Optimization-Based Control

Richard M. Murray
Control and Dynamical Systems
California Institute of Technology

With contributions from:
Karl Åström, William Dunbar, Ryan Franz, John Hauser,
Ali Jadbabaie, Mark Milam, Nicolas Petit, and ChatGPT

Version v2.3h (12 March 2023)

Chapter 7

Sensor Fusion

In this chapter we consider the problem of combining the data from different sensors to obtain an estimate of a (common) dynamical system. Unlike the previous chapters, we focus here on discrete-time processes, leaving the continuous-time case to the exercises. We begin with a summary of the input/output properties of discrete-time systems with stochastic inputs, then present the discrete-time Kalman filter, and use that formalism to formulate and present solutions for the sensor fusion problem. Some advanced methods of estimation and fusion are also summarized at the end of the chapter that demonstrate how to move beyond the linear, Gaussian process assumptions.

Prerequisites. The material in this chapter is designed to be reasonably self-contained, so that it can be used without covering Sections 5.3–5.4 or Chapter 6 of this supplement. We assume rudimentary familiarity with discrete-time linear systems, at the level of the brief descriptions in Chapters 3 and 7 of FBS2e, and discrete-time random processes as described in Section 5.2 of these notes.

7.1 Discrete-Time Stochastic Systems

We begin with a concise overview of stochastic system in discrete time, echoing our development of continuous-time random systems described in Chapter 5. We consider systems of the form

$$X[k+1] = AX[k] + Bu[k] + FV[k], \qquad Y[k] = CX[k] + W[k], \tag{7.1}$$

where $X \in \mathbb{R}^n$ represents the state, $u \in \mathbb{R}^m$ represents the (deterministic) input, $V \in \mathbb{R}^q$ represents process disturbances, $Y \in \mathbb{R}^p$ represents the system output and $V \in \mathbb{R}^p$ represents measurement noise.

As in the case of continuous-time systems, we are interested in the response of the system to the random input $V[k]$. We will assume that V is a Gaussian process with zero mean and correlation function $r_V(k, k+d)$ (or correlation matrix $R_V(k, k+d)$ if V is vector valued). As in the continuous case, we say that a random process is white noise if $r_V(k, k+d) = r_V\delta(d)$ with $\delta(d) = 1$ if $d = 0$ and 0 otherwise. (Note that in the discrete-time case, white noise has finite covariance.)

To compute the response $Y[k]$ of the system, we look at the properties of the state vector $X[k]$. For simplicity, we take $u = 0$ (since the system is linear, we can always add it back in by superposition). Note first that the state at time $k+d$ can be written as

$$\begin{aligned} X[k+d] &= AX[k+d-1] + FV[x+l-1] \\ &= A(AX[k+d-2] + FV[x+l-2]) + FV[x+l-1] \\ &= A^d X[k] + \sum_{j=1}^{d} A^{j-1} FV[k+d-j]. \end{aligned}$$

The mean of the state at time k is given by

$$\mathbb{E}(X[k]) = A^k \mathbb{E}(E[0]) + \sum_{j=1}^{k} A^{j-1} F \mathbb{E}(V[k-j]) = A^k \mathbb{E}(X[0]).$$

To compute the covariance $R_X(k, k+d)$, we start by computing $R_X(k, k+1)$:

$$\begin{aligned} R_X(k, k+1) &= \mathbb{E}(X[k] X^\mathsf{T}[k+1]) \\ &= \mathbb{E}((A^k x[0] + A^{k-1} Fw[0] + \cdots + AFw[k-2] + F[k-1]) \cdot \\ &\qquad (A^{k+1} x[0] + A^k Fw[0] + \cdots + Fw[k])^\mathsf{T}) \end{aligned}$$

Performing a similar calculation for $R_X(k, k+d)$, it can be shown that

$$\begin{aligned} R_X(k, k+d) = \big(A^k P[0] (A^\mathsf{T})^k + A^{k-1} F R_V[0] F^\mathsf{T} (A^\mathsf{T})^{k-1} + \dots \\ + F R_V[k] F^\mathsf{T}\big)(A^\mathsf{T})^d =: P[k](A^\mathsf{T})^d, \end{aligned} \tag{7.2}$$

where

$$P[k+1] = AP[k]A^\mathsf{T} + F R_V[k] F^\mathsf{T}. \tag{7.3}$$

The matrix $P[k]$ is the covariance of the state matrix and we see that its value can be computed *recursively* starting with $P[0] = \mathbb{E}(X[0] X^\mathsf{T}[0])$ and then applying equation (7.3). Equations (7.2) and (7.3) are the equivalent of Proposition 5.2 for continuous-time processes. If we additionally assume that V is stationary and focus on the steady state response, we obtain the following.

Proposition 7.1 (Steady state response to white noise)**.** *For a discrete-time, time-invariant, linear system driven by white noise, the correlation matrices for the state and output converge in steady state to*

$$R_X(d) = R_X(k, k+d) = PA^d, \qquad R_Y(d) = C R_X(d) C^\mathsf{T},$$

where P satisfies the algebraic equation

$$APA^\mathsf{T} + F R_V F^\mathsf{T} = 0, \qquad P > 0. \tag{7.4}$$

7.2 Kalman Filters in Discrete Time (FBS2e)

We now consider the optimal estimator in discrete time. This material is presented in FBS2e in slightly simplified (but consistent) form.

Consider a discrete time, linear system with input, having dynamics

$$\begin{aligned} X[k+1] &= AX[k] + Bu[k] + FV[k], \\ Y[k] &= CX[k] + W[k], \end{aligned} \tag{7.5}$$

where $V[k]$ and $W[k]$ are Gaussian, white noise processes satisfying

$$\begin{aligned} &\mathbb{E}(V[k]) = 0 && \mathbb{E}(W[k]) = 0 \\ &\mathbb{E}(V[k]V^\mathsf{T}[j]) = \begin{cases} 0 & k \neq j \\ R_V & k = j \end{cases} && \mathbb{E}(W[k]W^\mathsf{T}[j]) = \begin{cases} 0 & k \neq j \\ R_W & k = j \end{cases} \\ &\mathbb{E}(V[k]W^\mathsf{T}[j]) = 0. \end{aligned} \tag{7.6}$$

We assume that the initial condition is also modeled as a Gaussian random variable with

$$\mathbb{E}(X[0]) = x_0 \qquad \mathbb{E}(X[0]X^\mathsf{T}[0]) = P[0]. \tag{7.7}$$

We wish to find an estimate $\hat{X}[k]$ that gives the minimum mean square error (MMSE) for $\mathbb{E}((\hat{X}[k] - X[k])(\hat{X}[k] - X[k])^\mathsf{T})$ given the measurements $\{Y[l] : 0 \leq l \leq k\}$. We consider an observer of the form

$$\hat{X}[k+1] = A\hat{X}[k] + Bu[k] - L[k](C\hat{X}[k] - Y[k]). \tag{7.8}$$

The following theorem summarizes the main result.

Theorem 7.2. *Consider a random process $X[k]$ with dynamics* (7.5) *and noise processes and initial conditions described by equations* (7.6) *and* (7.7)*. The observer gain L that minimizes the mean square error is given by*

$$L[k] = AP[k]C^\mathsf{T}(R_W + CP[k]C^\mathsf{T})^{-1},$$

where

$$\begin{aligned} P[k+1] &= AP[k]A^\mathsf{T} + FR_VF^\mathsf{T} - AP[k]C^\mathsf{T}R_\epsilon^{-1}CP[k]A^\mathsf{T} \\ P[0] &= \mathbb{E}(X[0]X^\mathsf{T}[0]). \end{aligned} \tag{7.9}$$

Proof. We wish to minimize the mean square of the error, $\mathbb{E}((\hat{X}[k] - X[k])(\hat{X}[k] - X[k])^\mathsf{T})$. We will define this quantity as $P[k]$ and then show that it satisfies the recursion given in equation (7.9). Let $E[k] = C\hat{X}[k] - Y[k]$ be the residual between the measured output and the estimated output. By definition,

$$\begin{aligned} P[k+1] &= \mathbb{E}(E[k+1]E^\mathsf{T}[k+1]) \\ &= (A - LC)P[k](A - LC)^\mathsf{T} + FR_VF^\mathsf{T} + LR_WL^\mathsf{T} \\ &= AP[k]A^\mathsf{T} - AP[k]C^\mathsf{T}L^\mathsf{T} - LCP[k]A^\mathsf{T} + \\ &\qquad L(R_W + CP[k]C^\mathsf{T})L^\mathsf{T} + FR_VF^\mathsf{T}. \end{aligned}$$

Letting $R_\epsilon = (R_W + CP[k]C^\mathsf{T})$, we have

$$\begin{aligned}P[k+1] &= AP[k]A^\mathsf{T} - AP[k]C^\mathsf{T}L^\mathsf{T} - LCP[k]A^\mathsf{T} + LR_\epsilon L^\mathsf{T} + FR_VF^\mathsf{T}\\ &= AP[k]A^\mathsf{T} + \big(L - AP[k]C^\mathsf{T}R_\epsilon^{-1}\big)R_\epsilon\big(L - AP[k]C^\mathsf{T}R_\epsilon^{-1}\big)^\mathsf{T}\\ &\quad - AP[k]C^\mathsf{T}R_\epsilon^{-1}CP[k]A^\mathsf{T} + FR_VF^\mathsf{T}.\end{aligned}$$

In order to minimize this expression, we choose $L = AP[k]C^\mathsf{T}R_\epsilon^{-1}$ and the theorem is proven. □

Note that the Kalman filter has the form of a *recursive* filter: given $P[k] = \mathbb{E}(E[k]E[k]^\mathsf{T})$ at time k, can compute how the estimate and covariance *change*. Thus we do not need to keep track of old values of the output. Furthermore, the Kalman filter gives the estimate $\hat{X}[k]$ *and* the covariance $P[k]$, so we can see how reliable the estimate is. It can also be shown that the Kalman filter extracts the maximum possible information about output data: the correlation matrix for the estimation error of the filter is

$$R_E[j,k] = R\delta_{jk}.$$

In other words, the error is a white noise process, so there is no remaining dynamic information content in the error.

In the special case when the noise is stationary (R_V, R_W constant) and *if* $P[k]$ converges, then the observer gain is constant:

$$L = APC^\mathsf{T}(R_W + CPC^\mathsf{T})^{-1},$$

where P satisfies

$$P = APA^\mathsf{T} + FR_VF^\mathsf{T} - APC^\mathsf{T}\big(R_W + CPC^\mathsf{T}\big)^{-1}CPA^\mathsf{T}.$$

We see that the optimal gain depends on both the process noise and the measurement noise, but in a nontrivial way. Like the use of LQR to choose state feedback gains, the Kalman filter permits a systematic derivation of the observer gains given a description of the noise processes. The solution for the constant gain case is solved by the `dlqe` command in MATLAB and python-control.

7.3 Predictor-Corrector Form

The Kalman filter can be written in a two-step form by separating the correction step (where we make use of new measurements of the output) and the prediction step (where we compute the expected state and covariance at the next time instant).

We make use of the notation $\hat{X}[k|j]$ to represent the estimated state at time instant k given the information up to time j (where typically $j = k-1$). Using this notation, the filter can be solved using the following algorithm:

Step 0: Initialization.

$$\begin{aligned}k &= 1,\\ \hat{X}[0|0] &= \mathbb{E}(X[0]),\\ P[0|0] &= \mathbb{E}(X[0]X^\mathsf{T}[0]).\end{aligned}$$

Step 1: Prediction. Update the estimates and covariance matrix to account for all data taken up to time $k-1$:

$$
\begin{aligned}
\hat{X}[k|k-1] &= A\hat{X}[k-1|k-1] + Bu[k-1],\\
P[k|k-1] &= AP[k-1|k-1]A^{\mathsf{T}} + FR_V[k-1]F^{\mathsf{T}}.
\end{aligned}
$$

Step 2: Correction. Correct the estimates and covariance matrix to account for the data taken at time step k:

$$
\begin{aligned}
\tilde{L}[k] &= P[k|k-1]C^{\mathsf{T}}(R_W + CP[k|k-1]C^{\mathsf{T}})^{-1},\\
\hat{X}[k|k] &= \hat{X}[k|k-1] - \tilde{L}[k](C\hat{X}[k|k-1] - Y[k]),\\
P[k|k] &= P[k|k-1] - \tilde{L}[k]CP[k|k-1].
\end{aligned}
$$

(We use $\tilde{L}[k]$ to distinguish the optimal gain in this form from that given in Theorem 7.2, as discussed briefly at the end of this section.)

Step 3: Iterate. Set k to $k+1$ and repeat steps 1 and 2.

Note that the correction step reduces the covariance by an amount related to the relative accuracy of the measurement, while the prediction step increases the covariance by an amount related to the process disturbance.

This form of the discrete-time Kalman filter is convenient because we can reason about the estimate in the case when we do not obtain a measurement on every iteration of the algorithm. In this case, we simply update the prediction step (increasing the covariance) until we receive new sensor data, at which point we call the correction step (decreasing the covariance).

The following lemma will be useful in the sequel:

Lemma 7.3. *The optimal gain $\tilde{L}[k]$ satisfies*

$$
\tilde{L}[k] = P[k|k]C^{\mathsf{T}}R_W^{-1}.
$$

Proof. $\tilde{L}[k]$ is defined as

$$
\tilde{L}[k] = P[k|k-1]C^{\mathsf{T}}(R_W + CP[k|k-1]C^{\mathsf{T}})^{-1}.
$$

Multiplying through by the inverse term on the right and expanding, we have

$$
\begin{aligned}
\tilde{L}[k](R_W + CP[k|k-1]C^{\mathsf{T}}) &= P[k|k-1]C^{\mathsf{T}},\\
\tilde{L}[k]R_W + \tilde{L}[k]CP[k|k-1]C^{\mathsf{T}} &= P[k|k-1]C^{\mathsf{T}},
\end{aligned}
$$

and hence

$$
\begin{aligned}
\tilde{L}[k]R_W &= P[k|k-1]C^{\mathsf{T}} - \tilde{L}[k]CP[k|k-1]C^{\mathsf{T}},\\
&= (I - \tilde{L}[k]C)P[k|k-1]C^{\mathsf{T}} = P[k|k]C^{\mathsf{T}}.
\end{aligned}
$$

The desired results follows by multiplying on the right by R_W^{-1}. □

It can be shown that the predictor-corrector form matches the form in Theorem 7.2 if we define $\hat{x}[k] = \hat{x}[k|k-1]$, $P[k] = P[k|k-1]$ and $\tilde{L}[k] = AL[k]$.

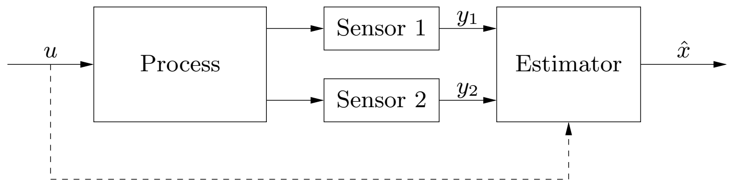


Figure 7.1: Sensor fusion. Multiple sensors report on data from a single process. The estimator fusions this information from the sensors to obtain an estimate of the state of the system. Depending on the use case, the input (dashed line) may not be available to the estimator.

7.4 Sensor Fusion

We now return to the main topic of the chapter: sensor fusion. Consider the situation described in Figure 7.1, where we have an input/output dynamical system with multiple sensors capable of taking measurements. The problem of sensor fusion involves deciding how to best combine the measurements from the individual sensors in order to accurately estimate the process state X. Since different sensors may have different noise characteristics, evidently we should combine the sensors in a way that places more weight on sensors with lower noise. In addition, in some situations we may have different sensors available at different times, so that not all information is available on each measurement update.

While sensor fusion can be used for estimation of the state of a system being controlled, another common application is to sense the state of a system in the environment. A difference for this use case is that the input to the system in the environment is often not available, requiring the estimator to use a model for the system in which the input is modeled by a random process.

Sensor weighting

To gain more insight into how the sensor data are combined, we investigate the functional form of $L[k]$. Suppose that each sensor takes a measurement of the form

$$Y^i = C^i X + W^i, \qquad i = 1, \dots, p,$$

where the superscript i corresponds to the specific sensor. Let W^i be a zero mean, white noise process with covariance $\sigma_i^2 = R_{W^i}(0)$. It follows from Lemma 7.3 that

$$L[k] = P[k|k] C^\mathsf{T} R_W^{-1}.$$

First note that if $P[k|k]$ is small, indicating that our estimate of X is close to the actual value (in the MMSE sense), then $L[k]$ will be small due to the leading $P[k|k]$ term. Furthermore, the characteristics of the individual sensors are contained in the different σ_i^2 terms, which only appears in R_W. Expanding the gain matrix, we have

$$L[k] = P[k|k] C^\mathsf{T} R_W^{-1}, \qquad R_W^{-1} = \begin{bmatrix} 1/\sigma_1^2 & & \\ & \ddots & \\ & & 1/\sigma_p^2 \end{bmatrix}.$$

We see from the form of R_W^{-1} that each sensor is inversely weighted by its covariance. Thus noisy sensors ($\sigma_i^2 \gg 1$) will have a small weight and require averaging over many iterations before their data can affect the state estimate. Conversely, if $\sigma_i^2 \ll 1$, the data is "trusted" and is used with higher weight in each iteration.

Information filters

An alternative formulation of the Kalman filter is to make use of the inverse of the covariance matrix, called the *information matrix*, to represent the error of the estimate. It turns out that writing the state estimator in this form has several advantages both conceptually and when implementing distributed computations. This form of the Kalman filter is known as the *information filter*.

We begin by defining the information matrix I and the weighted state estimate $\hat{Z}$:

$$I[k|j] = P^{-1}[k|j], \qquad \hat{Z}[k|j] = P^{-1}[k|j]\hat{X}[k|j].$$

In this form, it can be shown that the correction step of the Kalman filter for the multi-sensor case can be written as

$$\begin{aligned} I[k|k] &= I[k|k-1] + \sum_{i=1}^{p} (C^i)^\mathsf{T} R_{W^i}^{-1}[k|k] C^i, \\ \hat{Z}[k|k] &= \hat{Z}[k|k-1] + \sum_{i=1}^{p} (C^i)^\mathsf{T} R_{W^i}^{-1}[k|k] Y^i. \end{aligned}$$

The advantage of using the information filter version of the equation is that it allows a simple addition operation for the correction step, corresponding to adding the "information" obtained through the acquisition of new data. We also see the clear relationship between the information content in each sensor channel and the inverse covariance of that sensor.

Another feature of the information filter formulation is that it allows some efficiencies when implementing distributed estimation across networks. In particular, the information carried in the individual sensors can be simply added together through the updates of $I[k|k-1]$. This is helpful especially when the sensors have variable sampling rate and the measurement packets arrive at different times. New information is incorporated whenever it arrives and then a global update of $I[k|k–1]$ at a centralized node is used to integrate all sensor measurements (which can the be rebroadcast out to the sensors). The information form also makes clear how to handle missing data: if no data arrives for a given sensor then no information is added and only the time update is applied, hence the measurement update is skipped.

7.5 Implementation in Python

Steady state gains for a discrete time Kalman filter can be computed using the `dlqe` command:

```
L, P, E = ct.dlqe(A, F, C, Qv, Qw, C=C_sensors),
```

where `A`, `F`, and `C` are the (discrete-time) system matrices and `Qv` and `Qw` are the disturbance and noise covariance matrices.

The `create_estimator_iosystem` function can be used to implement a discrete-time Kalman filter in the form given in Theorem 7.2. This function creates an input/output system that implements a discrete-time state estimator of the form

$$\begin{aligned}\hat{x}[k+1] &= Ax[k] + Bu[k] - L(C\hat{x}[k] - y[k]),\\ P[k+1] &= APA^T + FQ_NF^T - APC^TR_\epsilon^{-1}CPA,\\ L &= APC^TR_\epsilon^{-1}, \qquad R_\epsilon = R_N + CPC^T.\end{aligned}$$

To create the estimator system, using the command

```
estim = ct.create_estimator_iosystem(sys, Qv, Qw)
```

where `sys` is the process dynamics and `Qv` and `Qw` are the covariance of the disturbance noise and sensor noise. The function returns the estimator `estim` as I/O system with a parameter `correct` that can be used to turn off the correction term in the estimation (for forward predictions).

Sensor fusion can be implemented by passing `create_estimator_iosystem` a system with full state output and then using the `C` keyword to define the measured values to be used by the estimator:

```
estim = ct.create_estimator_iosystem(sys, Qv, Qw, C=C_sensors)
```

The matrix `C_sensors` can have as many rows as there are sensors, with the matrix `Qw` describing the noise covariance (and cross-correlation, if diagonal entries are non-zero).

As described in Section 6.4, the `create_statefbk_iosystem` function can be used to create an I/O system that includes an estimator and state feedback to form a dynamic compensator.

7.6 Additional Topics

Unscented Kalman filter[1]

Consider a *nonlinear* system

$$\begin{aligned}\dot{X} &= f(X, u, V), \qquad && X \in \mathbb{R}^n, u \in \mathbb{R}^m,\\ Y &= h(X) + W, && Y \in \mathbb{R}^p,\end{aligned} \tag{7.10}$$

where V and W are Gaussian white noise processes with covariance matrices R_V and R_W. We saw in Chapter 6 how to extend the Kalman filter to this case by using the local linearization of the dynamics. In some cases this approximation is not a very good one, especially if there are strong nonlinearities in the update function f or output function h.

ChatGPT prompt: Explain how an unscented Kalman filter works, in the form of a textbook.

[1] This section was written by ChatGPT on 20 Feb 2023 and (lightly) edited by Richard Murray.

Response: The unscented Kalman filter (UKF) is a variant of the Kalman filter that can be used for nonlinear systems. In this [subsection], I'll assume that you're already familiar with the Kalman filter and its notation.

The basic idea behind the UKF is to approximate the probability distribution of the state variables using a set of sample points, called sigma points. These sample points are carefully chosen to capture the mean and covariance of the distribution, and are propagated through the nonlinear system to obtain a predicted distribution for the next time step.

At each time step, the UKF goes through two main steps: the prediction step and the update step.

Prediction step. In the prediction step, the UKF generates a set of sigma points from the current estimate of the state distribution. These sigma points are chosen to capture the mean and covariance of the distribution. The number of sigma points required is typically twice the number of state variables plus one.

Once the sigma points are generated, they are propagated through the nonlinear system dynamics to obtain a predicted mean and covariance for the next time step. This is done by passing each sigma point through the nonlinear function, and then computing the mean and covariance of the resulting set of points.

The predicted mean and covariance are then used to generate a predicted distribution for the state variables at the next time step.

Update step. In the update step, the UKF uses new measurements of the system to correct the predicted distribution generated in the prediction step.

First, a set of sigma points is generated from the predicted distribution, in the same way as in the prediction step. These sigma points are then propagated through the measurement function to obtain the predicted measurement values corresponding to each sigma point.

The predicted measurement mean and covariance are then computed from the resulting set of predicted measurement values. This predicted measurement distribution is compared to the actual measured values, and the difference is used to correct the predicted state distribution.

Finally, the corrected state distribution is used as the new estimate for the state variables at the current time step.

In summary, the UKF approximates the distribution of the state variables using a set of carefully chosen sample points, and propagates these points through the nonlinear system dynamics and measurement function to generate predicted distributions. The predicted distributions are then updated with new measurements to obtain a corrected estimate of the state variables at each time step.

ChatGPT prompt: Show me the equations for an unscented Kalman filter.

1. Initialization: initialize state estimate and covariance matrix:
    - $\hat{x}_0$: initial state estimate
    - P_0 : initial covariance matrix
2. Prediction step:
    - Generate sigma points:

- $X_0 = \hat{x}_k$,
- $X_i = \hat{x}_k + \left(\sqrt{(n+\lambda)P_k}\right)_i, i = 1, ..., n$,
- $X_i = \hat{x}_k - \left(\sqrt{(n+\lambda)P_k}\right), i = n+1, ..., 2n$.

- Propagate sigma points through nonlinear function $f(x)$:
  - $Y_i = f(X_i), i = 0, 1, ..., 2n$.
- Compute predicted state mean and covariance:
  - $\hat{x}_{k|k-1} = \sum_{i=0}^{2n} w_i^m Y_i$,
  - $P_{k|k-1} = \sum_{i=0}^{2n} w_i^c (Y_i - \hat{x}_{k|k-1})(Y_i - \hat{x}_{k|k-1})^T + Q_k$,

  where n is the dimensionality of the state, λ is a scaling parameter, P_k is the covariance matrix of the state at time k, w_i^m and w_i^c are weight coefficients that depend on n and λ, and Q_k is the process noise covariance matrix.

3. Update step:

- Generate sigma points from predicted state distribution:
  - $X_0 = \hat{x}_{k|k-1}$,
  - $X_i = \hat{x}_{k|k-1} + \left(\sqrt{(n+\lambda)P_{k|k-1}}\right)_i, i = 1, ..., n$,
  - $X_i = \hat{x}k|k-1 - \left(\sqrt{(n+\lambda)P_{k|k-1}}\right)_{i-n}, i = n+1, ..., 2n$.
- Propagate sigma points through measurement function $h(x)$:
  - $Z_i = h(X_i), i = 0, 1, ..., 2n$.
- Compute predicted measurement mean and covariance:
  - $\hat{z}_{k|k-1} = \sum_{i=0}^{2n} w_i^m Z_i$,
  - $S_k = \sum_{i=0}^{2n} w_i^c (Z_i - \hat{z}k|k-1)(Z_i - \hat{z}k|k-1)^T + R_k$,

  where R_k is the measurement noise covariance matrix.
- Compute cross-covariance matrix:
  - $P_{x,z,k} = \sum_{i=0}^{2n} w_i^c (X_i - \hat{x}k|k-1)(Z_i - \hat{z}k|k-1)^T$.
- Compute Kalman gain:
  - $K_k = P_{x,z,k} S_k^{-1}$.

  Compute updated state estimate and covariance:
  - $\hat{x}k = \hat{x}k|k-1 + K_k(z_k - \hat{z}_{k|k-1})$,
  - $P_k = P_{k|k-1} - K_k S_k K_k^T$.

  where z_k is the measurement at time k, $\hat{z}_{k|k-1}$ is the predicted measurement, K_k is the Kalman gain, $P_{x,z,k}$ is the cross-covariance matrix, and P_k is the updated covariance matrix.

The weight coefficients are defined as follows:

- $w_0^m = \frac{\lambda}{n+\lambda}$,
- $w_0^c = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta)$,
- $w_i^m = w_i^c = \frac{1}{2(n+\lambda)}, i = 1, ..., 2n$,

where α, β, and κ are tuning parameters that determine the spread of the sigma points. Typically, $\alpha = 0.9$, $\beta = 2$, and $\kappa = 3 - n$ are used in practice.

Ensemble Kalman filtering[2]

ChatGPT prompt: Write a section for a textbook on the ensemble Kalman filter, with equations.

Introduction. The ensemble Kalman filter (EnKF) is a popular data assimilation algorithm that is widely used in numerical weather prediction, oceanography, and other fields. It is a variant of the Kalman filter that replaces the Gaussian assumptions about the prior and posterior probability distributions with a set of Monte Carlo samples, or "ensemble members". The EnKF is particularly useful in high-dimensional problems where the computational cost of a full Kalman filter update becomes prohibitive.

Algorithm. The EnKF algorithm proceeds as follows:

1. Initialization: Generate an initial ensemble of N state vectors x_0^i, $i = 1, \dots, N$, with mean $\hat{x}_0$ and covariance matrix P_0.

2. Forecast step: Perturb each ensemble member using a random draw from a distribution with mean zero and covariance matrix Q_k, the process noise:

    - $x_k^i = f(x_{k-1}^i) + \epsilon_k^i$

    Evaluate the forecast ensemble mean and covariance:

    - $\hat{x}_{k|k-1} = \frac{1}{N} \sum_{i=1}^N x_k^i$
    - $P_{k|k-1} = \frac{1}{N-1} \sum_{i=1}^N (x_k^i - \hat{x}_{k|k-1})(x_k^i - \hat{x}_{k|k-1})^T + Q_k$

3. Analysis step: Obtain a set of M observations y_k and the associated observation operator h.

    - Perturb each ensemble member using a random draw from a distribution with mean zero and covariance matrix R_k, the observation error:
        - $y_k^i = h(x_k^i) + \delta_k^i$

      Evaluate the forecast ensemble mean and covariance:
        - $\hat{y}_{k|k-1} = \frac{1}{M} \sum_{i=1}^M y_k^i$
        - $S_k = \frac{1}{M-1} \sum_{i=1}^M (y_k^i - \hat{y}_{k|k-1})(y_k^i - \hat{y}_{k|k-1})^T + R_k$
    - Compute the cross-covariance matrix between the forecast state and the observations:
        - $P_{k|k-1} = \frac{1}{N-1} \sum_{i=1}^N (x_k^i - \hat{x}_{k|k-1})(y_k^i - \hat{y}_{k|k-1})^T$
    - Compute the Kalman gain:
        - $K_k(y_k - \hat{y}_{k|k-1})$
    - Update the state ensemble using the Kalman gain:
        - $x_k^i = x_{k|k-1}^i + K_k(y_k - h(x_{k|k-1}^i))$
    - Evaluate the updated ensemble mean and covariance:
        - $\hat{x}_k = \frac{1}{N} \sum_{i=1}^N x_k^i$

[2]This section was written by ChatGPT on 20 Feb 2023 and (lightly) edited by Richard Murray.

– $P_k = \frac{1}{N-1}\sum_{i=1}^{N}(x_k^i - \hat{x}_k)(x_k^i - \hat{x}_k)^T$

where ϵ_k^i and δ_k^i are random vectors drawn from the process and observation noise distributions, respectively.

The EnKF algorithm also requires specification of the process noise covariance matrix Q_k and the observation error covariance matrix R_k. These matrices can be estimated using statistical methods or by tuning them through trial and error.

One advantage of the EnKF over the traditional Kalman filter is that it is easily parallelizable. The forecast step and analysis step can be performed independently for each ensemble member, allowing the algorithm to take advantage of distributed computing architectures. However, the EnKF can suffer from sampling errors, particularly in the presence of nonlinearities or non-Gaussian distributions. Various modifications to the basic EnKF algorithm have been proposed to address these issues, such as the Local ensemble transform Kalman filter (LETKF) and the ensemble square root filter (ESRF).

Moving horizon estimation

The various extensions and variations of the Kalman filter equations all assume that the system disturbances and measurement noise can be modeled by Gaussian processes, in which the (potentially time-varying) mean and covariance of the signals are sufficient to model the underlying sources of uncertainty. In many cases, this is not a good assumption, for example when there are bounded, non-symmetric, or multi-modal sources of disturbances and noise.

Consider a nonlinear system with discrete time dynamics of the form

$$X[k+1] = f(X[k], u[k], V[k]), \qquad Y[k] = h(X[k]) + W[k], \tag{7.11}$$

where $X[k] \in \mathbb{R}^n$, $u[k] \in \mathbb{R}^m$, and $Y[k] \in \mathbb{R}^p$, and $V[k] \in \mathbb{R}^q$ and $W[k] \in \mathbb{R}^p$ represent random processes that are not necessarily Gaussian white noise processes. The estimation problem that we wish to solve is to find the estimate $\hat{x}[\cdot]$ that matches the measured outputs $y[\cdot]$ with "likely" disturbances and noise.

For a fixed horizon of length N, this problem can be formulated as an optimization problem where we define the likelihood of a given estimate (and the resulting noise and disturbances predicted by the model) as a cost function. Suppose we model the likelihood using a conditional probability density function $p\big(x[0], \dots, x[N] \mid y[0], \dots, y[N-1]\big)$. Then we can pose the state estimation problem as

$$\hat{x}[0], \dots, \hat{x}[N] = \arg \max_{\hat{x}[0],\dots,\hat{x}[N]} p\big(\hat{x}[0], \dots, \hat{x}[N] \mid y[0], \dots, y[N-1]\big) \tag{7.12}$$

subject to the constraints given by equation (7.11). The result of this optimization gives us the estimated state for the previous N steps in time, including the "current" time $x[N]$. The basic idea is thus to compute the state estimate that is most consistent with our model and penalize the noise and disturbances according to how likely the are (based on a some sort of stochastic system model for each).

Given a solution to this fixed horizon, optimal estimation problem, we can create an estimator for the state over all times by applying repeatedly applying the optimization problem (7.12) over a moving horizon. At each time k, we take the measurements for the last N time steps along with the previously estimated state at the start of the horizon, $x[k-N]$ and reapply the optimization in equation (7.12). This approach is known as a *moving horizon estimator* (MHE).

The formulation for the moving horizon estimation problem is very general and various situations can be captured using the conditional probability function $p(x[0], \dots, x[N] \mid y[0], \dots, y[N-1]$. We start by noting that if the disturbances are independent of the underlying states of the system, we can write the conditional probability as

$$p\bigl(x[0], \dots, x[N] \mid y[0], \dots, y[N-1]\bigr) = \\ p_{X[0]}(x[0]) \prod_{k=0}^{N-1} p_V\bigl(y[k] - h(x[k])\bigr)\, p\bigl(x[k+1] \mid x[k]\bigr).$$

This expression can be further simplified by taking the log of the expression and maximizing the function

$$\log p_{X[0]}(x[0]) + \sum_{k=0}^{N-1} \log p_W\bigl(y[k] - h(x[k])\bigr) + \log p_V(v[k]). \tag{7.13}$$

The first term represents the likelihood of the initial state, the second term captures the likelihood of the noise signal, and the final term captures the likelihood of the disturbances.

If we return to the case where V and W are modeled as Gaussian processes, then it can be shown that maximizing equation (7.13) is equivalent to solving the optimization problem given by

$$\min_{x[0], \{v[0], \dots, v[N-1]\}} \|x[0] - \bar{x}[0]\|_{P_0^{-1}} + \sum_{k=0}^{N-1} \|y[k] - h(x_k)\|^2_{R_W^{-1}} + \|v[k]\|^2_{R_V^{-1}}. \tag{7.14}$$

Note that here we only compute the estimated initial state $\hat{x}[0]$, but we can now reconstruct the entire history of estimated states using the system dynamics:

$$\hat{x}[k+1] = F(\hat{x}[k], u[k], v[k]), \quad k = 0, \dots, N-1,$$

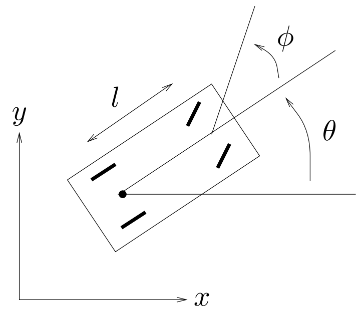
and we can implement the estimator in receding horizon fashion by repeatedly solving the optimization of a window of length N backwards in time.

One of the simpler cases where the moving horizon formulation is useful is when we have *a priori* knowledge that our disturbances are bounded. In this case, we simply add a constraint in the optimization in equation (7.14), for example requiring that $v[k] \in [v_{\min}, v_{\max}]$.

This functionality is implemented in python-control using the `solve_oep()` and `create_mhe_iosystem()` functions. An example demonstrating the implementation is available via the course website.

Exercises

7.1. Consider the problem of estimating the position of an autonomous mobile vehicle using a GPS receiver and an IMU (inertial measurement unit). The continuous time dynamics of the vehicle are given by



$$\dot{x} = \cos\theta\, v$$
$$\dot{y} = \sin\theta\, v$$
$$\dot{\theta} = \frac{1}{\ell}\tan\delta\, v,$$

We assume that the vehicle is disturbance free, but that we have noisy measurements from the GPS receiver and IMU and an initial condition error.

(a) Rewrite the equations of motion in discrete time, assuming that we update the dynamics at a sample time of $h = 0.005$ sec and that we can take $\dot{x}$ to be roughly constant over that period. Run a simulation of your discrete time model from initial condition $(0, 0, 0)$ with constant input $\delta = \pi/8$, $v = 5$ and compare your results with the continuous time model.

(b) Suppose that we have a GPS measurement that is taken every 0.1 seconds and an IMU measurement that is taken every 0.01 seconds. Write a MATLAB program that that computes the discrete time Kalman filter for this system, using the same disturbance, noise and initial conditions as Exercise 6.5.

7.2. Consider the problem of estimating the position of a car operating on a road whose dynamics are modeled as described in Example 2.3. We assume that the car is executing a lane change manuever and we wish to estimate its position using a set of available sensors:

- A stereo camera pair, which relatively poor longitudinal (x) accuracy but good lateral position (y) accuracy. We model the covaraiance of the sensor noise as $R_{\text{lat}} = \text{diag}(1, 0.1)$.
- An automotive grade radar, which has good longitudinal position (x) accuracy but poor lateral (y) accuracy, with $R_{\text{lon}} = \text{diag}(0.1, 1)$.
- We assume the radar can also measure the longitudinal velocity ($\dot{x}$) as an optional measurement, with $R_{\text{vel}} = 1$.

In this problem we assume that the detailed model of the system is not known and also that the inputs to the vehicle (throttle and steering) are not known. We use a variety of system models to explore how these different measurements can be fused to obtain estimates and predictions of the vehicle position.

(a) Consider a model of the vehicle consisting of a particle in 2D, with the velocity of particle in the x and y direction taken as the input:

$$\dot{x} = u_1, \qquad \dot{y} = u_2$$

A discrete-time version of the system dynamics is given by

$$x[k+1] = x[k] + u_1[k] * T_{\text{s}}, \qquad y[k+1] = y[k] + u_2[k] * T_{\text{s}},$$

where $T_s = 0.1$ s is the sampling time between sensor measurements.

Construct an estimator for the system using a combination of the stereo pair and the radar (position only). Estimate the state and covariance of the system during the lane change manuever from Example 2.3 and predict the state for the next 4 seconds.

(b) Assume now that we now add (noisy) measurement of the velocity from the radar as an approximation of the input u_1. Update your Kalman filter to utilize this measurement (with no filtering), and replot the estimate and prediction for the system.

(c) To provide a better prediction, we can increase the complexity of our model so that it includes the velocity of the vehicle as a state, allowing us to model the acceleration as the input. In continuous time, this model is given by

$$\ddot{x} = u_1, \qquad \dot{y} = u_2$$

(note that we are still modeling the lateral position using a single integrator).

Convert this model to discrete time and construct an estimator for the system using a combination of the stereo pair and the radar (position and velocity). Estimate the state and covariance of the system during the lane change manuever and predict the state for the next 4 seconds.

Note: in this problem you have quite a bit of freedom in how you model the disturbances, which should model the unknown inputs to the vehicle being observed. Make sure to provide some level of justification for how you chose these disturbances.

7.3. The form of the optimal feedback for a discrete time Kalman filter differes slightly depending on whether we use the form in Theorem 7.2 or the predictor-corrector form in Section 7.3.

(a) Show that the predictor-corrector form of the optimal estimator for a linear process driven by white noise matches the form in Theorem 7.2 if we define $\hat{x}[k] = \hat{x}[k|k-1]$, $P[k] = P[k|k=1]$ and $\tilde{L}[k] = AL[k]$.

(b) Alternatively, show that if we formulate the optimal estimate using an estimator of the form

$$\hat{X}[k+1] = A\hat{X}[k] + L[k](Y[k+1] - CA\hat{X}[k])$$

that we recover the update law in the predictor-corrector form.

7.4. The unscented Kalman filter (UKF) equations created by ChatGPT are convincing, but they are incorrect. Find and fix the errors.

7.5. Consider the discrete time linear system

$$X[k+1] = \begin{bmatrix} 0.5 & 1 & 0 \\ 0 & -0.8 & 1 \\ 0 & 0 & 0.5 \end{bmatrix} X[k] + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} V, \qquad Y[k] \quad = \sin(\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} X) + W,$$

where V is a discrete time, white noise process with covariance 0.01 and W is a discrete time, white noise process with covariance 10^{-4}. Let $u[k] = \sin(2\pi k/5)$ and assume that $P[0] = \mathbb{E}(X[0]X^{\mathsf{T}}[0]) = 0.5I$.

(a) Construct an optimal estimator (Kalman filter) for the system linearized about the origin and plot the state estimate and covariance of each state (using error bars) for a trajectory starting at the origin and for a duration of $K = 20$.

(b) Construct a moving horizon estimator for the system using time windows of length $N = 1$, $N = 3$, and $N = 6$ and compare the state estimate to the result from (a).

(c) Change the penalty on the initial state in the window from $P[0]$ to the value of P obtained from the steady state estimator for the linearized system and compare the performance for the three horizon lengths in (b).

(d) Suppose that the disturbance V is constrained to take values in the the range -0.1 to 0.1. Compare the steady state optimal estimator for the linearized system to a moving horizon estimator with appropriate horizon that takes the constraints into account. How does the moving horizon estimator compare to the linearized estimator?