$:** Variance of position estimate.

**$P_{11}$:** Variance of velocity estimate.

**$P_{01}, P_{10}$:** Cross-covariance terms between position and velocity.

**$Q$:** Process noise covariance matrix, $Q = \mathrm{diag}(q_x, q_{\dot{x}})$.

**$q_x$:** Per-scan process noise variance for position.

**$q_{\dot{x}}$:** Per-scan process noise variance for velocity.

**$R$:** Measurement noise covariance (scalar in this implementation).

**$r_x$:** Measurement noise variance for position.

**$K_k$:** Kalman gain vector at time step $k$.

**$K_0$:** Position component of Kalman gain.

**$K_1$:** Velocity component of Kalman gain.

**$y_k$:** Innovation (measurement residual), $y_k = z_k - \hat{x}_k^-$.

**$S_k$:** Innovation variance, $S_k = H P_k^- H^\mathsf{T} + R$.

**$I$:** Identity matrix.

**$\alpha$:** Velocity damping (bleed) factor, $0 < \alpha < 1$.

**$\eta_k$:** Injected measurement noise during experimental validation.

**$\mathcal{U}(a, b)$:** Continuous uniform distribution over interval $[a, b]$.

**$x_{\text{true}}(k)$:** Ground-truth injected test signal.

**$x_{\text{meas}}(k)$:** Noisy measured test signal.

**$\mathsf{T}$:** Matrix transpose operator.

**$(\cdot)^{-1}$:** Matrix inverse (scalar reciprocal in this implementation).

**Structured Text Implementation (Procedural Scalar Form)**

For completeness, the AOI logic is reproduced below in the same procedural order it executes within a PLC scan. The implementation is intentionally written as explicit scalar arithmetic (rather than matrix operations) and includes deterministic guards for enable/disable transitions, invalid timesteps, and division safety.

```
// ==========================================
// Enable / Timebase / Init
// ==========================================
IF NOT EnableIn THEN
    // Disable => deterministic pass-through and re-arm init.
    init := 0;
    y := x;
    y_dot := 0.0;
ELSE
    // Convert ms -> seconds; dt_s must be > 0 to proceed.
    IF dt <= 0.0 THEN
        dt_s := 0.0;
    ELSE
        dt_s := dt / 1000.0;
    END_IF;

    // One-time initialization on rising enable.
    IF NOT init THEN
        x_pred     := x;     // start estimate at measurement
        x_dot_pred := 0.0;   // assume stationary initially

        // Initial covariance (row-major): P00,P01,P10,P11
        P[0] := 1.0;   // P00
        P[1] := 0.0;   // P01
        P[2] := 0.0;   // P10
        P[3] := 10.0;  // P11

        K0 := 0.0;
        K1 := 0.0;
        init := 1;
    END_IF;

    // ==========================================
    // Predict + Update (only if dt is valid)
    // ==========================================
    IF dt_s > 0.0 THEN
        // ---------------------------
        // PREDICT (time update)
        // ---------------------------
        // Predict state: x^- = x + dt*v, v^- = v
        x_pred := x_pred + dt_s * x_dot_pred;

        // Predict covariance: P^- = FPF^T + Q (expanded scalar form)
        x_cov[0] := (P[0] + dt_s * P[2]) + dt_s * (P[1] + dt_s * P[3]); // P00^-
        x_cov[1] := (P[1] + dt_s * P[3]);                                // P01^-
        x_cov[2] := (P[2] + dt_s * P[3]);                                // P10^-
        x_cov[3] := (P[3]);                                              // P11^-

        // Add per-scan process noise (engineering tuning)
        x_cov[0] := x_cov[0] + q_x;
        x_cov[3] := x_cov[3] + q_x_dot;

        // ----------------------------
        // UPDATE (measurement)
        // ----------------------------
        // Innovation (residual): y = z - x^-
        y_res := x - x_pred;

        // Innovation variance: S = P00^- + r_x
        S := x_cov[0] + r_x;

        // Guard division: only update if S is positive
        IF S > 0.0 THEN

            // Kalman gains: K = [P00^-; P10^-] / S
            K0 := x_cov[0] / S;
            K1 := x_cov[2] / S;

            // Correct state: x = x^- + K0*y, v = v^- + K1*y
            x_pred     := x_pred     + K0 * y_res;
            x_dot_pred := x_dot_pred + K1 * y_res;

            // Optional heuristic: velocity damping near rest
            IF BleedEnable THEN
                IF ABS(x - x_pred) < BleedThresh THEN
                    x_dot_pred := x_dot_pred * BleedFactor;A
                END_IF;
            END_IF;

            // Covariance update: P = (I - K H) P^-  (H = [1 0])
            P[0] := (1.0 - K0) * x_cov[0];
            P[1] := (1.0 - K0) * x_cov[1];
            P[2] := x_cov[2] - (K1 * x_cov[0]);
            P[3] := x_cov[3] - (K1 * x_cov[1]);

            // Symmetry enforcement: keep P10 == P01
            P[2] := P[1];
        END_IF;
        // Outputs (posterior estimates)
        y     := x_pred;
        y_dot := x_dot_pred;
    ELSE
        // Invalid dt => deterministic pass-through
        y := x;
        y_dot := 0.0;
    END_IF;
END_IF;
```