3

Estimators and Filters for Indoor Positioning

Indoor positioning and navigation rely on robust state estimation to determine location from sensor data. This chapter reviews key estimators and filters used in indoor positioning, ranging from classical least squares methods to modern probabilistic filters and optimization frameworks. We discuss both the theoretical foundations and practical considerations of each approach, including their background, mathematical derivations, and applications in indoor positioning. Emphasis is placed on understanding how each method works, along with its advantages and limitations. These methods have evolved from early analytical solutions, such as least squares estimation, to optimal recursive estimators like the Kalman filter (introduced in 1960), as well as more advanced approaches such as particle filters and factor graph optimization (FGO), which leverage modern computing. The remainder of this chapter is organized as follows. Section 3.1 introduces least squares estimation. Section 3.2 discusses the Kalman filter and its nonlinear extensions, including the extended Kalman filter (EKF), iterated EKF (IEKF), and unscented Kalman filter (UKF). Section 3.3 presents particle filtering methods. Section 3.4 covers FGO for smoothing. Section 3.5 provides a comparative analysis of these approaches, while Section 3.6 concludes the chapter with final insights and future research directions.

3.1 Least Squares Estimation

Least-squares estimation is a fundamental technique for determining an unknown parameter (or set of parameters) by minimizing the sum of squared differences between observed values and the values predicted by a model. Suppose we have m measurements $y_1, \ldots, y_m$ related to an unknown parameter vector $\mathbf{x} \in \mathbb{R}^n$ through the model $y_i = h_i(\mathbf{x}) + \epsilon_i$, where ϵ_i represents measurement noise. In the least-squares estimation, we seek the estimate $\hat{\mathbf{x}}$ that minimizes the cost function shown below.

$$J(\mathbf{x}) = \sum_{i=1}^{m} \left(y_i - h_i(\mathbf{x}) \right)^2 \tag{3.1}$$

In other words, the total squared error between the model predictions $h_i(\mathbf{x})$ and the observations y_i. This method originated from the work of Gauss and Legendre on fitting astronomical data and remains a staple due to its simplicity and optimality under Gaussian noise assumptions.

In many indoor positioning problems, least-squares serves as a fundamental starting point. For example, determining a user's 2-D position from distance measurements to Wi-Fi APs or UWB beacons can be formulated as a least-squares problem: the unknown position $\mathbf{x} = (x,y)$ is estimated by minimizing the squared errors between the measured distances and the distances predicted based on an assumed position. Least-squares is also widely used as an underlying solver for calibration tasks, such as determining sensor biases, and frequently functions as a subroutine within more complex estimation frameworks.

If the ith measurement model $h_i(\mathbf{x})$ is linear in $\mathbf{x}$, that is, $h_i(\mathbf{x}) = \mathbf{h}_i^T \mathbf{x}$ for some known vector $\mathbf{h}_i$, then the least-squares solution can be obtained in a closed form. All vectors $\mathbf{h}_i^T$ can be stacked to form an $m \times n$ matrix $\mathbf{H}$. Let $\mathbf{y} = [y_1, \ldots, y_m]^T$. The cost function then becomes $J(\mathbf{x}) = \|\mathbf{y} - \mathbf{H}\mathbf{x}\|^2$. Setting the gradient to 0 yields the normal equations

$$\mathbf{H}^T \mathbf{H} \hat{\mathbf{x}} = \mathbf{H}^T \mathbf{y} \tag{3.2}$$

assuming $\mathbf{H}$ has full column rank. The solution is

$$\hat{\mathbf{x}} = \left(\mathbf{H}^T \mathbf{H} \right)^{-1} \mathbf{H}^T \mathbf{y} \tag{3.3}$$

which is the well-known ordinary least-squares estimator. This solution minimizes $J(\mathbf{x})$ and represents the maximum likelihood estimate if the noise ϵ_i is independent, zero-mean Gaussian. Moreover, it is an unbiased estimator and achieves the lowest possible variance among all linear unbiased estimators, as stated by the Gauss-Markov theorem.

When the model $h_i(\mathbf{x})$ is nonlinear (as is common in positioning, e.g., $h_i(\mathbf{x}) = \sqrt{(x - x_i)^2 + (y - y_i)^2}$ for distance to a known anchor at (x_i, y_i)), no closed-form solution exists. Instead, one must solve

$$\sum_{i=1}^{m}\left(y_i - h_i(\mathbf{x})\right)\nabla h_i(\mathbf{x}) = 0 \tag{3.4}$$

iteratively. $\nabla h_i(\mathbf{x})$ is the gradient of $h_i(\mathbf{x})$. A common approach is the Gauss-Newton method: start from an initial guess $\mathbf{x}_0$, and iteratively update $\mathbf{x}$ by linearizing $h_i(\mathbf{x})$ around the current estimate. At iteration t, one computes the Jacobian $J_t = \partial h/\partial x|_{\mathbf{x}_t}$ and residual $\mathbf{r}_t = \boldsymbol{y} - h(\mathbf{x}_t)$, then solves the linear least-squares $J_t\Delta\mathbf{x} \approx \mathbf{t}_t$. The estimate is updated as $\mathbf{x}_{t+1} = \mathbf{x}_t + \Delta\mathbf{x}$. This process repeats until convergence. The Gauss-Newton iteration effectively solves a sequence of linear approximations to find the minimum of $J(\mathbf{x})$. In practice, convergence to the global minimum is only guaranteed if the initial guess is close to the true solution and the cost function is relatively well-behaved (convex in a large neighborhood). Chapter 4 will introduce point positioning based on this iterative least square in detail.

3.1.1 Robust Estimators

There are several extensions of basic least squares that improve their robustness and applicability. Weighted least squares assigns a weight w_i to each residual in the cost: $J(\mathbf{x}) = \sum_i w_i(y_i - h_i(\mathbf{x}))^2$. This is useful when measurements have different noise variances; setting $w_i = 1/\sigma_i^2$ (the inverse of noise variance) yields the best linear unbiased estimate in the linear case. Robust least squares modifies the cost function to reduce the influence of outliers by using alternative loss functions. This is particularly important in indoor environments, where some measurements may be outliers due to nonline-of-sight (NLOS) conditions. To avoid the influence of these large-biased measurements, a robust estimator is a popular means. Table 3.1 shows a few popular robust estimators. Their pros and cons are discussed in [1].

If there is a measurement with a large bias, it means that its residual is big. As mentioned earlier, the estimation is achieved when the "error function" is minimized. A robust estimator is a function to bridge the residual function to a cost function that can mitigate the impact of the measurements with a large bias. Figure 3.1 shows the idea. The x-axis denotes the

Table 3.1
Popular Robust Estimators

Robust Estimator	**Error Function**
L2	$e(\mathbf{x}) = \frac{1}{2}\lVert \mathrm{r}(\mathbf{x}) \rVert^2$
Cauchy	$e(\mathbf{x}) = \frac{1}{2}\ln\left(1 + \lVert \mathrm{r}(\mathbf{x}) \rVert^2\right)$
Huber	$e(\mathbf{x}) = \begin{cases} \frac{1}{2}\lVert \mathrm{r}(\mathbf{x}) \rVert^2, & \lvert \mathrm{r}(\mathbf{x}) \rvert \le \delta \\ \delta\left(\lvert \mathrm{r}(\mathbf{x}) \rvert - \frac{1}{2}\delta\right), & \text{otherwise} \end{cases}$
G-M	$e(\mathbf{x}) = \frac{1}{2}\frac{\lVert \mathrm{r}(\mathbf{x}) \rVert^2}{1 + \lVert \mathrm{r}(\mathbf{x}) \rVert^2}$

measurement error, which is the error between the actual measurement and expected measurement from converting the true state into the measurement domain. The *y*-axis denotes the costs from the measurement errors based on different robust estimators, which is expected to be minimized during the estimation. For example, the measurement with a higher cost has a higher

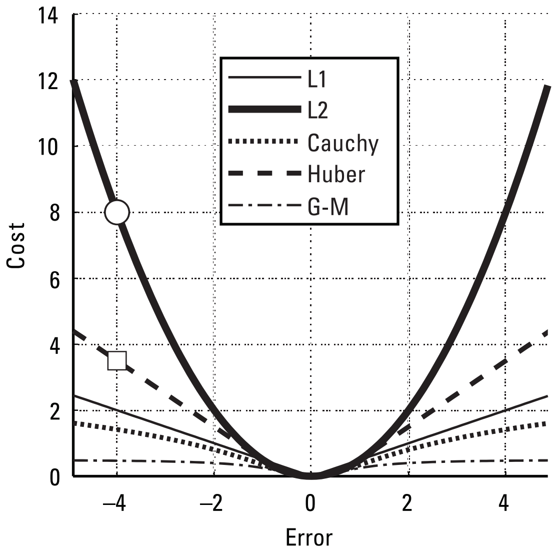


Figure 3.1 Cost of different robust functions.

priority to minimize its corresponding residual (i.e., has more impact during the estimation). The circle and square markers denote the cost calculated from a large-biased measurement using the L2 estimator and Huber estimator, respectively. The corresponding cost is large for the L2 estimator, resulting in a significant influence from this large-biased measurement during estimation. However, the cost when using the Huber estimator (a popular robust estimator) is much lower, in which the large-biased measurement will not affect the estimation too much (i.e., more robust against outliers). However, the robust estimator must be carefully tuned since the definition of a healthy measurement might not be the same for different IPIN applications. A state-of-the-art research proposed to adaptively tune the parameters G-M can be found at [2].

Another extension is regularized least squares, which adds a penalty term ($\lambda\|\mathbf{x}\|^2$) to encourage smaller parameter norms. This approach is commonly used in ill-posed problems to improve numerical stability and prevent overfitting.

The appeal of least squares lies in its simplicity and optimality under ideal conditions. It provides a closed-form solution for linear models and can be efficiently computed even with large numbers of measurements. Least-squares estimates are statistically optimal, achieving minimum-variance unbiased estimates when errors are Gaussian and independent. Furthermore, least-squares solutions are unique and globally optimal if the problem is convex, which holds for linear models. However, least-squares methods also have significant limitations. They are highly sensitive to outliers, as a single erroneous measurement can substantially skew the result due to the squared error penalty. Nonlinear least squares can converge to a local minimum if the initial guess is poor or if the cost surface has multiple minima, a common issue in complex indoor environments, where ambiguous distances or symmetries may arise. Additionally, least squares assume that the model form is correct; systematic errors, such as measurement bias or misrepresented physics, can lead to biased estimates. In dynamic scenarios, a simple least-squares approach that processes all data in a batch may not be suitable for real-time updates. Despite these limitations, least-squares remains a foundational technique and often serves as a starting point for more advanced filtering approaches, which are discussed later in this chapter.

3.2 Kalman Filters and Extensions

The Kalman filter is a recursive algorithm that provides optimal state estimates for linear dynamical systems with Gaussian noise. Originally formulated by

R. E. Kalman in 1960 [1], it has become a cornerstone of tracking and sensor fusion in navigation. We first describe the standard linear Kalman filter and then introduce several extensions designed for nonlinear systems.

3.2.1 Linear Kalman Filter

The Kalman filter (KF) is the most popular sensor fusion algorithm applied in navigation, particularly because it can achieve optimal performance, in this case, a maximum a posterior (MAP) estimation, under certain assumptions. During the early days of computing, such as in the 1960s, when computational power was very limited, directly applying MAP was impractical due to the exponential increase in computational load with an increasing number of time frames. An ideal solution is to estimate the state of the current time frame using a recursive approach, which fully uses the information from all previous time frames. The MAP estimation for this example can be represented as follows:

$$\hat{\chi}_{\mathrm{MAP}} = \arg\max_{\chi}\left(P\left(\chi|\mathbf{Z},\mathbf{U}\right)\right) \tag{3.5}$$

where $P(\chi|\mathbf{Z},\mathbf{U})$ is the classic posterior probability of $\chi = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\}$ given the measurement $\mathbf{Z} = \{\mathbf{z}_0, \mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_k\}$ and control input $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k\}$. The control input describes how the system moves from one state to the next. Equation (3.5) implies that MAP is achieved when a set of state vectors χ (across all time frames) can be found that satisfies the sets of conditions defined by the control input and measurement vectors (again, across all time frames). To decouple (3.5), the Bayes rule is employed, leading to the following equation.

$$P\left(\chi|\mathbf{Z},\mathbf{U}\right) \propto \prod_{k} P\left(\mathbf{z}_k|\mathbf{x}_k\right) P\left(\mathbf{x}_0\right) \prod_{k} P\left(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_{k-1}\right) \tag{3.6}$$

where $\prod_k P(\mathbf{z}_k|\mathbf{x}_k)$ represents the joint probability of the likelihoods $P(\mathbf{z}_k|\mathbf{x}_k)$, and $\prod_k P(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_{k-1})$ denotes the joint probability of the propagation probabilities $P(\mathbf{x}_k|\mathbf{x}_{k-1,}\mathbf{u}_{k-1})$. Additionally, $P(\mathbf{x}_0)$ is the prior probability. It is evident that MAP estimation accounts for the states, measurements, and propagation across all time frames, which can result in a significant computational load. The MAP estimate can then be obtained using the following equation.

$$P\left(\chi|\mathbf{Z},\mathbf{U}\right) \xleftrightarrow{\text{with assumptions}} P\left(\mathbf{x}_k|\mathbf{z}_k,\mathbf{u}_k\right) \propto P\left(\mathbf{z}_k|\mathbf{x}_k\right)\, P\left(\mathbf{x}_k|\mathbf{z}_{k-1},\mathbf{u}_k\right) \tag{3.7}$$

where $P(\mathbf{z}_k|\mathbf{x}_k)$ represents the likelihood of the current kth time frame and $P(\mathbf{x}_k|\mathbf{x}_{k-1,}\mathbf{u}_{k-1})$ is the prior probability of the state at the current time frame, based on the current propagation and the previous measurement. This conceptualization is realized through the Kalman filter.

To achieve (3.7), it is necessary to represent the information from all previous time frames using only the most recent time frame. The Kalman filter accomplishes this by leveraging the concept of uncertainty, represented by the state noise covariance matrix **p**. However, this is only possible under the following three essential assumptions. Any violation of these assumptions compromises the optimality of the Kalman filter:

- *Assumption 3.1:* The first essential assumption of the Kalman filter is linearity. This assumption states that both the system being modeled and the measurement process exhibit linear behavior. In other words, the relationship between the system's state and its measurements can be accurately described using linear equations. This implies that the system dynamics and the measurement process can be represented by matrices and vectors that undergo linear transformations. By assuming linearity, the Kalman filter leverages the principles of linear algebra and estimation theory to estimate the system's state and its uncertainty. It uses a linear model to propagate the state forward in time and update it based on new measurements. However, if the system or measurement process deviates significantly from linearity, the Kalman filter may produce suboptimal results or fail to provide accurate estimates.
- *Assumption 3.2:* The second crucial assumption of the Kalman filter is that the noise in both the system dynamics and measurements follows a Gaussian (normal) distribution. This assumption enables the Kalman filter to make probabilistic inferences and compute optimal estimates. By assuming Gaussian noise, the Kalman filter effectively manages uncertainty and refines state estimates using measurement data. It leverages the properties of the Gaussian distribution, particularly its mean and covariance, to determine the optimal weights for integrating predicted states with actual measurements. This Gaussian assumption is essential because deviations from Gaussian characteristics, such as non-Gaussian or heavy-tailed noise distributions, can introduce inaccuracies into the estimation process. Such deviations may significantly impact the accuracy and optimality of the Kalman filter's estimates.
- *Assumption 3.3:* The third crucial assumption of the Kalman filter is stationarity. This implies that both the system dynamics and the statistical properties of the noise remain consistent over time. In essence, the system being modeled is assumed to be time-invariant. Under the

stationarity assumption, the Kalman filter operates with the premise that the system's behavior remains relatively unchanged during the estimation process. This allows the filter to use a consistent set of parameters and matrices to characterize the system dynamics. Additionally, it assumes that the statistical properties of the noise, including its mean and covariance, remain constant throughout the estimation process. The first assumption of linearity enables the use of matrices to describe uncertainty, which can be calculated digitally. The second assumption, involving Gaussian noise, facilitates the application of statistical tools to handle the mean and covariance in describing independent probability density functions. With these two assumptions in place, the measurement model can be expressed as:

$$\mathbf{z}_k^{\text{measurement}} = \mathbf{H}_k \mathbf{x}_k + \mathbf{w}_{z,k}, \ \mathbf{w}_{z,k} \in N\left(0, \sigma_{z,k}\right) \tag{3.8}$$

where $N(0, \sigma)$ represents a zero-mean Gaussian distribution with a standard deviation σ. In (3.8), $\mathbf{H}_k$ represents the measurement model, and $\mathbf{w}_{z,k}$ represents the Gaussian random noise associated with the measurement. The mean and covariance of the likelihood can be calculated as follows:

$$\begin{aligned} \mathrm{E}\left[P\left(\mathbf{z}_k | \mathbf{x}_k\right)\right] &= \mathbf{H}_k \mathbf{x}_k \\ \mathrm{Cov}\left[P\left(\mathbf{z}_k | \mathbf{x}_k\right)\right] &= \mathbf{\Sigma}_{\mathbf{w}_{z,k}} \end{aligned} \tag{3.9}$$

Consequently, the likelihood can be described by the following Gaussian distribution,

$$P\left(\mathbf{z}_k | \mathbf{x}_k\right) \propto -\exp\left(\left(\mathbf{z}_k - \mathbf{H}_k \mathbf{x}_k\right)^T \mathbf{\Sigma}_{\mathbf{w}_{z,k}}^{-1} \left(\mathbf{z}_k - \mathbf{H}_k \mathbf{x}_k\right)\right) \tag{3.10}$$

Similarly, the propagation model can be expressed as:

$$\begin{aligned} &\mathbf{x}_{k,k-1}^{\text{propagation}} = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{w}_{x_{k-1}} + \mathbf{u}_k + \mathbf{w}_{u,k} \\ &\mathbf{w}_{x_{k-1}} \in N\left(0, \sigma_{x_{k-1}}\right) \text{ and } \mathbf{w}_{u,k} \in N\left(0, \sigma_{u,k}\right) \end{aligned} \tag{3.11}$$

where $\mathbf{F}_k$ denotes the state propagation model and $\mathbf{u}_k$ is the control input vector. The mean and covariance of the prior probability are

$$\begin{aligned} \mathrm{E}\left[P\left(\mathbf{x}_k | \mathbf{z}_{k-1}, \mathbf{u}_k\right)\right] &= \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{u}_k = \mathbf{x}_{k,k-1} \\ \mathrm{Cov}\left[P\left(\mathbf{x}_k | \mathbf{z}_{k-1}, \mathbf{u}_k\right)\right] &= \mathbf{F}_k \mathbf{\Sigma}_{\mathbf{x}_{k-1}} \mathbf{F}_k^T + \Sigma_{\mathbf{w}_{u,k}} = \mathbf{P}_{k,k-1} \end{aligned} \tag{3.12}$$

Thus, the prior probability can also be described as a Gaussian distribution,

$$P\left(\mathbf{x}_k|\mathbf{z}_{k-1},\mathbf{u}_k\right) \propto -\exp\left(\left(\mathbf{x}_k - \mathbf{x}_{k,k-1}\right)^T \mathbf{P}_{k,k-1}^{-1}\left(\mathbf{x}_k - \mathbf{x}_{k,k-1}\right)\right) \tag{3.13}$$

Substituting (3.10) and (3.13) into (3.7) yields the posterior probability:

$$P\left(\mathbf{x}_k|\mathbf{z}_k,\mathbf{u}_k\right) = -C\exp\begin{pmatrix}\left(\mathbf{z}_k - \mathbf{H}_k\mathbf{x}_k\right)^T \Sigma_{\mathbf{w}_{z,k}}^{-1}\left(\mathbf{z}_k - \mathbf{H}_k\mathbf{x}_k\right) \\ \cdot\left(\mathbf{x}_k - \mathbf{x}_{k,k-1}\right)^T \mathbf{P}_{k,k-1}^{-1}\left(\mathbf{x}_k - \mathbf{x}_{k,k-1}\right)\end{pmatrix} \tag{3.14}$$

where C is a normalizing constant.

As outlined in the assumptions, an accurate representation of $P(\mathbf{x}_0)$ is available, meaning that the uncertainty of the initial state $\mathbf{x}_0$ can be presented by $\boldsymbol{\Sigma}_0$ If the MAP of $\mathbf{x}_1$ (denoted as $\hat{\mathbf{x}}_{1,\mathrm{MAP}}$) and its associated uncertainty $\boldsymbol{\Sigma}_1$ can be estimated using $\mathbf{x}_0$ and $\boldsymbol{\Sigma}_0$, then $\hat{\mathbf{x}}_{k,\mathrm{MAP}}$ can be estimated recursively from the time frame 0, 1, …, $k-1$. In essence, with the accurate representation of $P(\mathbf{x}_0)$, the first-order Markov chain property is utilized in the Kalman filter. Therefore, a typical two-slice Kalman filter suffices to continuously provide MAP estimation for the current state in each time frame, as shown in Figure 3.2. The key lies in determining $\hat{\mathbf{x}}_{k,\mathrm{MAP}}$ using $\mathbf{x}_{k-1}$ and $\boldsymbol{\Sigma}_{\mathbf{x}_{k-1}}$. It is intuitive to apply the first-order derivative rule to find a state that maximizes its posterior probability. The MAP estimation can be derived using the following equation:

$$\left.\frac{\partial P\left(\mathbf{x}_k|\mathbf{z}_k,\mathbf{u}_k\right)}{\partial \mathbf{x}_k}\right|_{\mathbf{x}_k=\hat{\mathbf{x}}_{k,\mathrm{MAP}}} = 0 \tag{3.15}$$

As a result, the MAP of the state at the current time frame is given by:

$$\hat{\mathbf{x}}_{k,\mathrm{MAP}} = \left(\mathbf{H}_k^T \Sigma_{\mathbf{w}_{z,k}}^{-1}\mathbf{H}_k + \mathbf{P}_{k,k-1}^{-1}\right)^{-1} \times \left(\mathbf{P}_{k,k-1}^{-1}\mathbf{x}_{k,k-1} + \mathbf{H}_k^T \Sigma_{\mathbf{w}_{z,k}}^{-1}\mathbf{z}_k\right) \tag{3.16}$$

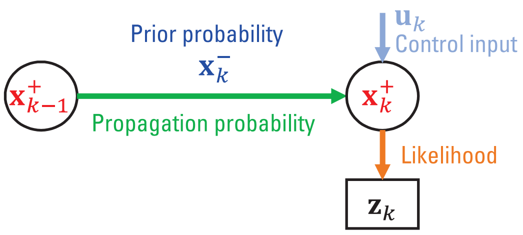


Figure 3.2 A typical example of a two slices Kalman filtering problem.

Equation (3.16) can be reorganized as:

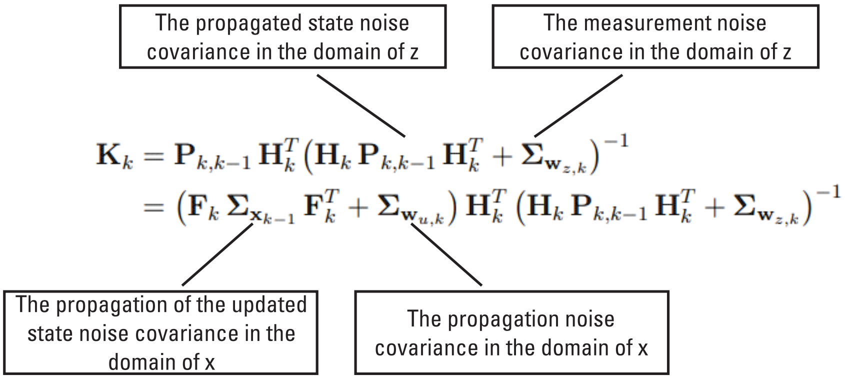
$$\hat{\mathbf{x}}_{k,\mathrm{MAP}} = \mathbf{x}_{k,k-1} + \mathbf{K}_k\left(\mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k,k-1}\right) \tag{3.17}$$

$$\mathbf{K}_k = \mathbf{P}_{k,k-1}\mathbf{H}_k^T\left(\mathbf{H}_k\mathbf{P}_{k,k-1}\mathbf{H}_k^T + \boldsymbol{\Sigma}_{\mathbf{w}_{z,k}}\right)^{-1} \tag{3.18}$$

where **K** represents the Kalman filter gain, which balances the weightings between the prior probability and likelihood. As illustrated in Figure 3.3, the term inside the inverse operation in the Kalman filter gain calculation reflects the uncertainty associated with the likelihood. It takes into account both the measurement uncertainty in the current time frame and the propagated state uncertainty, along with the observability of the measurement model. Conversely, the other terms represent the uncertainty of the prior probability, taking into account both the uncertainty in the propagation input at the current time frame and the observability of the propagation model. Let's explore some extreme cases to understand this better:

1. $\boldsymbol{\Sigma}_{\mathbf{w}_{u,k}} \ll \boldsymbol{\Sigma}_{\mathbf{w}_{z,k}}$: The uncertainty of the system propagation is much smaller than that of the measurement update. Thus, $\mathbf{K}_k \rightarrow 0$, $\hat{\mathbf{x}}_{k,\mathrm{MAP}} \approx \mathbf{x}_{k,k-1}$.
2. $\boldsymbol{\Sigma}_{\mathbf{w}_{u,k}} \gg \boldsymbol{\Sigma}_{\mathbf{w}_{z,k}}$: The uncertainty of the measurement update is much smaller than that of the system propagation. Thus, $\mathbf{K}_k \rightarrow \infty$, $\hat{\mathbf{x}}_{k,\mathrm{MAP}} \approx (\mathbf{H}_k^T\mathbf{H}_k)^{-1}\mathbf{H}_k^T\mathbf{z}_k$.

Intuitively speaking, if the system uncertainty is much smaller, it will rely more on the prior estimate. Otherwise, the posterior estimate will place greater


The propagated state noise covariance in the domain of z
The measurement noise covariance in the domain of z
The propagation of the updated state noise covariance in the domain of x
The propagation noise covariance in the domain of x

$$\begin{aligned}\mathbf{K}_k &= \mathbf{P}_{k,k-1}\,\mathbf{H}_k^T\left(\mathbf{H}_k\,\mathbf{P}_{k,k-1}\,\mathbf{H}_k^T + \boldsymbol{\Sigma}_{\mathbf{w}_{z,k}}\right)^{-1}\\ &= \left(\mathbf{F}_k\,\boldsymbol{\Sigma}_{\mathbf{x}_{k-1}}\,\mathbf{F}_k^T + \boldsymbol{\Sigma}_{\mathbf{w}_{u,k}}\right)\mathbf{H}_k^T\left(\mathbf{H}_k\,\mathbf{P}_{k,k-1}\,\mathbf{H}_k^T + \boldsymbol{\Sigma}_{\mathbf{w}_{z,k}}\right)^{-1}\end{aligned}$$

Figure 3.3 The computation of Kalman filter gain.

trust in the measurement-updated result. If the uncertainties fall between these two extremes, the Kalman gain is capable of optimally adjusting its weightings. This adjustment considers the observability of the model, ensuring that the gain is appropriately balanced for the given conditions. This ability to fine-tune its response based on varying levels of uncertainty is the core beauty of the Kalman filter, demonstrating its versatility and effectiveness in diverse scenarios.

Due to the recursive nature of the Kalman filter, the noise covariance of the current state, denoted as $\boldsymbol{\Sigma}_{\mathbf{x}_k}$, is indispensable for estimating the state in the subsequent time frame. The derivation of this covariance can be represented as follows:

$$\boldsymbol{\Sigma}_{\mathbf{x}_k} = \mathrm{Cov}\left[\mathbf{x}_k - \hat{\mathbf{x}}_{k,\mathrm{MAP}}\right] = \mathbf{P}_{k,k-1} - \mathbf{F}_k\mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k,k-1} \tag{3.19}$$

To summarize, based on the above derivation, the renowned five equations of the Kalman filter are sufficient to achieve MAP, provided that the three key assumptions are met. In practical scenarios, a certain degree of nonlinearity, non-Gaussian properties, and inaccuracies in the initial prior probability can be tolerated, allowing the Kalman filter to be effectively applied in many engineering contexts. However, implementing the Kalman filter for sensor fusion does necessitate careful calibration $\boldsymbol{\Sigma}_0$, $\boldsymbol{\Sigma}_{\mathbf{w}_{u},k}$, and $\boldsymbol{\Sigma}_{\mathbf{w}_{z},k}$ (also well-known as $\mathbf{P}_0$, $\mathbf{Q}$, and $\mathbf{R}$) to approximate the MAP. These parameters are crucial because they underpin the fundamental assumptions ensuring the optimality of the Kalman filter.

$$\begin{aligned}
\mathbf{x}_{k,k-1} &= \mathbf{F}_k\mathbf{x}_{k-1} + \mathbf{u}_k \\
\mathbf{P}_{k,k-1} &= \mathbf{F}_k\boldsymbol{\Sigma}_{\mathbf{x}_{k-1}}\mathbf{F}_k^T + \boldsymbol{\Sigma}_{\mathbf{w}_{u,k}} \\
\mathbf{K}_k &= \mathbf{P}_{k,k-1}\mathbf{H}_k^T\left(\mathbf{H}_k\mathbf{P}_{k,k-1}\mathbf{H}_k^T + \boldsymbol{\Sigma}_{\mathbf{w}_{z,k}}\right)^{-1} \\
\hat{\mathbf{x}}_{k,\mathrm{MAP}} &= \mathbf{x}_{k,k-1} + \mathbf{K}_k\left(\mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k,k-1}\right) \\
\boldsymbol{\Sigma}_{\mathbf{x}_k} &= \mathbf{P}_{k,k-1} - \mathbf{F}_k\mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k,k-1}
\end{aligned} \tag{3.20}$$

For linear-Gaussian problems, the Kalman filter is unmatched in both performance and elegance.

3.2.2 Extended Kalman Filter

The extended Kalman filter (EKF) is the most commonly adapted version of the Kalman filter for handling nonlinear systems. In an EKF, either the state transition model, the observation model, or both can be nonlinear functions:

$$\mathbf{x}_k = f\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \mathbf{w}_k, \mathbf{z}_k = h\left(\mathbf{x}_k\right) + \mathbf{v}_k \tag{3.21}$$

where $f(\cdot)$ and $h(\cdot)$ are nonlinear mappings (not necessarily matrices), such models are commonly found in indoor environments. For example, f might represent the motion equations of a robot, which are nonlinear due to rotations, while h could describe a distance measurement as a nonlinear function of position. The EKF applies the logic of the Kalman filter by linearizing these nonlinear functions around the current estimate. Essentially, it performs a local linear approximation, allowing the standard Kalman update equations to be used.

The EKF maintains a similar recursive structure: at each step, it predicts the state using f and then updates it using the new measurement and the linearized h. While it is not an optimal estimator in the general nonlinear case, it often performs well when the system's nonlinearity is mild or when the estimation error remains small, ensuring the validity of linearization. The EKF has been a reliable tool in indoor navigation. For instance, in pedestrian tracking, where inertial sensors provide a nonlinear motion model and radio beacons contribute nonlinear range measurements, the EKF has been widely adopted due to its relative simplicity and efficiency.

Start with mathematical derivation. At time $k-1$, assume that we have the state estimate $\hat{\mathbf{x}}_{k-1}$ and covariance $\mathbf{P}_{k-1}$. The EKF then proceeds as follows:

Prediction: Apply the nonlinear process model:

$$\begin{aligned} \hat{\mathbf{x}}_k^- &= f\left(\hat{\mathbf{x}}_{k-1}^-, \mathbf{u}_k\right) \\ \mathbf{P}_k^- &= \mathbf{F}_{k-1}\mathbf{P}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{Q} \end{aligned} \tag{3.22}$$

where $\mathbf{F}_{k-1} = \partial f / \partial \mathbf{x}|_{\hat{\mathbf{x}}_{k-1}}$ is the Jacobian of f with respect to the state, evaluated at the current estimate. This Jacobian, $\mathbf{F}_{k-1}$, plays the role of the state transition matrix in propagating uncertainty. Equation (3.22) is essentially a first-order approximation of how covariance evolves. The linearization of process model in EKF is shown in Figure 3.4.

Update: When $\mathbf{z}_k$ arrives, we linearize the observation model around the predicted state $\hat{\mathbf{x}}_k^-$. We compute $\mathbf{H}_k = \partial h / \partial \mathbf{x}|_{\hat{\mathbf{x}}_k^-}$, the Jacobian of h evaluated at the predicted state. Next, we form the innovation $\mathbf{y}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_k^-)$ and compute the innovation covariance as $\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}$. The Kalman gain and update formulas then follow the same structure as in the linear case:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_k^-\mathbf{H}_k^T\mathbf{S}_k^{-1} \\ \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_k^- + \mathbf{K}_k\mathbf{y}_k \\ \mathbf{P}_k &= \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_k^- \end{aligned} \tag{3.23}$$

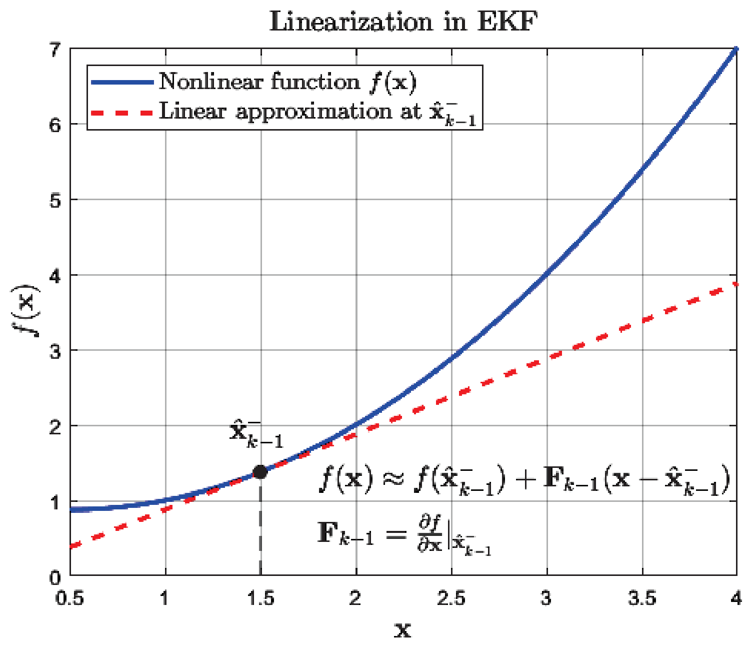


Figure 3.4 The linearization of process model in EKF.

These EKF equations demonstrate that, apart from the Jacobian calculations, the filter operations remain the same as those in the linear Kalman filter. The success of the EKF depends on the validity of the linearization as an approximation of the system's behavior. If $\hat{\mathbf{x}}_k^-$ is close to the true state, $\mathbf{H}_k$ will accurately capture how changes in the state affect the measurement, allowing the update step to correct the estimate appropriately. However, if the estimate is poor or the system is highly nonlinear, $\mathbf{H}_k$ may be an inadequate approximation, leading to incorrect updates that can potentially destabilize the filter.

The EKF inherits many strengths of the linear Kalman filter: it is relatively easy to implement, is computationally efficient (often $O(n^3)$ per step due to matrix operations, which is fine for moderate state sizes), and provides state estimates in real time with an associated covariance. It works well for a vast array of mild nonlinear problems and has been the default choice for many navigation systems.

3.2.3 Iterated Extended Kalman Filter

When measurement nonlinearities are severe, one way to improve the EKF's accuracy is to iterate the update step multiple times per measurement. The iterated extended Kalman filter (IEKF) applies the measurement update repeatedly in a single time step, relinearizing at the updated state each time. This effectively performs a Newton-like refinement to solve the measurement equation more accurately. Bell and Cathey (1993) showed that the IEKF update is equivalent to one or more Gauss-Newton iterations applied to the measurement residual [4].

{AU: Please cite ref [3] in numerical order in the text}

Followed by the EKF equations, after obtaining the predicted state $(\hat{\mathbf{x}}_k^-, \mathbf{P}_k^-)$, the IEKF performs:

1. Initialize $\mathbf{x}_k^{(0)} = \hat{\mathbf{x}}_k^-$.
2. For $j = 0, 1 \ldots, N$ (a small number of iterations, until convergence):
   - Compute $\mathbf{H}_k^{(j)} = \partial h/\partial \mathbf{x}|_{\mathbf{x}_k^{(j)}}$.
   - Compute the innovation $\mathbf{y}_k^{(j)} = \mathbf{z}_k - h(\mathbf{x}_k^{(j)}) + \mathbf{H}_k^{(j)}(\mathbf{x}_k^{(j)} - \hat{\mathbf{x}}_k^-)$. This is a modified residual that accounts for the shift in linearization point (so that $\mathbf{y}_k^{(j)}$ would be 0 if $\mathbf{x}_k^{(j)}$ were the correct state).
   - Compute the Kalman gain $\mathbf{K}_k^{(j)} = \mathbf{P}_k^-[\mathbf{H}_k^{(j)}]^T(\mathbf{H}_k^{(j)}\mathbf{P}_k^-[\mathbf{H}_k^{(j)}]^T + \mathbf{R})^{-1}$.
   - Update the state estimate: $\mathbf{x}_k^{(j+1)} = \hat{\mathbf{x}}_k^- + \mathbf{K}_k^{(j)}\mathbf{y}_k^{(j)}$.
3. Set $\hat{\mathbf{x}}_k = \mathbf{x}_k^{(N+1)}$ as the final estimate, and update covariance $\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k^{(N)}\mathbf{H}_k^{(N)})\mathbf{P}_k^-$.

In essence, the IEKF repeatedly linearizes the measurement model at the latest state estimate and applies a correction. The modified residual $\mathbf{y}_k^{(j)}$ ensures that, upon convergence, $\mathbf{x}_k^{(j)}$ closely satisfies the measurement equation. Mathematically, the IEKF is equivalent to performing a Gauss-Newton optimization of the measurement likelihood at each time step, starting from the predicted state. By reducing linearization error, it enhances filter stability, particularly in highly nonlinear scenarios.

3.2.4 Unscented Kalman Filter

The unscented Kalman filter (UKF), developed by Julier and Uhlmann [5], was introduced as an alternative to the EKF to overcome the limitations of linearization. The UKF belongs to the family of sigma-point filters. Its key idea is to statistically propagate a Gaussian distribution through nonlinear functions without relying on linearization. Instead of using derivatives, the UKF selects a set of carefully chosen sample points (sigma points) that capture the mean and covariance of the prior distribution. These points are then passed through the nonlinear function, and a new Gaussian approximation is computed from the transformed points. As a result, the UKF often achieves second-order accuracy for the mean and covariance in nonlinear transformations, whereas the EKF is only first-order accurate. Notably, the UKF does not require analytic Jacobians, simplifying implementation for complex models. This feature is particularly valuable in indoor positioning, where it can handle intricate motion models (potentially learned from data) or measurement models that are difficult to differentiate analytically. An example of mean and covariance propagation among actual (sampling), EKF, and UKF is shown in Figure 3.5.

We describe one time step of the UKF, which follows the standard predict-update structure:

- *Sigma point generation:* At time step $k - 1$, given the mean $\hat{\mathbf{x}}_{k-1}$ and covariance $\mathbf{P}_{k-1}$, we select $2n + 1$ sigma points in the state space, where n is the state dimension:

$$\chi_0 = \hat{\mathbf{x}}_{k-1},\ \chi_i = \hat{\mathbf{x}}_{k-1} + \delta_i,\ \chi_{i+n} = \hat{\mathbf{x}}_{k-1} - \delta_i,\ i = 1,\ldots,n, \qquad (3.24)$$

where δ_i is the ith column of $\mathbf{L} = \sqrt{(n+\lambda)\mathbf{P}_{k-1}}$, the matrix square root of $(n+\lambda)\mathbf{P}_{k-1}$ (commonly computed via the Cholesky decomposition). The parameter λ is a scaling factor (usually $\lambda = \alpha^2(n+\kappa) - n$ for some small α and κ). Intuitively, χ_i are spread around the mean so that their sample mean and covariance match $\hat{\mathbf{x}}_{k-1}$ and $\mathbf{P}_{k-1}$ up to the third order of smallness. Associated weights $W_i^{(m)}$ and $W_i^{(c)}$ are set

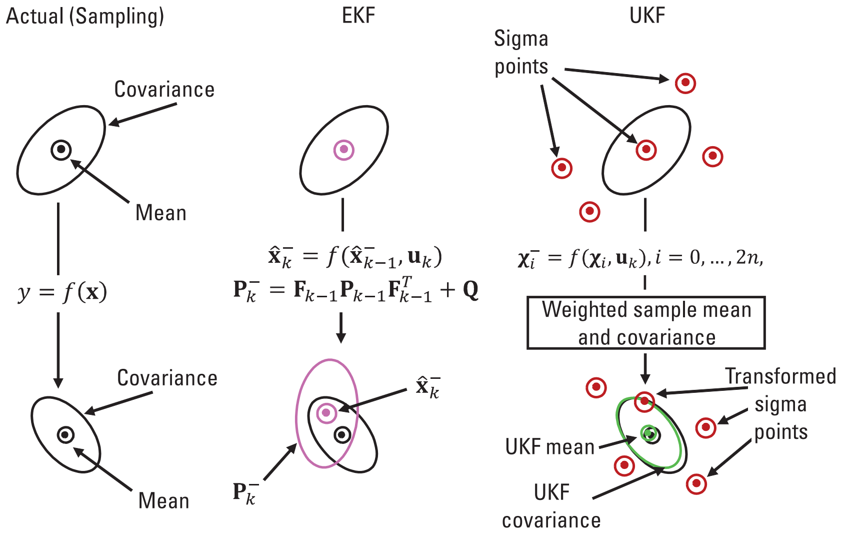

Actual (Sampling)
EKF
UKF
Sigma points
Covariance
Mean
$y = f(\mathbf{x})$
Covariance
Mean
$\hat{\mathbf{x}}_k^- = f(\hat{\mathbf{x}}_{k-1}^-, \mathbf{u}_k)$
$\mathbf{P}_k^- = \mathbf{F}_{k-1}\mathbf{P}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}$
$\hat{\mathbf{x}}_k^-$
$\mathbf{P}_k^-$
$\chi_i^- = f(\chi_i, \mathbf{u}_k), i = 0, \ldots, 2n,$
Weighted sample mean and covariance
Transformed sigma points
UKF mean
UKF covariance

Figure 3.5 Example of mean and covariance propagation: actual (sampling), EKF, and UKF [6].

for computing mean and covariance (with $W_0^{(m)} = \lambda/(n + \lambda)$, $W_0^{(c)} = \lambda/(n + \lambda) + (1 - \alpha^2 + \beta)$, and $W_i^{(m)} = W_i^{(c)} = 1/[2(n + \lambda)]$ for $i > 0$; β is an extra parameter often set to 2 for Gaussian priors).

- *Prediction:* Each sigma point is propagated through the potentially nonlinear motion model:

$$\chi_i^- = f\left(\chi_i, \mathbf{u}_k\right),\ i = 0,\ldots,2n \tag{3.25}$$

This yields a set of predicted sigma points, χ_i^-, for the state at time k. The prior state mean and covariance are then reconstructed as weighted averages:

$$\begin{gathered}\hat{\mathbf{x}}_k^- = \sum_{i=0}^{2n} W_i^{(m)} \chi_i^- \\ \mathbf{P}_k^- = \sum_{i=0}^{2n} W_i^{(c)} \left(\chi_i^- - \hat{\mathbf{x}}_k^-\right)\left(\chi_i^- - \hat{\mathbf{x}}_k^-\right)^T + \mathbf{Q}\end{gathered} \tag{3.26}$$

Equation (3.26) adds the process noise covariance $\mathbf{Q}$ after computing the spread of the transformed sigma points. At this stage, the predicted mean and covariance are obtained without ever computing the Jacobian of f.

- *Measurement update:* Next, we propagate the sigma points through the observation model:

$$\mathcal{Z}_i = h\left(\chi_i^-\right),\ i = 0,\ldots,2n \tag{3.27}$$

This generates a set of predicted measurements, from which we compute the predicted measurement mean and covariance:

$$\begin{gathered}\hat{\mathbf{z}}_k = \sum_{i=0}^{2n} W_i^{(m)} \mathcal{Z}_i, \\ \mathbf{P}_{zz} = \sum_{i=0}^{2n} W_i^{(c)} \left(\mathcal{Z}_i - \hat{\mathbf{z}}_k\right)\left(\mathcal{Z}_i - \hat{\mathbf{z}}_k\right)^T + \mathbf{R}\end{gathered} \tag{3.28}$$

We also compute the cross-covariance between the state and the measurement:

$$\mathbf{P}_{xz} = \sum_{i=0}^{2n} W_i^{(c)} \left(\chi_i^- - \hat{\mathbf{x}}_k^-\right)\left(\mathcal{Z}_i - \hat{\mathbf{z}}_k\right)^T \tag{3.29}$$

Finally, we compute the Kalman gain and perform the update, just as in the Kalman filter:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_{xz}\mathbf{P}_{zz}^{-1}, \\ \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_k^- + \mathbf{K}_k\left(\mathbf{z}_k - \hat{\mathbf{z}}_k\right), \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k\mathbf{P}_{zz}\mathbf{K}_k^T. \end{aligned} \tag{3.30}$$

This yields the posterior state estimate $\hat{\mathbf{x}}_k$ and covariance $\mathbf{P}_k$. The UKF requires carrying and transforming $2n + 1$ points, making its computational complexity per step approximately $O(n^3)$, due to the covariance updates and matrix inversions involving matrices of size n and the measurement dimension. For moderate dimensions, this complexity remains manageable. The key advantage of the UKF is its improved accuracy. With a proper choice of parameters (α, β, κ), it can capture the effects of nonlinearities more effectively than an EKF linearization. For instance, if the true posterior after a prediction or update is mildly non-Gaussian, the sigma points can approximate the mean and covariance more accurately than an EKF, which only propagates means and linearized variances.

3.2.5 Comparison Between the Kalman Filter and Its Variants

Each of these variants has its place. EKF is commonly the default choice in many engineering applications due to its familiarity and simplicity. IEKF iteratively refines EKF updates for improved linearization accuracy at the cost of additional computation. The UKF is preferred when model fidelity is critical or when avoiding Jacobians simplifies implementation. To select which Kalman filter to use, the following questions should be answered: when does each filter work best? Which filter is the most efficient and scalable in high-dimensional problems? What are the challenges in deriving and tuning parameters? How does each filter handle uncertainty and outliers? For the ease of the reader, Table 3.2 compares these aspects.

In the next section, we move beyond Kalman filters to particle filtering, which takes a fundamentally different, sampling-based approach to state estimation.

3.3 Particle Filters

When assumptions of Kalman filters, such as Gaussian noise, unimodal distributions, and near-linear dynamics, do not hold, particle filters provide a more

Table 3.2
Comparison Between the Kalman Filter and Its Variants

{AU: Cite ref [7] in order in text}

Aspect	Kalman Filter	EKF	IEKF	UKF
Accuracy and applicability [8–10]	Optimal for linear Gaussian systems: achieves minimum-variance, unbiased estimates when both system dynamics and observations are perfectly linear with Gaussian noise Limitations with nonlinear systems: performance deteriorates significantly in strongly nonlinear environments, which is why it is rarely used as a standalone solution for such cases	Handles mild nonlinearities: uses a first-order Taylor approximation to linearize around the current state; performs best when nonlinearities are moderate, and the operating point changes slowly between steps Limitations with high nonlinearity: struggles with highly nonlinear problems or poor initial estimates, as unmodeled higher-order effects may introduce bias or lead to divergence	Enhanced accuracy via iteration: improves accuracy in strongly nonlinear systems by iteratively refining the measurement update (Gauss-Newton method), providing a more optimal linearization point at each step Effective for significant nonlinearity: particularly beneficial when a single EKF linearization is insufficient. IEKF achieves closer-to-optimal estimates by converging locally to the nonlinear update solution, provided the initial prediction is reasonably accurate	Designed for highly nonlinear systems: utilizes sigma points to propagate Gaussian distributions through nonlinear transformations, accurately capturing mean and covariance without explicit linearization or analytic Jacobians, making it ideal for complex or black-box models Effective in complex scenarios: particularly suitable for models with significant nonlinearities or complexities, such as angle wrapping or range measurements, where EKF's first-order approximations would introduce substantial errors
Computational complexity [8, 9, 11]	Computational efficiency: each update involves matrix operations, making it efficient and suitable for real-time use.	Computational cost: per-step computational cost is comparable to a standard Kalman filter, involving similar matrix multiplications	Iterative computational cost: IEKF requires multiple iterations per update, increasing computation	Higher computational cost than EKF: Requires generating and propagating $(2n+1)$ sigma points (for

Table 3.2
(Cont.)

Aspect	**Kalman Filter**	**EKF**	**IEKF**	**UKF**
	Baseline computational load: provides the lowest computational effort among Kalman filter variants, as it requires neither iterative refinements nor sampling-based computations.	and inversions, plus the additional but typically minor; cost of computing Jacobians. Real-time feasibility: remains suitable for real-time applications, especially when Jacobian computations represent only a small fraction of the total update cost; sparsity optimizations further enable its use in large-scale problems such as SLAM.	proportionally to the number of iterations. Typically, 2–5 iterations are sufficient, resulting in roughly 2–5 times the cost of a single EKF update. Practical efficiency: Although iterations increase computational load, IEKF remains computationally efficient in practice. In most indoor applications, 1 to 2 iterations provide an effective balance between accuracy and computational complexity.	state dimension (n)), increasing computational load and memory usage linearly with state size. Although per-step complexity remains polynomial, the constant factors are higher compared to EKF. Practical performance and scalability: generally, fast enough for moderate-sized state dimensions but can become computationally challenging in high-dimensional scenarios.
Ease of implementation [11–13]	Simple implementation: easy to implement using well-known standard formulas, requiring only the specification of linear system matrices (**F**, **H**) and tuning of noise covariances (**Q**, **R**).	Moderate implementation difficulty: requires analytic Jacobians of motion and measurement models, which can be challenging to derive manually, especially in complex scenarios involving trigonometric relationships; these linearization steps increase initial development effort, debugging complexity, and maintenance if models change.	Simple extension of EKF: IEKF builds directly upon the EKF framework by introducing iterative updates, requiring minimal adjustments to existing EKF implementations. However, careful coding is needed to implement convergence checks or fixed-iteration loops clearly and correctly.	No Jacobian needed, direct model usage: directly utilizes the model's forward function without explicit linearization, simplifying initial setup and enabling easy application to complex or black-box systems. Implementation complexity: implementation involves unscented transform, including generating sigma points, propagating them

Table 3.2
(Cont.)

Aspect	Kalman Filter	EKF	IEKF	UKF
				through nonlinear models, and recombining results.
Robustness to noise and model errors [8, 9, 11, 14]	Sensitive to assumption violations: Kalman filter heavily relies on model accuracy; if the model is incorrect or noise is non-Gaussian, estimation errors can increase, potentially leading to divergence. Highly susceptible to outliers: Kalman filter treats unexpected sensor jumps as genuine observations, which can significantly distort state estimates unless additional measures such as outlier rejection or covariance inflation are applied.	Same assumption sensitivity as Kalman filter: the EKF relies on accurate models and Gaussian noise; if the model is significantly incorrect or the noise is highly non-Gaussian, EKF has no built-in resilience and may fail. Underestimates true uncertainty: EKF often provides overly optimistic covariance estimates, making it less adaptable to unexpected changes and exacerbating the impact of model errors. Requires a good initial estimate: if initialized too far from the true state, the linearization may be invalid, preventing the filter from recovering.	Reduces linearization error: by refining the estimate iteratively during the update, the IEKF reduces linearization errors, leading to more consistent estimates in highly nonlinear scenarios. More robust to poor initial linearization: if the initial linearization at a given time step is inaccurate, IEKF iterates to correct it. Still relies on a correct model: like EKF, IEKF assumes the overall model is correct; it does not inherently handle measurement outliers or bad data, which can still mislead the filter unless detected.	More robust to modeling errors and nonlinear effects: UKF does not rely on linearization, reducing the risk of divergence due to a poorly chosen linearization point. It also tracks state uncertainty more accurately, making it more adaptable when reality deviates from the assumed model. Less dependent on an accurate prior estimate: by propagating sigma points through the actual nonlinear system, UKF relies more on incoming measurements and is less sensitive to errors accumulated in the model; this allows it to correct deviations more effectively than EKF.

Table 3.2
(Cont.)

Aspect	Kalman Filter	EKF	IEKF	UKF
				Still assumes Gaussian noise: like Kalman filter and EKF, UKF does not inherently reject outliers or handle non-Gaussian noise; without additional robust techniques or careful noise covariance tuning, it will incorporate outliers just as the other filters would.
Best use cases for IPIN applications [10, 14–18]	Direct position measurements Barometric altimeter smoothing Single-axis drift correction Baseline for other filters	Default choice for moderate nonlinearities IPIN applications Indoor sensor fusion combining IMU data with radio measurements (e.g., Wi-Fi/BLE RSSI or UWB range) Smartphone navigation	LiDAR-inertial odometry for robots Visual-inertial odometry	RF-based (Wi-Fi, BLE, or UWB positioning) localization Robust navigation for autonomous robots

general framework for state estimation. Particle filters belong to the broader class of sequential Monte Carlo methods and represent the state's probability distribution using a set of random samples (particles) rather than a closed-form expression (mean and covariance). This approach allows particle filters to handle multimodal distributions, non-Gaussian noise, and highly nonlinear processes, albeit at the cost of increased computational complexity. Actually, UKF can be regarded as the integration of EKF and particle filter. Particle filter implements Bayes filtering (recursive Bayesian estimation) through sampling. The goal is to estimate $p(\mathbf{x}_k|\mathbf{z}_{1:k})$, the posterior distribution of the state $\mathbf{x}_k$ given all measurements up to time k. The Bayes filter update consists of two steps:

au: missing eq 3.31?

$$\begin{gathered} p\left(\mathbf{x}_k|\mathbf{z}_{1:k-1}\right) = \int p\left(\mathbf{x}_k|\mathbf{x}_{k-1}\right) p\left(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}\right) d\mathbf{x}_{k-1} \\ p\left(\mathbf{x}_k|\mathbf{z}_{1:k}\right) \propto p\left(\mathbf{z}_k|\mathbf{x}_k\right) p\left(\mathbf{x}_k|\mathbf{z}_{1:k-1}\right) \end{gathered} \tag{3.32}$$

In closed form, these integrals and multiplications are only tractable for simple cases—one such example is the Kalman filter, where everything remains Gaussian. Particle filters approximate these steps using a set of weighted samples $\{\mathbf{x}_k^{(i)}, w_k^{(i)}\}_{i=1}^N$ to represent $p(\mathbf{x}_k|\mathbf{z}_{1:k})$, where each $\mathbf{x}_k^{(i)}$ is a possible state (particle) and $w_k^{(i)}$ is its corresponding weight (importance probability).

The most commonly used algorithm is the sequential importance resampling (SIR) filter, also known as the Bootstrap filter [19]. The process of particle filter iteration is shown in Figure 3.6.

It follows these steps:

1. *Initialization:* Draw N samples $\mathbf{x}_0^{(i)}$ from the initial distribution $p(\mathbf{x}_0)$ and assign equal weights $w_0^{(i)} = 1/N$.

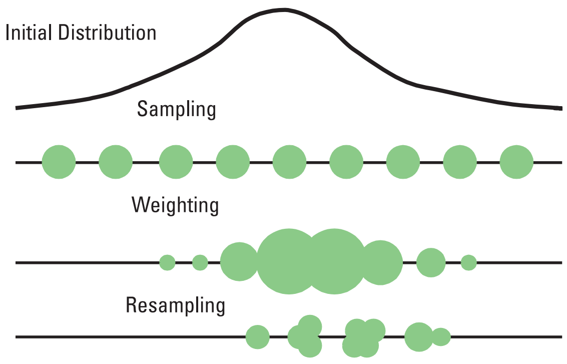


Figure 3.6 The process of particle filter iteration.

2. For each time step $k = 1, 2, \ldots$:
   (a) *Sampling (prediction):* For each particle i, generate a new sample from the process model:

$$\mathbf{x}_k^{(i)} \sim p\left(\mathbf{x}_k | \mathbf{x}_{k-1}^{(i)}\right) \tag{3.33}$$

   This is analogous to propagating each sample through the motion model with added noise.

   (b) *Importance weighting (update):* For each particle, compute the likelihood of the new measurement given the particle's state:

$$\tilde{w}_k^{(i)} = w_{k-1}^{(i)} p\left(\mathbf{z}_k | \mathbf{x}_k^{(i)}\right) \tag{3.34}$$

   Since, after resampling in the previous step, we typically have $w_{k-1}^{(i)} = 1/N$, this simplifies to $\tilde{w}_k^{(i)} \propto p(\mathbf{z}_k | \mathbf{x}_k^{(i)})$. Intuitively, if a particle's predicted measurement closely matches the actual observation $\mathbf{z}_k$, its weight increases; otherwise, its weight decreases.

   (c) *Normalization:* Normalize the weights $\{\tilde{w}_k^{(i)}\}$ so that $\Sigma_i w_k^{(i)} = 1$.

   (d) *Resampling:* Draw N particles from the current set $\{\mathbf{x}_k^{(i)}\}$ with probabilities proportional to $w_k^{(i)}$. This results in a new set $\{\mathbf{x}_k^{(i)}\}$ (reindexed), where particles with higher weights are more likely to be replicated multiple times, while low-weight particles may be eliminated. After resampling, set $w_k^{(i)} = 1/N$ for all i. Resampling helps prevent degeneracy, a phenomenon in which most particles acquire negligible weights after multiple updates.

3. The state estimate can be computed from the particles, for example, as the weighted mean $\hat{\mathbf{x}}_k = \Sigma_i w_k^{(i)} \mathbf{x}_k^{(i)}$. Alternatively, the particle with the highest weight may be selected as the representative estimate. The full posterior distribution is captured by the ensemble of particles.

This algorithm can represent virtually any distribution given a sufficient number of particles. For example, if there are two likely positions for a user, perhaps two symmetric locations relative to certain beacons, the particle set can maintain two distinct clusters. In contrast, a Kalman filter would instead approximate the distribution with a single Gaussian, resulting in a compromise that places the mean between the two modes, which is incorrect if the distribution is truly bimodal. {AU: Word missing here?}

Particle filters can be computationally intensive, especially when the state dimension is high or when high accuracy (low variance) is required, necessitating a large number of particles. The number of particles N needed increases with the complexity of the distribution. For a 2-D position, a few hundred to a few thousand particles may be sufficient for room-level accuracy. However, for a high-dimensional state, such as a full Simultaneous Localization and Mapping (SLAM) problem that includes both the robot's pose and multiple landmarks, basic particle filters become impractical. This challenge led to specialized methods like the Rao-Blackwellized particle filter (RBPF) for SLAM, where some variables are tracked using analytical filters while only part of the state is sampled.

Resampling can also introduce issues, such as particle impoverishment, where diversity is lost if all but a few particles have negligible weight. To mitigate this, techniques like roughening (adding random noise after resampling) or systematic resampling (which ensures better diversity) are commonly used.

In indoor scenarios, the primary state variables of interest, position, and possibly heading are typically low-dimensional, which makes particle filters feasible. However, if additional variables such as velocity or sensor biases are included in the state, the dimensionality increases, requiring more particles. A practical strategy to address this is hybrid estimation, where some state components are estimated using a Kalman filter within each particle, the core idea behind Rao-Blackwellized particle filters. For instance, in SLAM, FastSLAM 0 [20] utilizes a particle filter to estimate the robot's pose while embedding a Kalman filter in each particle to estimate the map. Similarly, in indoor positioning, a particle filter could sample a user's position, while a small Kalman filter within each particle estimates step length or phone orientation. This hybrid approach improves efficiency by reducing the number of required particles while maintaining accurate state estimation.

Particle filters offer several advantages that make it a powerful and flexible estimation technique. Unlike Kalman filter-based methods, particle filters can represent arbitrary probability distributions, including multimodal and skewed distributions, allowing it to handle complex and highly nonlinear systems. Its adaptability extends to any system model without requiring linearization or Gaussian assumptions, making it suitable for a wide range of applications. Additionally, particle filters can seamlessly integrate nontraditional information sources, such as map data or logical constraints, enhancing estimation accuracy in real-world scenarios. Its conceptual simplicity, combined with its ability to solve difficult localization problems—such as the kidnapped robot problem and global localization—further demonstrates its robustness in cases where traditional filtering techniques struggle.

Despite its strengths, the particle filter has several limitations that can pose challenges in practical applications. One major drawback is its computational intensity, as achieving stable results often requires a large number of particles, making real-time implementation difficult. However, a particle filter is well-suited for parallel processing since each particle independently undergoes prediction and weighting. Additionally, a particle filter suffers from approximation errors due to the finite number of particles; if the sample size is too small, the estimated distribution may contain significant sampling noise, leading to inconsistent or inaccurate state estimates. Another key issue is degeneracy, where insufficient particle spread or low process noise causes most particles to concentrate on an incorrect hypothesis, leading to divergence. While resampling helps mitigate degeneracy, it can also inadvertently discard useful particles due to randomness. Finally, a particle filter requires careful tuning of parameters such as the number of particles, process noise models, and measurement likelihoods. An inadequate number of particles may fail to capture the true uncertainty, while excessive particles increase computational costs, making parameter selection a critical but challenging aspect of particle filter implementation.

Particle filters have been successfully implemented on mobile and embedded devices for indoor localization. They are often combined with other methods, for example, using a particle filter for coarse global localization and then switching to a Kalman filter for fine tracking once a reliable hypothesis has been established.

3.4 Factor Graph Optimization

The methods discussed so far, Kalman filters and particle filters, perform filtering, estimating the current state sequentially in an online manner. In contrast, factor graph optimization (FGO) typically addresses smoothing or batch estimation, considering a collection of states over time and solving for the entire trajectory that best fits all measurements. This approach is closely related to least-squares estimation but is generally formulated within a probabilistic graphical model framework. The section begins with the derivation of FGO. Typical algorithm models are presented to illustrate the flexibility of the factor graph in solving problems related to multisensor integration.

Similar to the previous section, we are interested in the unknown state variables $\mathbf{X}$, such as position and orientation, given the measurements $\mathbf{Z}$. The MAP estimation is so named because it tries to maximize the posterior density $p(\mathbf{X}|\mathbf{Z})$ of the states $\mathbf{X}$ given the measurements $\mathbf{Z}$:

$$\begin{aligned}\hat{\mathbf{X}}_{\text{MAP}} &= \arg\max_{\mathbf{X}} p(\mathbf{X}|\mathbf{Z}) \\ &= \arg\max_{\mathbf{X}} \frac{p(\mathbf{Z}|\mathbf{X})\,p(\mathbf{X})}{p(\mathbf{Z})}\end{aligned} \tag{3.35}$$

The second equation above is obtained based on Bayes' law [21] and expresses the posterior as the product of the measurement density $p(\mathbf{Z}|\mathbf{X})$ and the prior $p(\mathbf{X})$ over the states, normalized by the factor $p(\mathbf{Z})$.

A distinct expression of Bayes' law is crucial to describe the actual computation underlying MAP estimation. Indeed, all the quantities in Bayes' law can theoretically be calculated from the Bayesian net [21]. In particular, the measurements $\mathbf{Z}$ are provided, and the normalization factor $p(\mathbf{Z})$ becomes irrelevant to the maximization and can be disregarded or assumed as $p(\mathbf{Z}) = 1$. Furthermore, while the conditional density $p(\mathbf{Z}|\mathbf{X})$ is a properly normalized Gaussian density in (3.35), our concern lies only with it as a function in the unknown states $\mathbf{X}$. Hence, the simplified form of Bayes' law is as follows:

$$\hat{\mathbf{X}}_{\mathrm{MAP}} = \arg\max_{\mathbf{X}} l(\mathbf{X};\mathbf{Z})\, p(\mathbf{X}) \tag{3.36}$$

Here $l(\mathbf{X};\mathbf{Z})$ is the likelihood of the states $\mathbf{X}$ given the measurements $\mathbf{Z}$, and is defined as a function that is proportional to $p(\mathbf{Z}|\mathbf{X})$:

$$l(\mathbf{X};\mathbf{Z}) \propto p(\mathbf{Z}|\mathbf{X}) \tag{3.37}$$

Factor graphs are graphical models [22–24] that are well suited to modeling complex estimation problems, such as SLAM or a structure from motion (SFM) [25]. In particular, there are variables and factors in factor graphs. The variables represent unknown quantities in the problem, and the factors represent functions on subsets of the variables. Edges in the factor graph are always between factors and variables and indicate that a particular factor depends on a particular variable.

A Wi-Fi inertial system is a widely used navigation system that combines the capabilities of both a Wi-Fi and an IMU. Let's begin with a classic example of Wi-Fi inertial navigation using FGO as illustrated in Figure 3.7. As can be seen, a factor graph is a bipartite graph with variable nodes (states at different times, or other parameters) and factor nodes (representing constraints or measurements relating variables). The factor graph captures the structure of the estimation problem.

In this derivation, $\mathbf{x}_k$, $\mathbf{u}_k$, and $\mathbf{z}_k$ represent the state vector, the control (or propagation) vector (e.g., from IMU), and the measurement vector in the *k*th time frame, respectively. In the application of Wi-Fi inertial navigation, $\mathbf{u}_k$ represents the acceleration and angular velocity from the IMU. We define that $\mathbf{Z} = \{\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_k\}$, $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_k\}$ and $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_{k-1}\}$. In addition, state propagation is $\mathbf{x}_k = \mathrm{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1})$, which can be derived based on IMU-based preintegration [26].

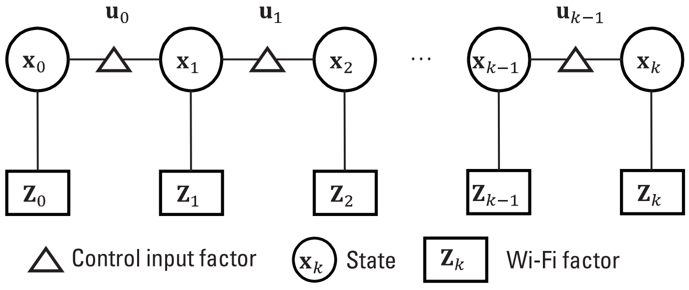


Figure 3.7 A typical example of the Wi-Fi inertial navigation problem. The triangle icon denotes the control input factor (e.g., from IMU). The rectangle icon denotes the Wi-Fi factor.

This proof does not delve into the details of the state propagation formula. More details and examples of IMU preintegration will be provided next. The measurement model is represented as $\mathbf{z}_k = \mathrm{h}(\mathbf{x}_k)$, which depends on the physical model of the sensor, such as Wi-Fi. Unlike the model for the EKF, the factor graph model is represented as an undirected graphical model, allowing the relationship between the states and the factors to be propagated both forward and backward.

Similar to EKF, which discusses the formulation of the EKF, the Gaussian noise assumption is adopted for both the state and all measurement models. Therefore, the state propagation conditional probability density, the measurement conditional probability density, and the prior probability density can be formulated as follows:

$$\begin{aligned} p\left(\mathbf{x}_k|\mathbf{u}_{k-1},\mathbf{x}_{k-1}\right) &= \frac{1}{\sqrt{|2\pi\mathbf{R}|}}\exp\left\{-\frac{1}{2}\left\|\mathrm{f}\left(\mathbf{x}_{k-1},\mathbf{u}_{k-1}\right)-\mathbf{x}_k\right\|_{\mathbf{R}}^2\right\} \\ p\left(\mathbf{z}_k|\mathbf{x}_k\right) &= \frac{1}{\sqrt{|2\pi\mathbf{P}|}}\exp\left\{-\frac{1}{2}\left\|\mathrm{h}\left(\mathbf{x}_k\right)-\mathbf{z}_k\right\|_{\mathbf{P}}^2\right\} \\ p\left(\mathbf{x}_0\right) &= \frac{1}{\sqrt{|2\pi\mathbf{Q}|}}\exp\left\{-\frac{1}{2}\left\|\mathbf{x}_0-\mu\right\|_{\mathbf{Q}}^2\right\} \end{aligned} \tag{3.38}$$

where $\mathbf{R}$, $\mathbf{P}$, and $\mathbf{Q}$ represent the covariance matrix of state propagation noise, measurement noise, and initial state noise, respectively. In particular, all these noises are subjected to Gaussian noise. In addition, the covariance matrix is applied as $\|\mathbf{F}\|_{\Sigma}^2 = \mathbf{F}^{\mathrm{T}}\Sigma^{-1}\mathbf{F}$. The μ denotes the prior information of the state. However, all these noises are determined based on the properties of the sensors applied. For example, the $\mathbf{R}$ for the IMU should be calibrated in advance.

According to (3.38), $p(\mathbf{Z}|\mathbf{X})p(\mathbf{X})$ can be transformed as follows:

$$\begin{aligned}
p(\mathbf{Z}|\mathbf{X})p(\mathbf{X}) &= \left\{\prod_k^0 p\left(\mathbf{z}_k|\mathbf{x}_k\right)\right\}\left\{p\left(\mathbf{x}_0\right)\prod_k^1 p\left(\mathbf{x}_k|\mathbf{u}_{k-1},\mathbf{x}_{k-1}\right)\right\} \\
&= \left\{\prod_k^0 \frac{1}{\sqrt{|2\pi\mathbf{P}|}}\exp\left\{-\frac{1}{2}\left\|\mathrm{h}\left(\mathbf{x}_k\right)-\mathbf{z}_k\right\|_{\mathbf{P}}^2\right\}\right\} \\
&\quad * \left\{\begin{array}{c} \frac{1}{\sqrt{|2\pi\mathbf{Q}|}}\exp\left\{-\frac{1}{2}\left\|\mathbf{x}_0-\mu\right\|_{\mathbf{Q}}^2\right\} \\ \prod_k^1 \frac{1}{\sqrt{|2\pi\mathbf{R}|}}\exp\left\{-\frac{1}{2}\left\|\mathrm{f}\left(\mathbf{x}_{k-1},\mathbf{u}_{k-1}\right)-\mathbf{x}_k\right\|_{\mathbf{R}}^2\right\} \end{array}\right\} \\
&= \mathbf{K} * \exp\left\{\begin{array}{c} -\frac{1}{2}\left\|\mathbf{x}_0-\mu\right\|_{\mathbf{Q}}^2-\sum_k^1\frac{1}{2}\left\|\mathrm{f}\left(\mathbf{x}_{k-1},\mathbf{u}_{k-1}\right)-\mathbf{x}_k\right\|_{\mathbf{R}}^2 \\ -\sum_k^0\frac{1}{2}\left\|\mathrm{h}\left(\mathbf{x}_k\right)-\mathbf{z}_k\right\|_{\mathbf{P}}^2 \end{array}\right\}
\end{aligned} \tag{3.39}$$

where $\mathbf{K}=\left(1/\sqrt{|2\pi\mathbf{Q}|}\right)\left(1/\sqrt{|2\pi\mathbf{R}|}\right)^k\left(1/\sqrt{|2\pi\mathbf{P}|}\right)^{k+1}$.

Calculating the logarithm of both sides of (3.39) yields:

$$\begin{aligned}
\log\left(p(\mathbf{Z}|\mathbf{X})p(\mathbf{X})\right) &= \log(\mathbf{K})-\frac{1}{2}\left\|\mathbf{x}_0-\boldsymbol{\mu}\right\|_{\mathbf{Q}}^2-\sum_k^1\frac{1}{2}\left\|\mathrm{f}\left(\mathbf{x}_{k-1},\mathbf{u}_{k-1}\right)-\mathbf{x}_k\right\|_{\mathbf{R}}^2 \\
&\quad -\sum_k^0\frac{1}{2}\left\|\mathrm{h}\left(\mathbf{x}_k\right)-\mathbf{z}_k\right\|_{\mathbf{P}}^2
\end{aligned} \tag{3.40}$$

Then (3.36) can be derived as follows:

$$\begin{aligned}
\hat{\mathbf{X}}_{\mathrm{MAP}} &= \arg\max_{\mathbf{X}} l(\mathbf{X};\mathbf{Z})p(\mathbf{X}) = \arg\max_{\mathbf{X}} p(\mathbf{Z}|\mathbf{X})p(\mathbf{X}) \\
&= \arg\max_{\mathbf{X}} \log\left(p(\mathbf{Z}|\mathbf{X})p(\mathbf{X})\right) \\
&= \arg\max_{\mathbf{X}} \left\{\begin{array}{c} -\frac{1}{2}\left\|\mathbf{x}_0-\boldsymbol{\mu}\right\|_{\mathbf{Q}}^2-\sum_k^1\frac{1}{2}\left\|\mathrm{f}\left(\mathbf{x}_{k-1},\mathbf{u}_{k-1}\right)-\mathbf{x}_k\right\|_{\mathbf{R}}^2 \\ -\sum_k^0\frac{1}{2}\left\|\mathrm{h}\left(\mathbf{x}_k\right)-\mathbf{z}_k\right\|_{\mathbf{P}}^2 \end{array}\right\}
\end{aligned}$$

$$= \underset{\mathbf{x}}{\arg\min} \left\{ \begin{array}{c} \frac{1}{2}\left\|\mathbf{x}_0 - \boldsymbol{\mu}\right\|_{\mathbf{Q}}^2 + \sum_k^1 \frac{1}{2}\left\|\mathrm{f}\left(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}\right) - \mathbf{x}_k\right\|_{\mathbf{R}}^2 \\ + \sum_k^0 \frac{1}{2}\left\|\mathrm{h}\left(\mathbf{x}_k\right) - \mathbf{z}_k\right\|_{\mathbf{P}}^2 \end{array} \right\} \tag{3.41}$$

In this case, the conditional probability problem is transformed into a quadratic problem. The minimum quadratic problem (3.41) can be solved using the Gaussian-Newton method, as referenced in [27]. In summary, FGO is derived based on the MAP estimation mentioned above. In practice, several popular open-source solvers are available to the community, including GTSAM [22] and the Ceres Solver [28]. For instance, GTSAM is open-source software that facilitates defining factors and solving FGO problems. It is one of the most widely used open-source tools for solving FGO in robotics and navigation. GTSAM comes with many commonly used factors already implemented and provided, such as the visual factor, IMU preintegration factor, and GNSS factor. Unlike GTSAM, the Ceres Solver is another open-source solver for nonlinear optimization, extensively used in robotics for state estimation, offering more foundational tools for users. Specifically, it allows users to customize the formulation of factors from scratch, which makes it a handy tool for users.

3.4.1 Numerical Optimization Methods

In optimization-based methods, three popular techniques are commonly employed to estimate the states by minimizing error functions: Gradient Descent, Gauss-Newton, and Levenberg-Marquardt, which are introduced.

3.4.1.1 Gradient Descent Method

A popular strategy to find the solution to minimize the cost function is the line search strategy, which iteratively updates the state by a certain step α along a chosen direction **d**:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\mathbf{d} \tag{3.42}$$

where subscripts k and $k + 1$ denotes the current and next iteration indexes, respectively. To ensure that the state update approaches the solution with the minimal cost function value, the cost function value is required to be decreased after each iteration:

$$f\left(\mathbf{x}_{k+1}\right) = f\left(\mathbf{x}_k + \alpha\mathbf{d}\right) < f\left(\mathbf{x}_k\right) \tag{3.43}$$

Thus, the line search strategy follows the procedures [29]: (1) find a descent direction $\mathbf{d}$ for the cost function in the state space; and (2) find a step length α achieving sufficient decrement on the cost function value. By repeating such procedures, the state converges to the solution when no sufficient decrement can be found. In practice, the step length α can be determined by generating a set of candidate values and selecting the one that achieves the minimum cost. A standard pseudocode is shown in Algorithm 3.1.

A straightforward way to guarantee a decrease in the cost function is to update the state along the direction of the steepest descent, which is known as the gradient descent method. The first-order Taylor expansion of cost function after an update can be expressed by

$$f\left(\mathbf{x}_k+\alpha\mathbf{d}\right)\approx f\left(\mathbf{x}_k\right)+\alpha\mathbf{d}^{\mathrm{T}}\mathbf{f}'\left(\mathbf{x}_k\right) \tag{3.44}$$

Assuming that the direction term $\mathbf{d}$ is normalized as a unit vector, the change rate of the objective function f due to an update with step size α in the gradient descent method can be expressed as

$$\mathbf{d}^{\mathrm{T}}\mathbf{f}'\left(\mathbf{x}_k\right)=\|\mathbf{d}\|\left\|\mathbf{f}'\left(\mathbf{x}_k\right)\right\|\cos\theta=\left\|\mathbf{f}'\left(\mathbf{x}_k\right)\right\|\cos\theta \tag{3.45}$$

where θ denotes the angle between $\mathbf{d}$ and the gradient $\mathbf{f}'(\mathbf{x}_k)$. The gradient descent method aims to decrease the objective function most effectively by

Algorithm 3.1
Standard Line Search Strategy for Optimization

```
Input:   Initial state $\mathbf{x}_0$, cost function $f(\mathbf{x})$, maximum iteration threshold $k_{\max}$
Output:  Solution state $\hat{\mathbf{x}}$
1        Initialize iteration step $k = 0$
2        **While** $f(\mathbf{x}_k)$ is not the lowest in the vicinity of $\mathbf{x}_k$ **and** $k < k_{\max}$
3            Find a descent direction $\mathbf{d}$ on $\mathbf{x}_k$ reducing $f(\mathbf{x})$
4            Find a step length $\alpha$ that $f(\mathbf{x}_k + \alpha\mathbf{d})$ is sufficiently lower than $f(\mathbf{x}_k)$
5            Update state $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\mathbf{d}$
6            Update iteration step $k = k + 1$
7        **End while**
8        Obtain solution state $\hat{\mathbf{x}} = \mathbf{x}_k$
```

aligning the update direction **d** with the negative gradient, that is, $\theta = \pi$ (or $\cos\theta = -1$). Hence, the update direction of the gradient descent method is given by

$$\mathbf{d}_{\mathrm{gd}} = -\mathbf{f}'\left(\mathbf{x}_k\right) \tag{3.46}$$

This ensures that each update step is taken in the direction of the steepest decrease of the function.

Detailed analysis of its convergence can be found in [30]. An example of state optimization using the gradient descent method is shown in Figure 3.8. The gradient descent method is easy to implement and only requires computing the first derivative of the cost function. However, it typically converges linearly, which can make it very slow during the final stage of optimization.

3.4.1.2 Gauss-Newton Method

Another popular optimization method is the Newton's method, which has a higher convergence rate compared to the gradient descent method. Instead of finding the gradient direction, the Newton method optimizes the state by finding a direction that the first derivative of cost function approaches 0, which is the necessary condition of a solution [29]. The first-order Taylor expansion of the first derivatives of the cost function after an update can be expressed by

$$\mathbf{f}'\left(\mathbf{x}_k + \mathbf{d}_n\right) \approx \mathbf{f}'\left(\mathbf{x}_k\right) + \mathbf{f}''\left(\mathbf{x}_k\right)\mathbf{d}_n \tag{3.47}$$

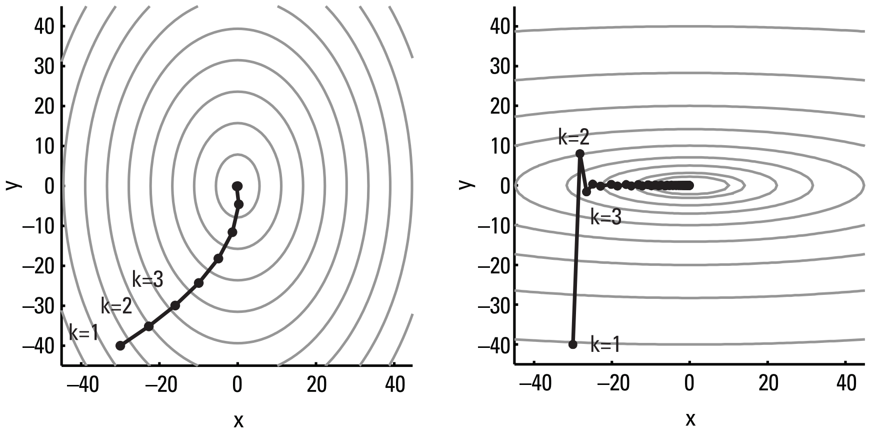


Figure 3.8 Example of gradient descent method. (Left) Good convergence. (Right) Slow convergence.

where $\mathbf{f}''(\mathbf{x}_k)$ denotes the second derivative (or Hessian) of the cost function with state $\mathbf{x}_k$. Thus, by assuming $\mathbf{f}'(\mathbf{x}_k + \mathbf{d}_n)$ approaches 0 after the update, the state update direction $\mathbf{d}_n$ can be computed by solving

$$\mathbf{f}''\left(\mathbf{x}_k\right)\mathbf{d}_n = -\mathbf{f}'\left(\mathbf{x}_k\right) \tag{3.48}$$

Then the state is updated by $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_n$. Note that the classic Newton method updates the state with a unit step length $\alpha = 1$. The Newton method can achieve a quadratic convergence rate and good for the final stage of optimization [30]. However, the Hessian $\mathbf{f}''(\mathbf{x}_k)$ needs to be positive definite to guarantee $\mathbf{d}_n$ is a descent direction for optimization. Moreover, $\mathbf{f}''(\mathbf{x}_k)$ may hard to be computed or not even available for complicated problems.

To improve the efficiency and practicability, the Newton method is modified to approximate the calculation of the Hessian $\mathbf{f}''(\mathbf{x}_k)$ by the Jacobian of the residual function $\mathbf{r}(\mathbf{x}_k)$, namely the Gauss-Newton method. The first and second derivatives of the cost function can be derived by [30]:

$$\mathbf{f}'\left(\mathbf{x}_k\right) = \mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}} \mathbf{r}\left(\mathbf{x}_k\right) \tag{3.49}$$

$$\mathbf{f}''\left(\mathbf{x}_k\right) = \mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}} \mathbf{r}'\left(\mathbf{x}_k\right) + \sum_{m=1}^{M} r_m\left(\mathbf{x}_k\right) r''_m\left(\mathbf{x}_k\right) \tag{3.50}$$

For small residuals $r_m(\mathbf{x}_k)$, $\mathbf{f}''(\mathbf{x}_k)$ is dominated by the first term, such that

$$\mathbf{f}''\left(\mathbf{x}_k\right) \approx \mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}} \mathbf{r}'\left(\mathbf{x}_k\right) \tag{3.51}$$

With this approximation, the Gauss-Newton method obtains the update direction $\mathbf{d}_{gn}$ by solving the (3.48) under (3.49) and (3.51), as follows:

$$\left(\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}} \mathbf{r}'\left(\mathbf{x}_k\right)\right)\mathbf{d}_{gn} = -\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}} \mathbf{r}\left(\mathbf{x}_k\right) \tag{3.52}$$

Then the state is updated by $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_{gn}$. A comparison of Gauss-Newton method with the gradient descent method is shown in Figure 3.9. The Gauss-Newton method avoids the computation of $r''_m(\mathbf{x}_k)$ and maintains a quadratic convergence rate in the final stage of optimization. Meanwhile, a descent direction can be guaranteed when $\mathbf{r}'(\mathbf{x}_k)$ has full rank and $\mathbf{f}'(\mathbf{x}_k)$ is nonzero. However, its convergence rate is still expected to be linear for general situations [29]. Besides, the Gauss-Newton method is not well behaved with infinite number of solutions of $\mathbf{d}_{gn}$ when $\mathbf{r}'(\mathbf{x}_k)$ is rank-deficient. It may fail to converge when having large residuals (i.e., a poor initial guess in the beginning stage of optimization [31]).

3.4.1.3 Levenberg-Marquardt Method

The Levenberg-Marquardt method [32, 33] is developed by adding a damping term on the Gauss-Newton method, which can avoid its drawbacks under rank-deficient and large residuals. One can regard the Levenberg-Marquardt method as an integration of the gradient descent method and the Gauss-Newton method [34]. The update direction of the Levenberg-Marquardt method is obtained by solving

$$\left(\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}}\mathbf{r}'\left(\mathbf{x}_k\right)+\mu\mathbf{I}\right)\mathbf{d}_{lm}=-\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}}\mathbf{r}\left(\mathbf{x}_k\right) \tag{3.53}$$

where $\mu \geq 0$ is the damping parameter and $\mathbf{I}$ is the identity matrix. The value of μ controls the update behavior during optimization. A large value of μ makes (3.53) dominated by the damping term, resulting in an update direction in the form of the gradient descent method, as follows:

$$\mathbf{d}_{lm}=-\left(\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}}\mathbf{r}'\left(\mathbf{x}_k\right)+\mu\mathbf{I}\right)^{-1}\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}}\mathbf{r}\left(\mathbf{x}_k\right)\approx-\frac{\mathbf{f}'\left(\mathbf{x}_k\right)}{\mu} \tag{3.54}$$

On the contrary, a small value of μ makes (3.53) the damping term becomes little, resulting in a dominating update direction in the form of the Gauss-Newton method, as follows:

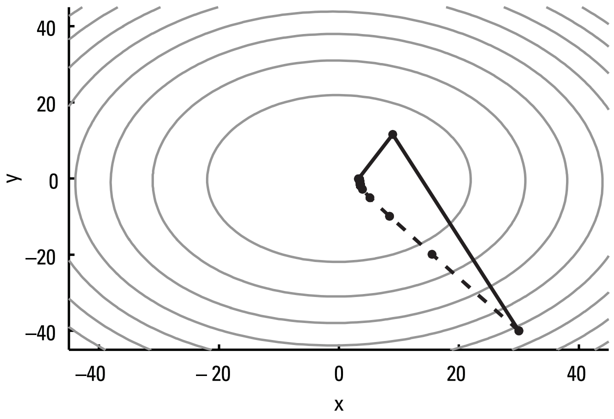


Figure 3.9 Example of optimization using the Gauss-Newton method (dash line) and the gradient descent method (solid line).

$$\begin{aligned}\mathbf{d}_{lm} &= -\left(\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}}\mathbf{r}'\left(\mathbf{x}_k\right)+\mu\mathbf{I}\right)^{-1}\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}}\mathbf{r}\left(\mathbf{x}_k\right)\\ &\approx -\left(\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}}\mathbf{r}'\left(\mathbf{x}_k\right)\right)^{-1}\mathbf{r}'\left(\mathbf{x}_k\right)^{\mathrm{T}}\mathbf{r}\left(\mathbf{x}_k\right)\end{aligned} \tag{3.55}$$

By appropriate adjustments of μ, the Levenberg-Marquardt method can behave like the gradient descent method in the beginning stage to achieve good convergence under large residuals and can behave like the Gauss-Newton method in the final stage to achieve quadratic convergence under small residuals. Note that the convergence rate of the Levenberg-Marquardt method is still linear under large residuals.

In practice, the value of μ is initialized by user experiences or the elements in $\mathbf{r}'(\mathbf{x}_k)^{\mathrm{T}}\mathbf{r}'(\mathbf{x}_k)$ [34]. Then the Levenberg-Marquardt method iteratively updates the state by $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_{lm}$ with $\mathbf{d}_{lm}$ solved from (3.53) and adjusts μ for the next update based on the gain ratio g derived from [29]:

$$g = \frac{f\left(\mathbf{x}_k\right) - f\left(\mathbf{x}_k + \mathbf{d}_{lm}\right)}{\frac{1}{2}\mathbf{d}_{lm}^{\mathrm{T}}\left(\mu\mathbf{d}_{lm} - \mathbf{f}'\left(\mathbf{x}_k\right)\right)} \tag{3.56}$$

The damping parameter μ is reduced for a large g, indicating that the update fits with the approximation (3.51) by the Gauss-Newton method, and is increased for a small g, indicating a poor approximation. The iterative update will be terminated either if the first derivative approaches 0 (i.e., $\|\mathbf{f}'(\mathbf{x}_{k+1})\|_\infty$ is sufficiently small), or the change of state $\|\mathbf{x}_{k+1} - \mathbf{x}_k\|$ is sufficiently small. The example and outline of applying the Levenberg-Marquardt method are shown in Figure 3.10 and Algorithm 3.2 [29].

Finally, we summarize the key properties of the methods discussed above (see Table 3.3).

3.5 Comparison of Estimation Methods

3.5.1 Qualitative Comparison of Estimation Methods for Indoor Positioning

First, we compare the estimation methods presented in terms of typical accuracy, computational complexity, and convergence/stability properties, particularly in the context of indoor positioning. Table 3.4 provides a qualitative summary. In the Kalman filter family, EKF is selected to compare with the others due to its popularity in IPIN applications.

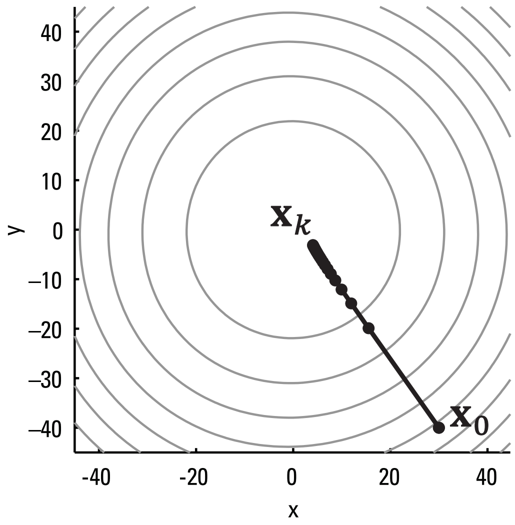


Figure 3.10 Example of optimization using the Levenberg-Marquardt method.

Algorithm 3.2
Example of Applying the Levenberg-Marquardt Method.

Input: Initial state $\mathbf{x}_0$, cost function $f(\mathbf{x})$, maximum iteration threshold $k_{\max}$, initial damping parameter μ_0, terminate threshold ε_1 for the first derivative, and terminate threshold ε_2 for the change of state.

Output: Solution state $\hat{\mathbf{x}}$

```
1   Initialize iteration step $k = 0$, $\nu = 2$ ($\nu$ is chosen as a conservative initial value to control the adjustment speed of the damping parameter)
2   Solve $(\mathbf{r}'(\mathbf{x}_0)^{\mathrm{T}}\mathbf{r}'(\mathbf{x}_0) + \mu_0\mathbf{I})\mathbf{d}_{lm} = -\mathbf{r}'(\mathbf{x}_0)^{\mathrm{T}}\mathbf{r}(\mathbf{x}_0)$
3   **While** $\|\mathbf{f}'(\mathbf{x}_k)\|_\infty > \varepsilon_1$ **and** $\|\mathbf{d}_{lm}\| > \varepsilon_2(\|\mathbf{x}_k\| + \varepsilon_2)$ **and** $k < k_{\max}$
4       Calculate gain ratio $g$ by (2.56)
5       **If** $g \leq 0$
6           Refuse poor update, $\mathbf{x}_{k+1} = \mathbf{x}_k$
7           $\mu_{k+1} = \mu_k \cdot \nu$, $\nu = 2\nu$
8       **else**
9           Update state $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_{lm}$
10          $\mu_{k+1} = \mu_k \cdot \max\{1/3, 1 - (2g - 1)^3\}$, $\nu = 2$
11      **End if**
```

Algorithm 3.2
(Cont.)

```
12        Solve $(\mathbf{r}'(\mathbf{x}_{k+1})^{\mathrm{T}}\mathbf{r}'(\mathbf{x}_{k+1}) + \mu_{k+1}\mathbf{I})\mathbf{d}_{lm} = -\mathbf{r}'(\mathbf{x}_{k+1})^{\mathrm{T}}\mathbf{r}(\mathbf{x}_{k+1})$
13        Update iteration step $k = k + 1$
14    End while
15    Obtain solution state $\hat{\mathbf{x}} = \mathbf{x}_k$
```

Table 3.3
Comparison of Numerical Optimization Methods

Optimization Algorithm	Benefits	Drawbacks
Gradient descent method	Requires only the first derivative; simple and easy to implement	Linear convergence rate; can be slow in the final stages
Newton method	Quadratic convergence rate; performs well near the optimum	Second derivative may not be available; the update direction may not always be a descent direction
Gauss-Newton method	Approximates the second derivative using the first derivative term; nearly quadratic convergence rate near the solution	Poor performance if the Jacobian is rank-deficient; may fail with large residuals
Levenberg-Marquardt method	Initially behaves like the gradient descent method, enhancing global convergence; transitions to Gauss-Newton behavior near the solution, leading to faster local convergence	May exhibit linear convergence under large residuals

Ultimately, the choice among these estimation methods depends on the specific application requirements and constraints. For pedestrian tracking scenarios where the starting position is known and motion remains smooth, a well-tuned EKF offers a computationally efficient solution with sufficient accuracy. However, if the initial position is less known or if the system must incorporate map constraints, a particle filter becomes a more suitable option due to its ability to handle arbitrary probability distributions, despite its higher computational cost. For applications demanding maximum accuracy in complex multisensor fusion, such as robotic systems integrating lidar, cameras, and IMU, FGO provides the most effective data fusion framework. FGO enables global consistency and loop closure corrections, making it ideal for long-term navigation and mapping. In such cases, a hybrid approach often combines FGO with filtering methods to balance accuracy and real-time performance.

The discussion on selecting between EKF and FGO is becoming important due to the increasing available open-source code-based FGO. Next we detail their comparison.

3.5.2 Comparison of the EKF and FGO for Sensor Integration

We also focus on comparing EKF and FGO for sensor integration, as these are the most commonly used integration methods in modern navigation systems.

Accuracy and computational efficiency are the two most crucial factors for estimators. EKF has dominated estimation applications for decades due to its maturity and efficiency in sensor integration. However, with the increasing diversity of sensors for seamless positioning, the flexibility of the estimator becomes another key factor to consider before selecting the best estimator. For instance, the addition of new sensor measurements to the sensor integration problem often requires a redesign of the majority of the matrices for the EKF.

- *Computational accuracy:* EKF can achieve optimal state estimation accuracy if two key assumptions are perfectly met: the Gaussian noise and the first-order Markov assumptions. EKF mainly considers information from two consecutive epochs, assuming that the state at the current epoch is independent of past states and measurements. However, this first-order Markov assumption is challenging to satisfy in complex scenarios where measurements are highly time-correlated. For instance, NLOS effects in the UWB range measurements can introduce time-correlated errors due to environmental reflections. Consequently, violating this assumption can lead to accuracy degradation in sensor integration with the EKF. Unlike the EKF, the factor graph utilizes information from multiple epochs simultaneously to estimate a series of states, thereby eliminating the need for the first-order Markov assumption. This approach enables the exploration of time correlation within multiple epochs simultaneously. In other words, both forward and backward optimizations are feasible with the factor graph, an undirected graphical model, whereas only forward optimization is possible with the EKF. Additionally, the linearization error resulting from

Table 3.4
Qualitative Comparison of Estimation Methods for Indoor Positioning

Method	**Accuracy**	**Computational Complexity**	**Convergence/Stability**
LS	High for linear problems, achieving optimal accuracy in the least-squares sense. For nonlinear problems, accuracy can be high if the solution converges to the global minimum, which depends on the quality of the initial guess.	A one-time batch solve (using normal equations) typically has a complexity of $O(n^3)$ for n parameters or can be solved iteratively. Efficient for moderate-sized problems.	Guaranteed global optimum for linear models. For nonlinear problems, convergence is only local (as with Gauss-Newton or Newton methods) and may lead to an incorrect solution if the initial estimate is poor or the cost surface is nonconvex.
EKF	Provides good accuracy for moderate nonlinear systems but may exhibit bias in cases of strong nonlinearities. Often sufficient but not optimal for many indoor positioning scenarios.	Same complexity as a Kalman filter, plus the additional computation of Jacobians (either analytical or numerical). Still real-time for typical state sizes.	Locally stable if the initial error is small; however, it may diverge in highly nonlinear systems or with poor initialization, as it relies on linearization. Requires careful tuning but generally converges when operated within a valid regime.

Table 3.4
(Cont.)

Method	**Accuracy**	**Computational Complexity**	**Convergence/Stability**
Particle filter	Can achieve arbitrarily high accuracy given a sufficient number of particles, approaching the true Bayesian solution as $N \to \infty$. Effectively handles multimodal distributions, which EKF/UKF may fail to capture. However, a finite N introduces approximation errors.	Per step: $O(N \cdot n)$ complexity, where each particle update contributes to the overall cost. If N is large (hundreds or thousands of particles), computation can be significant. Best suited for lower-dimensional problems or scenarios where parallelization is feasible.	Converges in probability as N increases. With a limited number of particles, it may suffer from degeneracy (where the filter loses track if no particles represent the true state). Resampling helps maintain stability but introduces random fluctuations. Generally stable if N and noise models are appropriately chosen.
FGO	Typically achieves the highest accuracy, offering global least-squares optimality under Gaussian assumptions. Minimizes drift by incorporating all available data and can correct past errors.	A batch solve can be computationally expensive, with a naive complexity of $O(T^3)$ for T states. However, sparse solvers significantly improve efficiency. Sliding window or incremental methods further reduce the ongoing cost, often achieving near-linear complexity per step for new data. Still, FGO is heavier than filtering and may run at a lower frequency in real-time applications.	Converges to the global minimum if the problem is convex (or given a good initial guess and robust cost functions for nonconvex cases). Not prone to divergence, although it may converge to a local minimum if mis-initialized. Once converged, it is generally very stable and can adjust past estimates when new information (e.g., loop closures) becomes available.

the single iteration of the EKF can also limit computational accuracy. With increasing sensor diversity, the nonlinearity of the measurement model poses new challenges for the EKF, where a single iteration may not suffice to achieve global optimality. For example, the visual feature reprojection model [35] is highly nonlinear and requires several iterations to converge. In contrast, FGO employs multiple iterations to mitigate linearization errors. A comparison between the EKF and factor graph in GNSS/inertial integration, presented in [36], demonstrates the computational accuracy of the factor graph with several datasets collected in urban areas of Hong Kong.

- *Computational efficiency:* EKF efficiently solves the sensor integration problem due to the first-order Markov assumption, as it only considers information within two consecutive epochs. In contrast, the computational time of the factor graph increases significantly with the number of epochs considered. The complexity of the optimization is dominated by the variable k. Therefore, the computational load using the factor graph can increase significantly compared to EKF-based sensor integration. To address this challenge, sliding window-based FGO optimizes only the most recent epochs of states and measurements simultaneously, achieving a balance between computational accuracy and efficiency. Specifically, states and measurements outside the sliding window are marginalized as a prior factor to constrain the factor graph, using the Schur complement [37]. As shown in Figure 3.11, the dashed areas represent the marginalized information, while the remaining states are optimized simultaneously. The states within the dashed areas are not optimized but serve as a prior constraint. An example of sliding window FGO using the Schur complement will be provided next. Incremental smoothing [22] is another strategy to maintain the computational efficiency of the factor graph by applying a Bayes tree through incremental variable reordering and fluid relinearization, thus eliminating the need for periodic batch steps. This strategy presupposes that periodic batch relinearization is unnecessary, significantly enhancing computational efficiency, even with large-scale window sizes. Notably, the implementation of incremental smoothing is open-sourced in the GTSAM library. In summary, while the computational load increases with the factor graph compared to EKF-based sensor integration, computational efficiency can be achieved by employing either sliding window optimization with marginalization techniques [37] or incremental smoothing [22].

- *Estimator flexibility:* EKF is a recursive and sequential estimator that struggles with delayed measurements due to sensor synchronization issues. Furthermore, the core equations of the EKF require redesign whenever a new sensor is incorporated, limiting its flexibility in multisensor integration. Figure 3.12 illustrates an example of rich sensor integration, incorporating UWB, inertial, and Wi-Fi measurements. Compared to Figure 3.11, the UWB range measurement is a new addition, and its corresponding factor can be expressed as: $\left\|\left\|\mathbf{p}_k - \mathbf{p}^{\mathrm{UWB}}\right\|_2 - z_k^{\mathbf{UWB}}\right\|^2_{P_{\mathrm{UWB}}}$. Here, $\mathbf{p}_k$ and $\mathbf{p}^{\mathrm{UWB}}$ represent the position state and the UWB anchor position, respectively, while $z_k^{\mathbf{UWB}}$ and P_{UWB} denote the UWB range measurement and its associated variance, respectively. If EKF is employed for sensor integration, it necessitates a redesign of the measurement model of the estimator. However, with the inclusion of UWB, an additional cost function can be seamlessly added to (3.41), which can be efficiently solved using either the Ceres Solver or the GTSAM library. More importantly, the relevant Jacobian matrix for optimizing (3.41) can be automatically calculated using the Ceres Solver or the GTSAM library, further simplifying the implementation of sensor integration. The dashed pentagon in Figure 3.12 represents the delayed UWB measurements. These delayed UWB measurements can also be accommodated in FGO, a feat difficult to achieve with the EKF. In summary, the flexibility of the factor graph

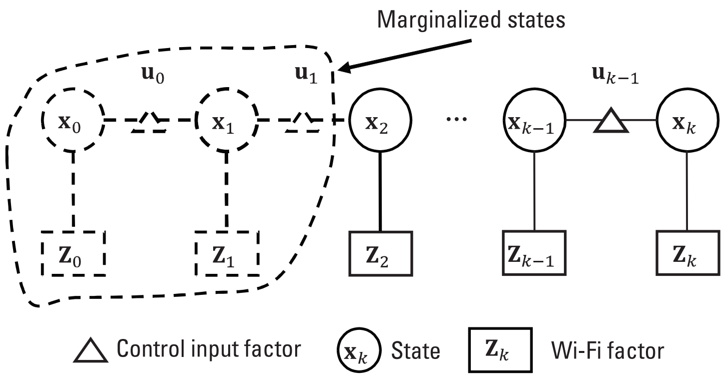


Figure 3.11 A typical example of the Wi-Fi inertial navigation problem. The triangle icon denotes the control input factor (e.g. from IMU). The rectangle icon denotes the Wi-Fi factor. The dashed areas denote the marginalized information (state and measurements).

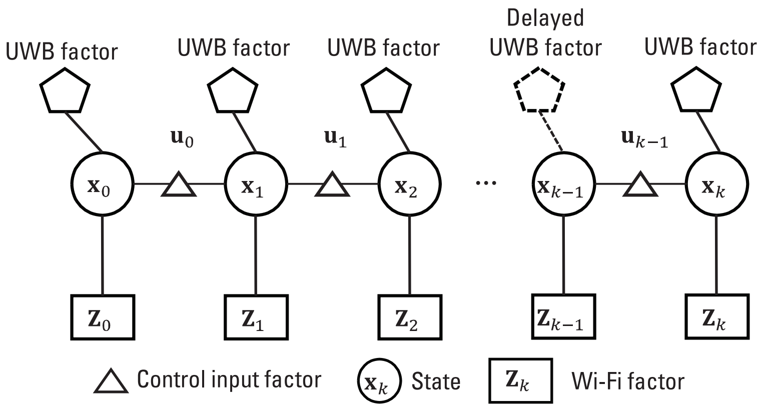


Figure 3.12 Illustration of the rich sensor integration problem, which includes the Wi-Fi, inertial, and UWB. The dashed pentagon icon denotes the delayed UWB measurements, for example, due to the sensor delay.

enhances the intuitiveness of sensor integration, providing plug-and-play capabilities.

3.6 Conclusions

This chapter reviewed key estimators and filters essential for indoor positioning. Starting with least-squares estimation for static problems, we discussed Kalman filters and variants (EKF, IEKF, and UKF), emphasizing their efficiency in real-time sensor fusion under uncertainty. These methods handle linear or linearized models and Gaussian assumptions, with UKF offering improved flexibility. Particle filters were introduced for managing non-Gaussian, multimodal uncertainties common in complex indoor environments, useful in ambiguous scenarios despite higher computational costs. FGO, performing batch or sliding-window optimization, was presented for its superior accuracy by revisiting past data, central to modern SLAM and visual-inertial navigation.

No single method dominates universally: EKF excels in speed and simplicity for small deviations; particle filters address ambiguity and complexity; and factor graphs achieve maximum accuracy given sufficient resources.

Emerging trends include automation through machine learning, distributed estimation across devices, and multimodal sensor fusion. Hybrid solutions integrating various filters and optimization techniques represent the forefront of indoor positioning, supporting critical applications such as emergency response, augmented reality, and smart buildings.

References

{AU: Call out all references sequentially in text}

[1] MacTavish, K., and T. D. Barfoot, "At All Costs: A Comparison of Robust Cost Functions for Camera Correspondence Outliers," *2015 12th IEEE Conference on Computer and Robot Vision*, 2015, pp. 62–69.

[2] Yang, H., et al., "Graduated Non-Convexity for Robust Spatial Perception: From Non-Minimal Solvers to Global Outlier Rejection," *IEEE Robotics and Automation Letters*, Vol. 5, No. 2, 2020, pp. 1127–1134.

[3] Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, Vol. 82, No. 1, March 1960, pp. 35–45.

[4] Bell, B. M., and F. W. Cathey, "The Iterated Kalman Filter Update as a Gauss-Newton Method," *IEEE Transactions on Automatic Control*, Vol. 38, No. 2, 1993, pp. 294–297.

[5] Julier, S. J., and J. K. Uhlmann, "Unscented Filtering and Nonlinear Estimation," *Proceedings of the IEEE*, Vol. 92, No. 3, 2004, pp. 401–422.

[6] Van Der Merwe, R., and E. A. Wan, "The Square-Root Unscented Kalman Filter for State and Parameter-Estimation," *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Proceedings (Cat. No. 01CH37221), Vol. 6, 2001, pp. 3461–3464.

[7] Mourikis, A. I., and S. I. Roumeliotis, "A Multi-State Constraint Kalman Filter for Vision-Aided Inertial Navigation," *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3565–3572.

[8] Zanetti, R., "Adaptable Iterative and Recursive Kalman Filter Schemes," *AAS Spaceflight Mechanics Meeting*, No. JSC-CN-30315, 2014.

[9] Huang, G. P., A. I. Mourikis, and S. I. Roumeliotis, "On the Complexity and Consistency of UKF-Based SLAM," *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 4401–4408.

[10] Kolakowski, M., "Comparison of Extended and Unscented Kalman Filters Performance in a Hybrid BLE-UWB Localization System," *2020 23rd IEEE International Microwave and Radar Conference (MIKON)*, 2020, pp. 122–126.

[11] Yang, C., W. Shi, and W. Chen, "Comparison of Unscented and Extended Kalman Filters with Application in Vehicle Navigation," *The Journal of Navigation*, Vol. 70, No. 2, 2017, pp. 411–431.

[12] Shademan, A., and F. Janabi-Sharifi, "Sensitivity Analysis of EKF and Iterated EKF Pose Estimation for Position-Based Visual Servoing," *Proceedings of 2005 IEEE Conference on Control Applications*, 2005, pp. 755–760.

[13] LaViola, J. J., "A Comparison of Unscented and Extended Kalman Filtering for Estimating Quaternion Motion," *Proceedings of the 2003 IEEE American Control Conference*, Vol. 3, 2003, pp. 2435–2440.

[14] Bloesch, M., et al., "Iterated Extended Kalman Filter Based Visual-Inertial Odometry Using Direct Photometric Feedback," *The International Journal of Robotics Research*, Vol. 36, No. 10, 2017, pp. 1053–1072.

[15] Cruz, G. P. Jr., et al., "EKF-LOAM: An Adaptive Fusion of LiDAR SLAM with Wheel Odometry and Inertial Data for Confined Spaces with Few Geometric Features," *IEEE Transactions on Automation Science and Engineering*, Vol. 19, No. 3, 2022, pp. 1458–1471.

[16] Bai, S., et al., "Towards Persistent Spatial Awareness: A Review of Pedestrian Dead Reckoning-Centric Indoor Positioning with Smartphones," *IEEE Transactions on Instrumentation and Measurement*, 2024.

[17] Xu, W., et al., "Fast-lio2: Fast Direct Lidar-Inertial Odometry," *IEEE Transactions on Robotics*, Vol. 38, No. 4, 2022, pp. 2053–2073.

[18] You, W., et al., "Data Fusion of UWB and IMU Based on Unscented Kalman Filter for Indoor Localization of Quadrotor UAV," *IEEE Access*, Vol. 8, 2020, pp. 64971–64981.

[19] Arulampalam, M. S., et al., "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking," *IEEE Transactions on Signal Processing*, Vol. 50, No. 2, 2002, pp. 174–188.

[20] Montemerlo, M., et al., "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," *AAAI/IAAI*, 2002, pp. 593–598.

[21] Barfoot, T. D., *State Estimation for Robotics*, Cambridge, UK: Cambridge University Press, 2024.

[22] Kaess, M., et al., "iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree," *The International Journal of Robotics Research*, Vol. 31, No. 2, 2012, pp. 216–235.

[23] Kaess, M., A. Ranganathan, and F. Dellaert, "iSAM: Incremental Smoothing and Mapping," *IEEE Transactions on Robotics*, Vol. 24, No. 6, 2008, pp. 1365–1378.

[24] Dellaert, F., and M. Kaess, "Factor Graphs for Robot Perception," *Foundations and Trends® in Robotics*, Vol. 6, No. 1-2, 2017, pp. 1–139.

[25] Zhao, L., Z. Mao, and S. Huang, "Feature-Based SLAM: Why Simultaneous Localisation and Mapping?" *Robotics: Science and Systems*, 2021.

[26] Forster, C., et al., "On-Manifold Preintegration for Real-Time Visual--Inertial Odometry," *IEEE Transactions on Robotics*, Vol. 33, No. 1, 2016, pp. 1–21.

[27] Grisetti, G., et al., "A Tutorial on Graph-Based SLAM," *IEEE Intelligent Transportation Systems Magazine*, Vol. 2, No. 4, 2010, pp. 31–43.

[28] Agarwal, S., and K. Mierle, "Ceres solver: Tutorial & Reference," Google Inc., No. 72, 2012, p. 8.

[29] Madsen, K., H. B. Nielsen, and O. Tingleff, *Methods for Non-Linear Least Squares Problems*, Second Edition, Informatics and Mathematical Modelling, Technical University of Denmark, 2004.

[30] Nocedal, J., and S. J. Wright, (eds.), *Numerical Optimization*, New York: Springer, 1999.

[31] Ipsen, I. C. F., C. T. Kelley, and S. R. Pope, “Rank-Deficient Nonlinear Least Squares Problems and Subset Selection,” *SIAM Journal on Numerical Analysis*, Vol. 49, No. 3, 2011, pp. 1244–1266.

[32] Levenberg, K., “A Method for the Solution of Certain Non-Linear Problems in Least Squares,” *Quarterly of Applied Mathematics*, Vol. 2, No. 2, 1944, pp. 164–168.

[33] Marquardt, D. W., “An Algorithm for Least-Squares Estimation of Nonlinear Parameters,” *Journal of the Society for Industrial and Applied Mathematics*, Vol. 11, No. 2, 1963, pp. 431–441.

[34] Gavin, H. P., “The Levenberg-Marquardt Algorithm for Nonlinear Least Squares Curve-Fitting Problems,” Department of Civil and Environmental Engineering, Duke University, August 3, 2019, pp. 1–23.

[35] Qin, T., P. Li, and S. Shen, “Vins-mono: A Robust and Versatile Monocular Visual-Inertial State Estimator,” *IEEE Transactions on Robotics*, Vol. 34, No. 4, 2018, pp. 1004–1020.

[36] Wen, W., et al., “Factor Graph Optimization for GNSS/INS Integration: A Comparison with the Extended Kalman Filter,” *NAVIGATION: Journal of the Institute of Navigation*, Vol. 68, No. 2, 2021, pp. 315–331.

[37] Leutenegger, S., et al., “Keyframe-Based Visual–Inertial Odometry Using Nonlinear Optimization,” *The International Journal of Robotics Research*, Vol. 34, No. 3, 2015, pp. 314–334.