8

Practical Aspects of Sensor Fusion

Sensor fusion is the process of combining data from multiple sensors to produce an estimate of the state of a system. By integrating diverse sensor measurements, a sensor fusion system [1] can improve positioning accuracy, increase robustness, and increase resilience against the effects of noise or uncertainty present in individual sensors. Indoor navigation systems heavily rely on sensor fusion. A pedestrian's smartphone can integrate inertial dead reckoning with signals from Wi-Fi access points or BLE beacons to overcome the lack of GPS indoors. The primary objective is to exploit complementary information. For example, one sensor may provide precise short-term motion cues while another gives absolute positioning, and their combination yields a more reliable and stable solution.

This chapter is organized as follows. Section 8.1 introduces the concepts of loosely coupled and tightly coupled sensor fusion frameworks, including their respective advantages and trade-offs, followed by a comparative summary. Section 8.2 examines the observability of sensor fusion systems, highlighting which states can and cannot be estimated reliably, along with techniques to address drift from unobservable states. Section 8.3 focuses on the tuning of sensor fusion systems, covering covariance estimation and the setting of thresholds. Section 8.4 examines calibration techniques, including intrinsic calibration for individual sensors such as cameras, LiDARs, and IMUs, as well as extrinsic calibration methods based on targets or motion. Section 8.5

addresses temporal calibration, discussing time synchronization, measurement timing, interpolation, and continuous-time methods. Section 8.6 discusses system efficiency, comparing real-time and offline processing strategies and considering memory usage. Finally, Section 8.7 summarizes key insights and practical takeaways.

8.1 Loosely Coupled and Tightly Coupled

Sensor fusion architectures are often categorized as loosely coupled (LC) or tightly coupled (TC) integration. These terms describe how deeply the sensor data are integrated with each other. In an LC approach, each sensor (or each subset of sensors) is first processed independently to produce its own state estimate, and then these estimates are combined at a higher level. In a TC approach, the raw measurements from all sensors feed into a single estimator that computes the fused state directly. The choice between LC or TC designs affects the system's accuracy, complexity, and robustness in different scenarios.

8.1.1 Loosely Coupled

Loosely coupled (LC) fusion is characterized by a two-stage process. First, each sensor is solved on its own for the quantity of interest; then those results are fused. Each sensor can function (to some degree) on its own, and their outputs are combined at the navigation solution level. This makes design and implementation straightforward. One can swap out one sensor for a better one as long as it provides the same type (meaning the same format and unit) of output, and the fusion logic remains mostly unchanged. LC fusion is also computationally lighter in many cases because each sub-estimator deals with a smaller problem.

Figure 8.1 represents a typical IPIN example of LC fusion using Kalman filters based on IMU with auxiliary sensor updates. Imagine a smartphone tracking a user's steps (via IMU) to estimate relative movement, while a secondary sensor, such as Wi-Fi, occasionally provides absolute position fixes, for example, through fingerprinting. These intermittent updates are treated as observations in a high-level Kalman filter that corrects accumulated PDR drift. A practical example is a smartphone tracking a user's steps with its IMU while periodically receiving Wi-Fi-based location estimates. The PDR runs continuously, and each Wi-Fi fix anchors the trajectory. LC designs fuse only the processed outputs of each sensor, not raw data, enabling modularity and robustness. This architecture is widely adopted in real-world indoor navigation systems.

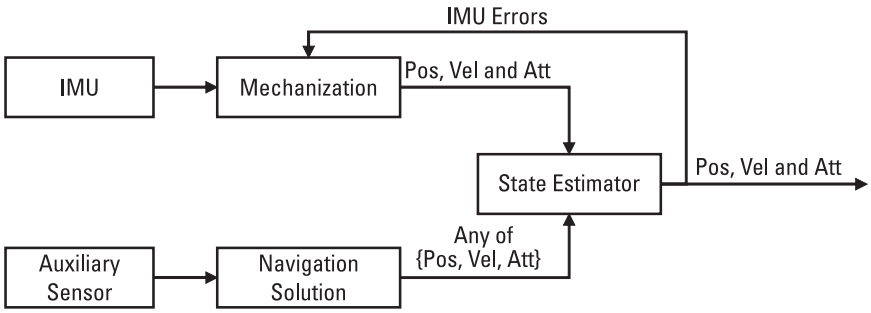

Figure 8.1 An example of a typical IMU/auxiliary sensor LC system.

However, LC systems have limitations. Because they rely on each sensor's independent solution, they may not fully utilize the information. There is information loss. For example, in an LC camera-IMU system, one might run visual SLAM to get a pose and integrate IMU for the pose, and then fuse poses. If the visual SLAM loses tracking (e.g., too few features), it provides no pose, and the fusion might fail to correct IMU drift. A tightly coupled (TC) VIO would incorporate even a few tracked features with IMU data to still estimate pose. This underscores an LC drawback: it discards raw info that the individual subsystem could not solve alone. That said, LC systems often perform well in practice and are easier to validate. They remain very popular when the sensors individually can provide good solutions most of the time.

{AU: Spell out all acronyms at first mention}

8.1.2 Tightly Coupled

Tightly coupled (TC) fusion integrates sensor data at the measurement level, in a unified estimation process. Instead of first solving each sensor's outputs separately, a TC system feeds all raw measurements (or minimally processed measurements) into a single state estimator (such as a single EKF or a bundle adjustment based on FGO). This means that the estimator simultaneously considers, for example, IMU increments, visual features, and barometer readings all at once to estimate the state. TC fusion is generally more accurate and robust, especially in difficult conditions. Because it does not require each sensor to individually produce a full solution, it squeezes the maximum information out of every observation. For example, Figure 8.2 represents a TC visual-inertial (VI) system that will treat tracked feature pixel coordinates from the camera as measurements and the IMU's readings as helping to predict motion between image frames. The estimator (often a bundle adjustment based on FGO) will solve for the camera pose and scene feature positions along with IMU biases

altogether. If at some frame only a few features are visible (insufficient for a purely visual pose estimate), the IMU's prediction plus those few features' measurements can still constrain the pose. In contrast, an LC system might have failed to produce a visual pose and thus lost track until more features appear. This is why TC VIO has become increasingly preferred in modern systems. It is considerably more robust to motion blur, low-texture scenes, or rapid maneuvers where the camera alone struggles, because the IMU is fully integrated into the estimation of camera pose [2]. The improved observability [3] (which will be discussed next) of TC systems is the main reason why the TC outperforms the LC. By merging data at the lowest level, states that would be unobservable in one sensor can become observable.

{AU: Edits correct?}

The trade-off for TC is increased complexity. The state space is larger (you might be estimating sensor biases and feature coordinates together) and the mathematical model is more complex (directly relating raw measurements to state), so the cross-correlation between measurements needs to be considered. The computational cost can be significantly higher because the estimator (especially if batch optimization like an FGO) might need to consider many raw measurements and large state vectors simultaneously. For example, a TC SLAM system with a camera and IMU might end up optimizing thousands of feature points and poses in a window. This is something an LC system might avoid by only passing along a summarized pose from the vision module. Therefore, TC systems often require more careful implementation to achieve real-time performance (e.g., efficient solvers, sparsity exploitation, and parallel processing). The Kalman filter formulation of TC integration will have a larger Kalman gain matrix to compute (dimension equals state × measurements). If one sensor provides m measurements and another provides n, the

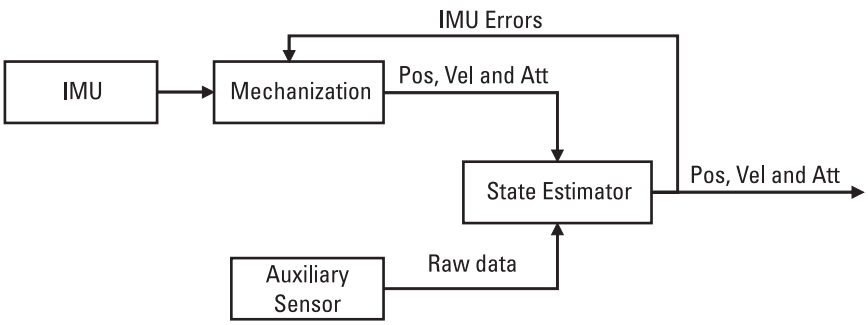

Figure 8.2 An example of a typical IMU/auxiliary sensor TC system.

combined system might have $m + n$ measurements in one update. This can be computationally heavy if m and n are large.

Despite the complexity, advancements in computing and algorithms have made TC fusion feasible for many applications. The benefit in accuracy and robustness is often well worth the extra effort. Finally, this section is concluded using the following example. Consider an autonomous drone in a tunnel: GPS signals are blocked, UWB ranging is sparse (and affected by multipath and nonline-of-sight (NLOS)), and LiDAR features may be repetitive (walls looking similar). A TC system could fuse IMU, barometer, LiDAR scan matches, and maybe vision (if active lighting is available), all in one graph optimization. Each sensor on its own might be unreliable (LiDAR scan matching could drift among similar walls, or visual features could be repetitive due to the wall markings), but, together in one estimation framework, they constrain each other. The IMU ensures continuity and short-term precision, the UWB (even 1 to 2 beacons giving partial information) provides a local reference, the barometer stabilizes altitude, and the LiDAR anchor local structure detects loop closures. Such a TC approach is resilient: losing one sensor temporarily does not break the entire solution; it just reduces the dimension of the measurements until that sensor returns. In LC, losing one sensor would remove an entire module's output from the fusion, potentially causing large errors if that module were providing vital corrections.

8.1.3 Comparative Summary

LC and TC sensor fusion methods each have distinct advantages suited for different indoor positioning and navigation scenarios. Table 8.1 summarizes their advantages and disadvantages. LC, characterized by simplicity, is advantageous when computational resources are limited, rapid prototyping is necessary, or off-the-shelf sensor solutions suffice. It performs effectively in ideal conditions but significantly degrades when sensors partially fail or data is sparse. In contrast, TC fusion integrates raw sensor data directly, achieving consistently higher accuracy, especially in challenging indoor environments such as visually sparse areas, by effectively leveraging partial measurements. However, this comes with increased computational complexity and a greater need for meticulous calibration and fault detection. LC methods excel in scalability and flexibility, enabling the easy integration of new sensors or rapid prototyping. TC systems, although more resilient in difficult conditions, demand robust fault management and careful tuning to handle sensor interdependencies. Thus, for high-performance applications

Table 8.1
Comparison Between LC and TC Sensor Fusions

Aspects	LC Sensor Fusion	TC Sensor Fusion
Data utilization	LC fuses data at the solution level.	TC fuses data at the measurement level, utilizing raw sensor outputs. This maximizes information usage.
Accuracy (performance in nominal versus degraded conditions)	LC performs optimally under nominal conditions when all sensors provide full data. In degraded environments, LC tends to lose accuracy or even fail when a sensor's data becomes unreliable or unavailable.	TC performs optimally under nominal conditions when all sensors provide full data. In degraded environments, TC achieves higher accuracy and reliability by fusing raw measurements compared with the LC integration. It can maintain performance with partial data.
Complexity (design, implementation, computational)	LC has a simpler design and implementation. Uses multiple smaller filters or modules in cascade/parallel, each handling a sensor independently. The modular approach has lower computational requirements. The development is more straightforward because sensors communicate through standard interfaces, making LC easier to implement on limited resources.	TC has a higher complexity with a centralized fusion filter. All sensor data is processed in one unified estimator, requiring a comprehensive system model. The design must handle a larger state vector (including sensor error states) and manage all sensor inputs simultaneously, which increases computational load. Its software is more intricate, needing careful calibration and synchronization of sensors. TC demands more processing power and expertise to develop.
Robustness (sensor failures and environmental challenges)	LC offers independent resilience of subsystems. Sensors operate semi-independently; each can fall back on their own if the other fails. This means that LC is robust to a complete failure of one sensor, but not to partial observation degradations.	TC is resilient with partial data, but sensitive to bad data. TC can continue operating with limited or degraded sensor inputs. Robustness in TC relies on quality sensor models and fault detection to handle sensor glitches (because all data feeds into a single filter, an outlier or fault in one sensor can contaminate the entire solution if not detected).

Table 8.1
(Cont.)

Aspects	LC Sensor Fusion	TC Sensor Fusion
Failure behavior	LC has a graceful degradation in the face of sensor failures. If one sensor degrades or fails entirely, LC experiences a gradual performance drop rather than an abrupt collapse. LC will rely on the remaining sensor(s) to continue operating.	TC has a critical dependency on all inputs. A failing sensor's data, if not handled, directly deteriorates the overall solution. In TC, sensor failures must be mitigated by the filter logic (e.g. disabling or down-weighting a faulty sensor's measurements) to avoid feeding wrong data into state estimation. This is why TC is a popular solution when companies wish to have technical barriers to their solution.
Scalability (ability to extend/ integrate additional sensors)	LC scales easily to additional sensors. New sensors can be integrated by adding modules or separate filters without redesigning the entire system architecture. The modular nature means the system can grow in a plug-and-play fashion (with an assumption in nominal condition).	TC is challenging to scale due to intertwined design. Adding or changing sensors in a tightly coupled system often requires significant rework, meaning the state vector and measurement models must be expanded and re-derived to include the new sensor data. The high interdependence means increasing the sensors can introduce complexity and unexpected interactions that need careful handling.
Real-time performance	LC is well-suited for real-time operation, even on limited hardware. Each sensor's data is processed at its native rate and fused periodically, making it efficient. This approach has historically been favored when computational resources are limited.	TC can be achieved in real time with modern computing, but the computational load is higher. TC filters process more data in one cycle, which can introduce latency (depending on the problem) if the hardware is slow.

requiring precision, reliability, and robustness indoors (e.g., autonomous robots, augmented reality, or precise indoor navigation), TC sensor fusion is advantageous. Conversely, LC approaches remain practical in simpler indoor navigation systems where ease of implementation, computational efficiency, and flexibility are prioritized.

8.2 Observability in Sensor Fusion

Observability is a concept in sensor fusion that used to describe which states of the system can be determined (or observed) from the available sensor data. A system is observable if, given the sensor measurements over time, the internal state can be uniquely inferred. Conversely, an unobservable state is one that cannot be determined based on the sensors' outputs, leading to ambiguity or drift in that dimension. Understanding observability is essential for designing fusion algorithms.

8.2.1 Observability

In multisensor fusion, certain combinations of sensors and motions can result in degenerate scenarios where some aspect of the state is not observable. Each sensor has its own limitations, and, when fusing them, one must ensure they complement each other sufficiently to cover those limitations. Challenges often arise when the sensors provide redundant information in some dimensions but none in others.

Consider LiDAR-Inertial Odometry (LIO), a 3-D LiDAR provides rich spatial information, and an IMU provides motion dynamics. In theory, a well-calibrated LiDAR+IMU can estimate a platform's full 6-DoF pose. However, there are tricky cases. If the environment has low geometric variation (e.g., a tunnel or corridor with uniform walls), as shown in Figure 8.3(a), LiDAR scan matching might struggle with degeneracy, meaning that moving forward in a tunnel produces little unique 3-D feature change, so the LiDAR essentially cannot tell if it slid forward or backward. At the same time, the IMU can sense motion, but its accelerometer bias could cause drift. Similarly, pure rotation of the sensor (standing in place and turning) gives the LiDAR no translation data; translation in X or Y is unobservable from the scans because no parallax occurs, as shown in Figure 8.3(b). The IMU in that scenario senses rotation but might be very difficult to estimate position. As a result, certain LIO implementations notice increased uncertainty or drift in unobservable directions (like along a corridor's direction or around the vertical axis in place rotation). This is fundamentally an observability issue. The sensor data (scans + IMU readings) in that scenario do not inform all degrees of freedom.

VIO has a well-known observability issue. A monocular camera on its own cannot observe the absolute scale of the environment. When an IMU is added, the combination becomes observable in scale under general motion, because the accelerometer measures actual accelerations in m/s^2, which, when compared against the camera's relative pose change (in unknown units), yields the scale. However, if the motion is insufficient (e.g., camera only rotates but

{AU: Edits correct?}

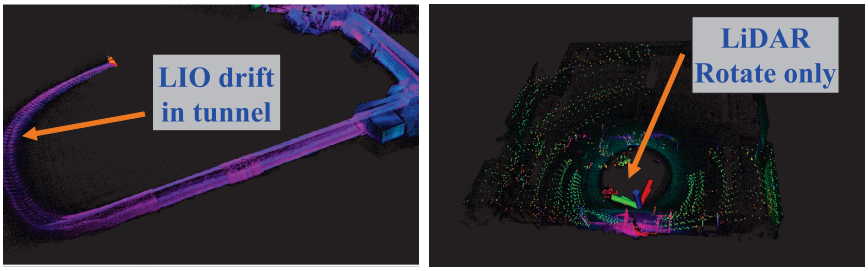

Figure 8.3 (a) The LIO drift in long tunnel. The actual path follows a straight tunnel corridor. (b) The LiDAR has rotated motion only.

does not translate), then even monocular VIO cannot determine scale; for example, the accelerometer only senses gravity and the gyros sense rotation, and there is no translational acceleration to link image movement to physical units. This is a degenerate motion for VIO, causing scale to remain unobservable during that motion [4]. Another challenge in VIO is observing the IMU biases. Typically, accelerometer biases are observable only when the device's acceleration is varied (including holding it still, as then acceleration is zero and any measured acceleration must be bias). Gyro biases are observable when there is actual rotation (no rotation expected versus some rotation measured indicates bias). If the device's motion is too limited (e.g., it never experiences certain axis movements), bias in those axes might not be fully observable and can cause drift.

Integration challenges in LIO, VIO, and Wi-Fi, often mean "what can we actually learn about the state from these sensors?" When the answer is "not everything," the missing pieces manifest as drift or ambiguity. The next sections discuss drift resulting from unobservable states and how to handle unobservable modes.

8.2.2 Unobservable States, Drift, and Solutions

When certain state variables are unobservable, the estimates for those variables can drift arbitrarily. The fusion system may assign any value to an unobservable state and still get the same predicted measurements, that meaning sensor data does not pull that state back to truth. As a result, small errors in unobservable directions accumulate over time (drift) because nothing in the measurements can correct them.

To systematically identify unobservable states in a sensor fusion system, one can perform an observability analysis [3, 5, 6] of the system's mathematical

model. This often involves computing the observability matrix (for linear systems) or using nonlinear observability tools for nonlinear systems. If the observability matrix is not full rank, it indicates that certain combinations of states are unobservable. For example, by analyzing the observability matrix for a VIO, one can pinpoint degenerate motions [7] that cause loss of observability in certain directions. In one study, Lee et al. [8] listed degenerate motions for multisensor navigation by examining the rank of the observability matrix under those motions. They could formally show, for instance, that, if a vehicle only undergoes constant velocity and constant heading (no change in those inputs), then certain biases or scale factors become unobservable. Similarly, in calibration problems, observability analysis can reveal if, say, moving with only one rotational axis active fails to observe some components of the extrinsic rotation [4] (this is related to the notion that you need excitation in all degrees of freedom).

In the observability analysis, we first derive the generally quantified rule for a nonlinear input/output model. Then we apply the rule to the Kalman filter and FGO for real state estimation applications. According to [9], the definition of locally observable (identifiable) is:

Definition 1. For a state estimation problem, a nonlinear model $\hat{\mathbf{y}} = h(\boldsymbol{\theta}, \mathbf{u}; x_0) : \Theta \rightarrow \mathcal{H}$, where $\hat{\mathbf{y}}$ is a prediction of $\mathbf{y} := [\boldsymbol{y}_1^T \ \dots \ \boldsymbol{y}_N^T]$ ($\boldsymbol{y}_k \in \mathbb{R}^p$), $\mathbf{u} := [\boldsymbol{u}_1^T \ \dots \ \boldsymbol{u}_N^T]$ ($\boldsymbol{u}_k \in \mathbb{R}^m$) is a given input vector and x_0 is an initial state vector, is called locally observable (identifiable) in $\boldsymbol{\theta}_m \in \Theta \subset \mathbb{R}^q$ if for all $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ in the neighborhood of θ_m hold that

$$\left\{h\left(\boldsymbol{\theta}_1, \mathbf{u}; x_0\right) = h\left(\boldsymbol{\theta}_2, \mathbf{u}; x_0\right)\right\} \Rightarrow \boldsymbol{\theta}_1 = \boldsymbol{\theta}_2 \tag{8.1}$$

Based on the nonlinear model, we usually consider parameter estimation by minimizing the LS function $V(\boldsymbol{\theta})$:

$$V(\theta) := \frac{1}{2}\boldsymbol{\epsilon}(\boldsymbol{\theta})^T \boldsymbol{P}_v \boldsymbol{\epsilon}(\boldsymbol{\theta}) \tag{8.2}$$

where $\in(\boldsymbol{\theta}) = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - h(\boldsymbol{\theta}, \mathbf{u}; x_0)$ is the residual vector. It is trivial to derive the Jacobian matrix $\partial V(\boldsymbol{\theta})/\partial \boldsymbol{\theta}$ and the Hessian matrix $\partial^2 V(\boldsymbol{\theta})/\partial \boldsymbol{\theta}^2$ if $V(\boldsymbol{\theta})$ is derivable. Here the parameter estimation of $\boldsymbol{\theta}$ is equivalent to minimize the LS function $V(\boldsymbol{\theta})$. Therefore, the observability condition is written as $\partial V(\boldsymbol{\theta})/\partial \boldsymbol{\theta} = 0$ (first-order necessary condition) or $\partial^2 V(\boldsymbol{\theta})/\partial \boldsymbol{\theta}^2 \succ 0$ (second-order necessary condition) at $\hat{\boldsymbol{\theta}} = \arg\min_{\theta} V(\boldsymbol{\theta})$. Next, we mainly employ the first-order necessary condition, which can also be written as $\text{rank}(\partial V(\boldsymbol{\theta})/\partial \boldsymbol{\theta}) = q$.

Based on the rule, we can get the observability matrix for an EKF:

$$\mathcal{O}_{\text{EKF}} = \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1\Phi(t_1, t_0) \\ \mathbf{H}_2\Phi(t_2, t_0) \\ \vdots \\ \mathbf{H}_T\Phi(t_T, t_0) \end{bmatrix} \tag{8.3}$$

where $\mathbf{H}_i$ is the observation matrix and $\Phi(t_i, t_0)$ is the state transition matrix of an EKF at time i. Unlike the EKF, the state transition model of FGO is included in the factors. Thus, the observability matrix of FGO is given as:

$$\mathcal{O}_{\text{FGO}} = \begin{bmatrix} \mathbf{J}_{t_0} \\ \mathbf{J}_{t_1} \\ \vdots \\ \mathbf{J}_{t_T} \end{bmatrix} \tag{8.4}$$

where $\mathbf{J}_{t_i}$ is the Jacobian matrix of residuals in FGO at time i. In conclusion, to check the observability of a state estimation model, we need to check the rank of the observability matrix $\mathcal{O}_{\text{EKF}}$ or $\mathcal{O}_{\text{FGO}}$. A full-rank observability matrix with respect to the state $\boldsymbol{\theta}$ implies that it is locally observable. If the observability matrix has rank-deficient columns for some state variables, the system cannot estimate these state variables, meaning that they are unobservable.

Once unobservable state dimensions are recognized, the next step is to mitigate their impact. The primary strategy is to add additional information to make those states observable. This could be additional sensors, as discussed: add a magnetometer to observe global heading (resolving yaw), add a GPS to observe global position and scale (resolving those drifts), and use wheel odometry to constrain motion in-vehicle navigation. Each extra sensor and/or information essentially provides equations that lock down the unobservable states. For monocular VIO, adding even occasional known distance measurements (such as detecting the floor for altitude or a known object size) can calibrate scale and prevent scale drift.

Another strategy is the use of constraints and assumptions. In some cases, certain unobservable states are not critical to the task and can be constrained by assumption. For example, in a quadrotor drone doing VIO indoors, global yaw (heading) might be unobservable and also might not be directly needed for local navigation. One could simply fix the initial yaw as the reference (basically assuming an arbitrary yaw = 0 at the start) and then the problem becomes observable in yaw relative to that reference (the system will still have yaw drift if there is bias, but one might accept slow drift or use periodic known heading

fixes if possible). In indoor pedestrian tracking, one might assume the person's start orientation or occasionally use map alignment to reset heading drift (e.g., knowing building corridor directions to snap the orientation). This approach essentially chooses a convenient value for the unobservable state rather than letting it float. In EKF terms, one might set a high uncertainty for that state but also occasionally reset if possible.

However, when unobservable states do matter to the mission (such as scale in a mapping that needs metric units), then assumptions are not enough and real measurements or extra constraints are needed. If adding sensors is not feasible (like in pure vision SLAM in unknown environment with no priors), drift in unobservable directions is inevitable. In such cases, one might at least quantify the uncertainty in those directions growing over time, so the system is aware of its limitations. For example, a SLAM algorithm can report increasing covariance on global pose, indicating growing uncertainty.

A diagnostic flow for handling unobservable states is outlined in Figure 8.4:

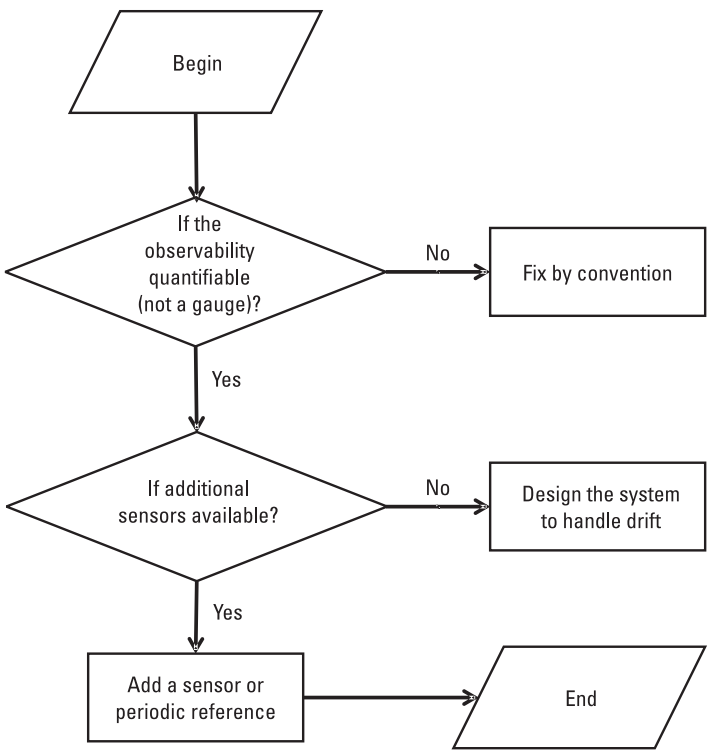

Figure 8.4 Flowchart to handle the unobservable state.

1. Determine if it is a critical physical quantity or a gauge (pure frame reference such as a missing yaw angle that we mentioned above).
2. If it is a gauge, just fix it by convention (no physical impact, just coordinate choice).
3. If it is a critical physical quantity, add a sensor or periodic reference to observe it (e.g., magnetometer for heading, altimeter for altitude, GNSSS/Wi-Fi for position).
4. If an additional sensor is not possible, then use environmental or motion constraints (nonholonomic motion constraints for vehicles, planar motion assumption, zero-velocity updates) to supply pseudo-measurements. Please refer back to Section 6.6 for details.
5. If none of the above, accept that drift will occur and design the system to handle it gracefully (perhaps by resetting estimates occasionally or warning the user of increasing error).

{AU: Correct, or should this be "GNSS"?}

{AU: Confirm this is the correct section}

Finally, one practical approach is to include redundant modalities specifically to address the unobservable aspects to make the system more robust.

8.3 Tuning of Sensor Fusion

Accurate sensor fusion relies not only on correct models and observability but also on properly tuning the assumed noise characteristics (covariances) of sensor measurements and process dynamics to make them nominal. The covariance values (process noise **Q** and measurement noise **R** in Kalman filtering terms) determine how the fusion algorithm weights different information sources. If these covariances are poorly chosen, the filter or optimizer can become overconfident in bad data or underutilize good data, leading to suboptimal or inconsistent estimates. Indoor environments pose special challenges to sensor fusion, often requiring careful tuning and additional compensation techniques. Key issues include multipath interference for RF sensors (Wi-Fi, UWB) and drift of inertial sensors due to lack of absolute references.

Tuning covariance is both a science and an art. It often requires experimentation and validation, but some techniques exist to estimate or adjust covariances systematically. Beyond initial covariance estimation, continuous adjustment and tuning of parameters is often needed as conditions change. The term "parameter" here included the noise covariances as discussed (**Q** and **R**), as well as other algorithm parameters such as outlier rejection thresholds and data association parameters. Ensuring that the fusion algorithm remains well-tuned in varied scenarios is challenging. Parameters that work well in

one environment might not in another environment. Therefore, robust sensor fusion designs incorporate mechanisms for parameter adaptation and set parameters to values that offer a good balance across conditions.

It is desired to find a region in parameter space where performance is acceptable under various conditions. This means parameter robustness. In other words, sensor fusion algorithm should not be extremely sensitive to exact parameter values. Ideally, if covariances are off by some factor, the filter should still operate without diverging (maybe just with suboptimal performance).

Achieving this robustness is a goal of tuning. One approach is stress-testing the fusion with worst-case scenarios (e.g., drop one sensor, introduce large noise) and adjusting parameters to handle those. This often leads to conservative tuning; for example, set measurement noises higher than nominal so that if an unexpected error occurs, the filter is not thrown off. It is known that an EKF can diverge if covariances are badly tuned (especially if **R** is too low relative to actual sensor noise). Thus, many engineers will intentionally inflate **R** beyond the measured noise variance [10] to account for unmodeled errors and ensure stability. Figure 8.5 illustrates the impact: with a large initial state error, using a precise **R** in Figure 8.5(a) allows fast convergence but risks instability, while an overbounded **R** in Figure 8.5(b) slows convergence but improves robustness. The EKF is sensitive to this trade-off, whereas FGO remains stable in both cases.

8.3.1 Covariance Estimation Techniques

One approach to covariance tuning is to empirically estimate noise characteristics from real data. For example, for an IMU, one can collect data with

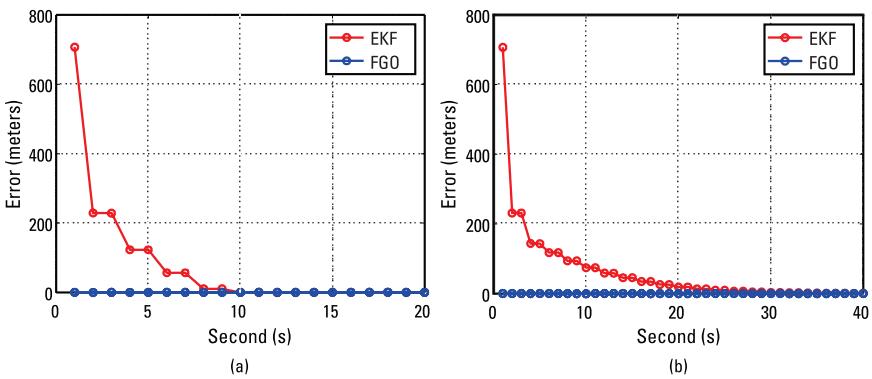

Figure 8.5 (a) Large initial guess and a precise **R**. (b) Large initial guess and an overbounded **R**.

the device static and calculate the variance of accelerometer and gyroscope readings to determine sensor noise covariance. Allan deviation plots are used in inertial sensor analysis to separate noise into bias instability, random walk, and so forth, which then informs the process noise model as we introduced in Chapter 6. For a Wi-Fi or UWB ranging sensor, one can measure the error distribution by placing the receiver at a known reference point and logging many measurements; the sample variance of the error gives the measurement noise covariance. This kind of characterization gives a starting point for **Q** and **R** values in a filter.

However, real-world conditions can deviate from the lab's precalibrated characterizations. Thus, adaptive techniques are often employed. A notable technique for state estimation filters is the adaptive EKF, which adjusts noise parameters on the fly. One such technique is innovation-based adaptive estimation: the filter monitors the statistical properties of the innovation (the difference between predicted measurement and actual measurement).

$$\tilde{y}_k = z_k - h\left(\hat{x}_{k|k-1}\right) \tag{8.5}$$

where $\tilde{y}_k$ is the innovation at time k, z_k is the actual measurement, and $h(\,)$ is the measurement function to the predicted state $\hat{x}_{k|k-1}$.

If the innovations are larger than expected (based on an assumed **R**), it indicates that the measurement noise might be higher than assumed, so **R** should be increased. Conversely, if innovations are consistently smaller, the filter might be able to decrease **R** to trust the sensor more. In practice, one can adjust **Q** and **R** in real time to keep the innovation covariance aligned with a theoretical expectation [11]. The innovation covariance is given by

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k{}^T + \mathbf{R}_k \tag{8.6}$$

where $\mathbf{S}_k$ is the innovation covariance. $\mathbf{H}_k$is the Jacobian matrix of $h(\,)$ evaluated at the predicted state (linearizes the measurement function). $\mathbf{P}_{k|k-1}$ is the predicted state covariance. For instance, many Wi-Fi /INS filters will inflate the Wi-Fi measurement noise when the pedestrian is in a signal dark zone (RF signals are likely to be blocked) by detecting the increased innovation magnitude.

There is also an approach that incorporates using robust function, such as the Huber loss [12], which takes the innovation residual as input to it. Then the weighting given by the robust function will act as a scaling factor to the noise covariance, often formulated as

$$\mathbf{R}_k = \omega\left(\tilde{y}_k\right) \cdot \mathbf{R}_k \tag{8.7}$$

where $\omega(\tilde{y}_k)$ is the weight output by the robust function based on the current innovation. This approach helps to mitigate the impact of large outliers in the measurement residuals.

Another systematic approach is using the Expectation-Maximization (EM) algorithm to estimate noise covariances. EM can treat the true trajectory as hidden data and the sensor outputs as observed data and then iterate to find the covariance values that maximize the likelihood of the observations [13]. Researchers have applied EM in Kalman filter contexts [14] to auto-tune process and measurement noise, with some success in off-line or slow adaptation scenarios.

Some recently proposed methods use multiple models (Multiple Model Adaptive Estimation) with different noise assumptions and pick the best fit. Others leverage machine learning: for instance, using reinforcement learning (RL) to tune covariance [15] has been explored (the RL agent adjusts noise values to minimize state estimate error over time). In graph-based fusion, a similar approach is dynamic covariance scaling. If a residual is large, they scale up its covariance (down-weight that measurement) to prevent it from dominating the optimization (this is akin to a robust loss function in optimization).

For Kalman filter-based fusion, the primary parameters are indeed the covariance matrices **Q** and **R**, as discussed earlier. Another important parameter is the Kalman filter initial covariance, $\mathbf{P}_0$, which represents the filter's initial uncertainty about the system state. This matrix plays a critical role during the early stages of estimation, particularly before the filter has had sufficient observations to converge. Some filters include a small stabilizing noise added to **Q** to ensure numerical stability (even if a model is deterministic). Adjusting that can affect how quickly the filter forgets past information. A common tuning trick is covariance inflation. If one suspects that the filter is too confident, one can artificially inflate **P** or **Q** slightly.

Finally, in multisensor setups, covariance estimation could also involve understanding correlations, if any. Usually, different sensors are treated as independent (no cross-covariance), but if they have common error sources (e.g., two-wheel encoders on left/right wheels have correlated errors when the surface is uneven), one might model that. However, many fusion frameworks assume a diagonal **R** matrix for simplicity (each sensor noise independent). If the correlation is significant, a more complex error model is needed (e.g., the TDOA model mentioned in Chapter 4).

8.3.2 Thresholds

Outlier rejection thresholds (such as gating thresholds for innovations that used to exclude measurements) are another important set of parameters. If too tight, good data might be discarded; if too loose, outliers slip in and disturb the estimate. Systems often adapt these based on the recent history of measurement residuals. For instance, if a particular sensor (say, a LiDAR odometry) has been consistent for a while, the system might tighten the gate threshold to be more sensitive; if it then experiences a sudden change (perhaps a difficult scene causing large residuals), an adaptive logic might widen the gate to allow data rather than reject everything (or conversely, realize lots of outliers are coming and widen to avoid triggering constant rejections). Assuming the nominal noise distribution and over a single fault at a time, a threshold based on chi-square hypothesis testing can be optimal. In this case, a measurement is accepted if the squared Mahalanobis distance,

$$d_k^2 = \tilde{y}_k^T \mathbf{S}_k^{-1} \tilde{y}_k \tag{8.8}$$

satisfies

$$d_k^2 < \chi_{m,\alpha}^2 \tag{8.9}$$

where $\tilde{y}_k$ is the innovation at time k, $\mathbf{S}_k$ is the innovation covariance, m is the measurement dimension, and $\chi_{m,\alpha}^2$ is the upper α quantile of the chi-square distribution with m degrees of freedom (e.g., $\alpha = 0.95$ for 95% confidence). However, this chi-square threshold is challenging when there are multiple faults (bias and/or non-nominal distribution).

Beyond the threshold gating, robust estimation techniques provide a principled way to handle outliers by smoothly reducing their influence rather than outright rejecting them. A common class of such methods includes M-estimators [2, 16], which replace the squared error in the measurement update step with a robust loss function that saturates for large residuals. The Huber loss [12], for instance, behaves quadratically for small residuals (like standard LS) and linearly for large ones, reducing the impact of outliers while retaining sensitivity to inliers. Another widely used robust loss is the Cauchy loss [17], which aggressively suppresses large residuals, making it more robust in high-outlier scenarios. Its influence function decreases to zero as the residual grows, effectively down-weighting extreme outliers. More sophisticated approaches such as graduated nonconvexity (GNC) [18] estimators gradually transition from a convex to a nonconvex loss landscape, allowing robust

convergence even in high-outlier regimes. These robust cost functions effectively tune the weight of each measurement in the update step, enabling the system to remain stable in the presence of unexpected disturbances without requiring manual threshold tuning.

8.4 Calibration Techniques

Accurate calibration of sensors is a fundamental prerequisite for effective sensor fusion. Calibration determines the parameters that map a sensor's raw output to real-world units or to a common reference frame. It also removes biases and systematic errors. Poor calibration can lead to large errors, because inconsistencies between sensors will propagate through the fusion algorithm. Historically, sensor calibration has been recognized as a foundational step. This includes that both an intrinsic calibration addresses each individual sensor's internal errors and an extrinsic calibration finds the spatial alignment between different sensors. Early multisensor systems often relied on careful manual calibration, meaning factory-set intrinsics and labor-intensive extrinsic using known reference objects or alignment instrumentation. Over time, techniques have evolved toward automated and online calibration, supported by deeper understanding of system observability and the development of algorithms that can estimate calibration parameters during a normal operation.

8.4.1 Intrinsic Calibration

Intrinsic calibration focuses on correcting errors and biases inherent to individual sensors, thereby ensuring each sensor's output is as accurate as possible before fusion. This involves estimating sensor-specific parameters that affect measurements in systematic ways. Intrinsic calibration is typically performed as a priori to extrinsic calibration and sensor fusion, because one must trust each sensor's readings individually before combining them. Many intrinsic parameters remain stable once determined (e.g., a camera's focal length or a LiDAR's laser firing angles are fixed after manufacturing), so one common approach is an offline calibration using controlled references.

{AU: Edits correct?}

8.4.1.1 Camera

For example, a camera's intrinsic parameters [19] include focal length, optical center, and lens distortion coefficients; calibrating these allows the camera to produce geometrically correct image measurements (e.g., undistorted pixel coordinates). A well-established method is to have the camera observe a known

pattern such as a checkerboard from various angles and distances; from the correspondences between image points and known pattern geometry, algorithms (such as Zhang's method [20]) compute the camera's intrinsic matrix and distortion coefficients. Chapter 7 has briefly introduced this technique. These techniques have become standard and highly refined over years of research [21–23] and are now routinely available in computer vision libraries such as OpenCV [24].

8.4.1.2 LiDAR

LiDARs and depth sensors may be factory-calibrated for parameters such as laser beam angles or intensity response (for example, a Velodyne LiDAR is calibrated to detect a reference reflectivity level [25]). This calibration process typically compensates for manufacturing variations across laser diodes, ensuring uniformity in both angular resolution and intensity output. Some manufacturers also calibrate for temperature drift, synchronization lag, and even individual channel timing delays. While these factory settings are sufficient for many out-of-the-box applications. In real-world deployment environments, such as those involving wide temperature ranges, aggressive motion, or sensor remounting, may require recalibration or fine-tuning. Advanced systems may perform online calibration [26] to adjust for such shifts during operation, especially when fusing LiDAR data with other sensors such as IMUs or cameras in high-precision applications.

8.4.1.3 IMU

Similarly, an IMU has biases in accelerometers and gyroscopes, scale factor errors, and axis misalignments that can be calibrated so that its reported accelerations and rotations closely reflect true motion. For IMUs, manufacturers often provide nominal bias and scale values, but each unit may still require fine-tuning. A practical example of IMU intrinsic calibration is bias and noise tuning: the IMU can be placed stationary to measure the accelerometer biases (as the average stationary output minus gravity) and gyroscope biases (average turn-rate when still), and undergoing controlled rotations to detect any scale factor errors. Such procedures, sometimes aided by statistical tools such as Allan variance analysis, yield parameters to correct the IMU's raw outputs. This was introduced in Chapter 6.

8.4.2 Extrinsic Calibration

Beyond intrinsic factors, multisensor systems require spatial calibration to align sensors in space. Extrinsic calibration determines the rigid-body transformation

(usually, 6-DoF) between sensors' coordinate frames. Accurate spatial calibration is essential for sensor fusion; the misalignment of a few centimeters can degrade fusion accuracy or even make the sensor data inconsistent.

This calibration finds the rotation and translation that map one sensor's coordinate system into that of another. For example, if a LiDAR and camera are mounted on a vehicle, extrinsic calibration will yield the 3-D rotation and translation {**R**, **t**} that transforms a point observed by the LiDAR into the camera's coordinate frame. These extrinsic parameters allow fusing of data, such as projecting LiDAR points onto camera images or combining their measurements for obstacle detection. There are two broad approaches for extrinsic calibration: target-based and motion-based methods, as shown in Figure 8.6.

8.4.2.1 Target-Based Calibration

In target-based calibration, one uses specially designed objects or patterns visible to multiple sensors to provide known reference points. Common targets include checkerboard patterns [19] (seen in camera images and also detectable as planes in 3-D LiDAR), geometric arrangements of retro-reflectors or LED markers, or calibration objects with multiple modalities (e.g., a planar board with an AprilTag [27] for camera and reflective surfaces for LiDAR). By collecting sensor data of the same target and running optimization, the relative pose of sensors can be solved. For example, a camera-LiDAR calibration might involve placing a checkerboard in view of both the camera and LiDAR; the camera sees the checkerboard corners while the LiDAR perceives the planar board. By aligning these, it yields the transform between camera and LiDAR frames. Target-based methods can achieve high accuracy but require a controlled setup.

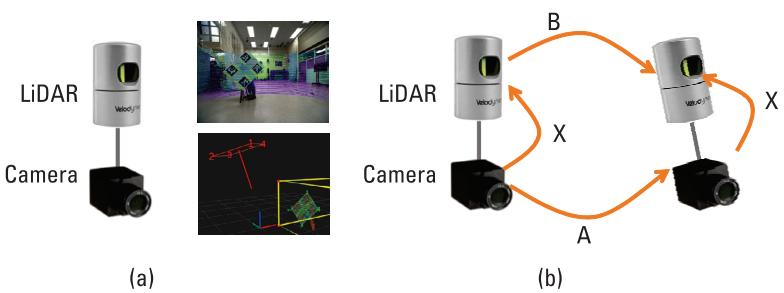

Figure 8.6 (a) The target-based method. (b) The motion-based method where **A** and **B** are camera and LiDAR motions, and **X** is the extrinsic transformation solved from **AX** = **XB**.

8.4.2.2 Motion-Based Calibration

Motion-based calibration does not require a special object; instead, it leverages the sensors' own observations of the environment or their motion. A classic example is the hand-eye calibration [28] technique in a multisensor context. If one sensor can estimate its motion (e.g., an IMU provides relative orientation and position via integration, or a LiDAR SLAM gives pose changes), and another sensor also measures motion or has observations that depend on the same motion, the relative pose can be inferred by aligning the two motion estimates. These targetless methods are convenient as they use natural features (such as alignments of 3-D point clouds and camera features) or common motion. However, they can be sensitive to the environment. Both sensors must observe enough corresponding structure or motion for the calibration to converge [29]. Furthermore, certain motions can lead to ambiguities: for full 6-DoF extrinsic calibration, the platform should experience rotational and translational motion in all axes (to make sure the observability is strong enough); purely planar motion, for example, might leave the relative roll angle unobservable. As noted earlier, a lack of excitation in some degrees of freedom means those parts of the transform cannot be determined [4].

8.5 Temporal Calibration

In addition to spatial calibration, temporal calibration is equally critical in modern sensor fusion systems, particularly when fusing high-rate, asynchronous sensors such as cameras, IMUs, LiDARs, and UWB receivers. Even with perfectly calibrated intrinsics and extrinsics, discrepancies in timestamp alignment between sensors can introduce significant errors. Small timing offsets could lead to large spatial misalignments. Temporal calibration ensures that the data from different sensors is correctly synchronized, either by estimating fixed time delays or by modeling clock drift.

8.5.1 Synchronization

Proper time synchronization ensures that sensor measurements used together in the fusion algorithm indeed correspond to the same real-world moment. In a sensor fusion scenario with different sensor modalities, each device might have its own oscillator (clock) and sampling rate, leading to potential misalignment in time. Even a small timestamp error can introduce significant fusion errors, especially in high-speed or highly dynamic systems. For example, considering fusing an IMU reading taken slightly before a camera image, the

IMU might indicate a rotation that the camera has not yet observed, causing an inconsistency in estimation. Therefore, aligning data streams in time is crucial. In SLAM and other TC fusion systems, synchronization errors can result in ghosting or smearing of features and degraded state estimates [29], ultimately causing drift or divergence in the solution.

Figure 8.7 illustrates three representative time synchronization scenarios commonly encountered in VIO systems. In the ideal case (Case 1), the IMU and camera are well synchronized, with their measurements occurring in consistent alignment. This is typically achieved through precise hardware triggering or the use of a shared clock source. However, deviations from this ideal are common in practice. Case 2 shows a situation where the camera and IMU share the same clock but have different internal trigger times, resulting in a fixed offset between their measurement events. This misalignment, if uncorrected, can introduce significant temporal errors into the fusion process. Case 3 depicts a more challenging scenario in which the IMU and camera operate on independent time sources. This leads not only to a constant offset but also to time-varying drift, where the relative timing between sensor streams changes over time. To achieve accurate temporal calibration in such systems, two problems must be addressed: (1) a constant offset between sensor clocks, as shown in Case 3 in Figure 8.7 (one sensor's timestamps are always, say, 5 ms behind that of the other sensor), and (2) the drift (one sensor's clock runs slightly faster, so the offset changes over time). A straightforward approach to handle a known constant offset is to subtract it from timestamps, but finding that offset requires precise calibration. Often, a synchronization calibration is done by observing a common event. For example, if a camera sees an LED turning on at time t_{cam} and an IMU detects a vibration (from the LED's control circuit) at time t_{imu}, one can deduce an initial offset. More systematically, many systems adopt a common time base via hardware: all sensors are GPS timestamped or triggered by the same clock signal [30]. This hardware synchronization is ideal because it eliminates relative drift (all sensors effectively become one clock). It is commonly used in research platforms (for instance, a UAV might distribute a 1-Hz PPS from a GNSS receiver or use IEEE 1588 PTP [31] to all sensors, aligning their times to within microseconds.

{AU: Spell out all acronyms at first mention}

{AU: Spell out all acronyms at first mention}

Without hardware synchronization, a software one is needed. Filters and smoothing algorithms can estimate time offset as part of the state. For example, a VIO system might include a time offset parameter between a camera and IMU and adjust it by observing that IMU-reported motions consistently lead or lag the camera's observed motions, as shown in Figure 8.8. Another approach is using filtering on timestamps themselves. Techniques treat the

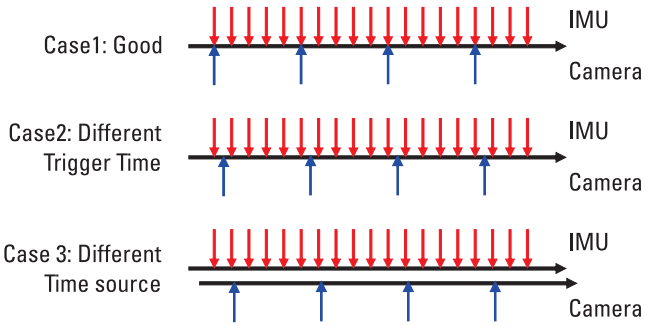

Figure 8.7 Illustration of different time synchronization cases in the VI system.

sensor clocks as having unknown skew and offset and use timestamp exchange or common observations to estimate those parameters [32]. One study [33] introduced a passive sync algorithm that monitors the discrepancy in sensor timestamps and can significantly reduce sync error even when sensors have unknown latency and drift. In essence, it periodically recalculates how one clock is drifting relative to another and corrects future timestamps accordingly.

Sensor fusion algorithms can be made somewhat robust to small sync errors. For example, increasing the assumed measurement noise for asynchronous sensors can prevent the filter from overtrusting possibly mis-timed data (essentially blurring the measurement in time). Some advanced algorithms even estimate an effective time offset online by examining innovation residuals. If the data consistently arrives "late" (causing a predictable residual pattern), the filter can infer a time delay and compensate. In critical systems, where an out-of-sync sensor reading could cause a hazard (e.g., an autonomous car mis-localizing itself), designers often include fail-safes, meaning that if synchronization uncertainty becomes too high, the system might temporarily reduce reliance on that sensor or issue an alert.

Figure 8.8 Illustration of time offset in a VIO system.

8.5.2 Measurement Timing

Due to differing sensor update rates, a fusion system often needs to interpolate measurements so that information is available on a common timeline or at desired intervals. For example, an IMU might provide readings at 200 Hz, while a Wi-Fi receiver outputs positions at 5 Hz. If the fusion algorithm operates at 100 Hz (perhaps tied to a control loop), it must integrate these inputs appropriately. It can use each new IMU reading as it arrives, but only occasionally get a Wi-Fi update. Measurement interpolation fills the temporal gaps by estimating what a sensor reading would be at an intermediate time. This is crucial when combining fast sensors with slower ones. Without interpolation, one might have to wait for the slow sensor or else process asynchronous data with complex handling of time delays.

8.5.2.1 Direct Measurement Interpolation

{AU: Do you mean "t_0 and t_1"?}

{AU: Spell out all acronyms at first mention}

The simplest approach is linear interpolation in time. If a sensor provides readings at times *t_0* and *t_1*, one can linearly interpolate to get an estimate at *t* in between. For instance, if a magnetometer gives heading at 1 Hz, but an AHRS filter runs at 50 Hz, the magnetometer heading can be linearly interpolated for intermediate filter steps. Linear interpolation is computationally trivial and works reasonably if the sensor signal changes slowly or nearly linearly between samples. However, many sensor quantities are not linear over time (e.g., angles on a circle or an object accelerating). Linear interpolation of orientation can cause issues because rotations live on a manifold (a non-Euclidean space). In such cases, specialized interpolation [34] such as spherical linear interpolation (SLERP) for quaternions is used to smoothly interpolate orientations on the unit sphere. For example, given two orientation quaternions at times *t_0* and *t_1*, SLERP provides a steady rotation at constant angular velocity between them, avoiding anomalies such as gimbal lock that could occur with naive linear interpolation of Euler angles (as we mentioned in Chapter 2). Similarly, one can linearly interpolate position coordinates if acceleration is mild; if acceleration is significant, linear interpolation will lag behind the actual curved trajectory.

{AU: Do you mean "t_0 and t_1"?}

8.5.2.2 Continuous-Time Method

To improve on linear assumptions, one can use polynomial splines (e.g., cubic splines) or other smooth interpolation methods. A spline fit through a sequence of sensor samples can provide continuously differentiable estimates and better capture the dynamics. These methods are known as continuous-time

methods. For example, an odometer that outputs distance every second could be interpolated with a cubic spline to estimate distance at 0.1-second intervals, preserving continuity of velocity. Engineers often use splines for IMU orientation integration or to interpolate delayed measurements. A special case is orientation: techniques such as SQUAD (Spherical and Quadratic interpolation) [35] extend SLERP to multiple segments, ensuring a smooth orientation trajectory through several keyframes. Spline-based interpolation adds some computational cost (solving for polynomial coefficients) but offers greater accuracy when sensor data can be assumed to vary smoothly.

A more advanced method treats interpolation as a probabilistic regression problem. One can model the sensor's time series as a Gaussian process, which provides a continuous function that best fits the known samples and a measure of uncertainty. GP interpolation (also called Bayesian interpolation) [36] can be very powerful, especially if the sensor noise characteristics are known. For example, continuous-time SLAM approaches [37] sometimes use GP regression to represent a robot's trajectory as a continuous function, allowing sensors such as IMUs and cameras to be fused without discrete resampling. A GP can naturally incorporate uncertainty. If there is a large gap between sensor updates, the interpolation uncertainty grows, which the fusion algorithm can take into account. The downside is computational complexity. GP inference involves inverting covariance matrices, which can be expensive as the number of points grows. Researchers have developed efficient GP frameworks for sensor fusion, but they are heavier than simpler interpolation schemes. Figure 8.9 illustrates the difference between cubic spline interpolation and GP regression for continuous-time estimation from discrete sensor data samples.

{AU: Please spell out all acronyms at first mention}

8.5.2.3 Physics-Based Method

There is a classic trade-off when choosing an interpolation method. Direct linear interpolation is fast and often good enough for small gaps, but can introduce error if the sensor reading is highly nonlinear over the interval. High-order interpolation (splines and GP) can be more accurate but costs more CPU and potentially adds latency. In real-time systems, one usually cannot perform very heavy interpolation computations at a high rate. A practical strategy is to use physics-based models for interpolation. For example, in between Wi-Fi fixes, one can integrate IMU readings (assuming constant acceleration between IMU samples, effectively a linear model in acceleration), which are a form of model-based interpolation for the Wi-Fi position. This leverages the IMU to provide a physically realistic interpolation of position and velocity, rather than just a mathematical spline.

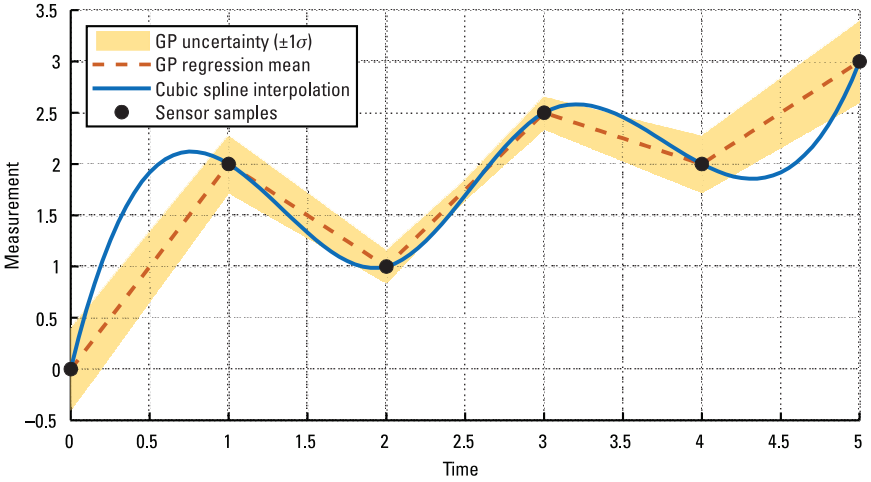

Figure 8.9 Illustration of continuous-time methods for sensor data interpolation. Discrete sensor samples (black dots) are interpolated using a cubic spline (solid blue curve), providing a smooth and continuously differentiable estimate. GP regression (orange dashed curve) provides both a continuous estimate and a measure of uncertainty (shaded region), which grows in areas with sparse samples. These approaches allow for high-resolution and uncertainty-aware trajectory estimation in sensor fusion systems.

8.6 Efficiency

In designing sensor fusion algorithms, especially for real-time positioning, efficiency is a key consideration. Efficiency can refer to computational complexity (how the algorithm scales with number of states or measurements), memory usage, and the ability to meet real-time deadlines with limited hardware. There is often a trade-off between the accuracy of an algorithm and its complexity. More accurate methods (such as batch optimization) tend to be more computationally expensive than simpler approximations (such as a basic Kalman filter). The choice between real-time versus offline processing also influences which algorithms are viable. This section discusses strategies to achieve real-time performance, as well as considerations for offline processing where latency is not an issue.

8.6.1 Real-Time Versus Offline

Real-time systems require algorithms that can guarantee an update within a fixed time budget for each sensor input. For example, if IMU comes at 200

Hz (5-ms period), the fusion algorithm must handle prediction at 200 Hz. If camera frames at 30 Hz provide updates, the system has ~33 ms to integrate the IMU between frames and incorporate the vision update. A VIO might do a lot of work for each frame (feature tracking, marginalization of the old keyframe, solving a small BA problem). Systems such as VINS-Mono [32] achieve this by limiting the number of feature points and keyframes in the optimization (keeping it to a sliding window of maybe 5 to 10 keyframes) so the optimization remains small (state vector of maybe 50 variables, solved in tens of milliseconds). They also use multithreading to do feature tracking in parallel with state estimation. The complexity per frame is roughly proportional to the number of features tracked and number of iterations in the solver; with 100 features and few iterations, it fits in 33 milliseconds on mobile CPUs. If one tried to use thousands of features, it would not meet real-time. Thus, practical efficiency often requires trading off some accuracy (not using all features or marginalizing some poses) for speed.

{AU: Please spell out all acronyms at first mention}

Efficiency can be improved by using specialized hardware. Many modern sensor fusion tasks offload some computations to GPUs (which is specialized in visual processing and deep learning parts, if any). However, classical filter matrix operations can also use linear algebra libraries that exploit CPU vector instructions or multicore. Ensuring real-time performance often involves profiling the algorithm, identifying bottlenecks (such as a big matrix inversion or a resampling step), and optimizing those (maybe using block matrix math or reducing the frequency of heavy steps).

Offline processing (mapping or smoothing after data collection) can afford heavier computation. For example, one might run a full bundle adjustment on hours of data, which might take minutes or hours on a PC/server to get the most accurate map. For online use, one might run a reduced algorithm and then refine offline. Some mapping solutions indeed do a coarse real-time pose estimation and log data and then do a big optimization offline to refine the map (useful for applications such as surveying where final accuracy is needed but not in real time).

8.6.2 Memory Considerations

Efficiency also includes memory usage. A particle filter with many particles or an EKF-SLAM with large covariance matrix can be memory heavy. For instance, storing a 1,000 × 1,000 covariance matrix (for 1,000 landmarks) is 10^6 elements, which is fine (8 million bytes if double) on a PC but might be heavy on a microcontroller. Sparse methods alleviate memory by storing only nonzero entries. Factor graphs typically store factors in memory that

scale with number of measurements (which can be large but manageable if using pointers to data and streaming methods). Two examples on quantifying memories are given:

- *Example 1:* Particle filter versus EKF for indoor localization—If the state is just (x, y, θ) for a person, an EKF is trivial to run (constant time basically per step). A particle filter with 1,000 particles might also be fine but is certainly heavier (1,000 times the motion and measurement evaluation). On a smartphone, 1,000 particles can be processed in a few milliseconds, which is still fine. If more complex sensors such LiDAR scans are used, evaluating a scan's likelihood for each particle is heavy. One might downsample the scan or reduce particle count to compensate. There is a known approach. A particle filter often is used when there are multimodal uncertainties, but, once localized, one might switch to an EKF for efficiency. Some systems implement a dual mode: use particle filter for global localization and then hand off to an EKF or a single-hypothesis smoother for precise tracking once confident. This leverages efficiency. A particle filter (with more cost) is only used when needed.
- *Example 2:* For a 2-D SLAM scenario with 500 3-DOF poses and 200 2-DOF landmarks, an EKF would manage a 1,900-dimensional state vector and a correspondingly dense $1{,}900 \times 1{,}900$ covariance matrix. The computational complexity of each EKF update is O(n2) (where $n = 1{,}900$), which critically limits scalability as the map grows. In stark contrast, FGO represents this problem using a sparse graph structure with 700 nodes (500 poses and 200 landmarks). This sparsity allows the full optimization problem to be solved efficiently using sparse factorization techniques, while incremental methods such as iSAM [38] can provide efficient online updates. Consequently, FGO methods offer substantially superior scalability compared to EKF as the environment size and trajectory length increase.

8.7 Conclusions

Sensor fusion has advanced significantly, enabling robust positioning and navigation in scenarios that would be impossible with any single sensor. By carefully calibrating sensors, synchronizing their data streams, and choosing appropriate fusion architectures (loose versus tight coupling), engineers have

built systems that exploit the best aspects of each sensor to compensate for the weaknesses of others. Throughout this chapter, we discussed that effective fusion requires attention to practical details: calibration ensures that sensor outputs are comparable; time alignment prevents spurious errors; observability analysis tells us which quantities we can actually estimate; and the tuning of noise covariances and other parameters is key to filter consistency and performance in the field.

In conclusion, sensor fusion is a cornerstone of modern navigation and situational awareness systems. It embodies the saying "the whole is greater than the sum of its parts," as fused systems achieve robust performance that no individual sensor could accomplish. The progress in sensor fusion algorithms (from Kalman filters to graph optimizers) and computing power has continually expanded the horizons of what fused sensors can do, enabling everything from a smartphone guiding you through a mall to a robotics autonomously exploring a underground tunnel site. As sensors proliferate and diversify, effective fusion will remain a critical and evolving discipline, bridging the gap between raw sensor outputs and coherent understanding of position and movement in the world.

In terms of trends, computational improvements and theoretical advances are allowing more complex fusion to run in real time. We see autonomous robotics using factor graphs with hundreds of landmarks and states on the fly, and smartphones performing AI-based sensor fusion (such as Google's Visual Positioning Service, which fuses camera images with GNSS and map data). The line between mapping and localization has become a blur. Many systems simultaneously build a map and use it, effectively doing SLAM, which is a grand application of sensor fusion bringing together IMUs, LiDAR, vision, wheel encoders, Wi-Fi, and GNSS.

References

[1] Song, Y., and L. -T. Hsu, "Tightly Coupled Integrated Navigation System Via Factor Graph for UAV Indoor Localization," *Aerospace Science and Technology*, Vol. 108, 2021, p. 106370.

[2] Bai, X., W. Wen, and L.-T. Hsu, "Robust Visual-Inertial Integrated Navigation System Aided by Online Sensor Model Adaption for Autonomous Ground Vehicles in Urban Areas," *Remote Sensing*, Vol. 12, No. 10, 2020, p. 1686.

[3] Zheng, X., W. Wen, and L. -T. Hsu, "Tightly-Coupled Visual/Inertial/Map Integration with Observability Analysis for Reliable Localization of Intelligent Vehicles," *IEEE Transactions on Intelligent Vehicles*, 2024.

[4] Walters, C., et al., "A Robust Extrinsic Calibration Framework for Vehicles with Unscaled Sensors," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 36–42.

[5] Hermann, R., and A. Krener, "Nonlinear Controllability and Observability," *IEEE Transactions on Automatic Control*, Vol. 22, No. 5, 1977, pp. 728–740.

[6] Yang, Y., and G. Huang, "Observability Analysis of Aided INS with Heterogeneous Features of Points, Lines, and Planes," *IEEE Transactions on Robotics*, Vol. 35, No. 6, 2019, pp. 1399–1418.

[7] Huang, G. P., A. I. Mourikis, and S. I. Roumeliotis, "Observability-Based Rules for Designing Consistent EKF SLAM Estimators," *The International Journal of Robotics Research*, Vol. 29, No. 5, 2010, pp. 502–528.

[8] Lee, W., C. Chen, and G. Huang, "Degenerate Motions of Multisensor Fusion-Based Navigation," *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 8113–8119.

[9] Van Doren, J. F., et al., "Identifiability: From Qualitative Analysis to Model Structure Approximation," *IFAC Proceedings Volumes*, Vol. 42, No. 10, 2009, pp. 664–669.

[10] Simon, D., *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*, New York: John Wiley & Sons, 2006.

[11] Sun, W., P. Sun, and J. Wu, "An Adaptive Fusion Attitude and Heading Measurement Method of MEMS/GNSS Based on Covariance Matching," *Micromachines*, Vol. 13, No. 10, p. 1787, 2022, https://www.mdpi.com/2072-666X/13/10/1787.

[12] Huber, P. J., *Robust Statistical Procedures*, SIAM, 1996.

[13] Dempster, A. P., N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data Via the EM Algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, Vol. 39, No. 1, 1977, pp. 1–22.

[14] Gao, X., et al., "A Novel Residual-Based Bayesian Expectation–Maximization Adaptive Kalman Filter with Inaccurate and Time-Varying Noise Covariances," *Measurement*, Vol. 235, 2024, p. 114937.

[15] Gu, J., J. Li, and K. Tei, "A Reinforcement Learning Approach for Adaptive Covariance Tuning in the Kalman Filter," *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, Vol. 5, December 16–18, 2022, pp. 1569–1574.

[16] Lucas, A., "Robustness of the Student T Based M-Estimator," *Communications in Statistics-Theory and Methods*, Vol. 26, No. 5, 1997, pp. 1165–1182.

[17] Hampel, F., et al., *Robust Statistics: The Approach Based on Influence Functions*, Wiley, 1986.

[18] 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.

[19] Oth, L., et al., "Rolling Shutter Camera Calibration," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1360–1367.

[20] Zhang, Z., "Camera Calibration with One-Dimensional Objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 26, No. 7, 2004, pp. 892–899.

[21] De la Escalera, A., and J. M. Armingol, "Automatic Chessboard Detection for Intrinsic and Extrinsic Camera Parameter Calibration," *Sensors*, Vol. 10, No. 3, 2010, pp. 2027–2044.

[22] Gennery, D. B., "Generalized Camera Calibration Including Fish-Eye Lenses," *International Journal of Computer Vision*, Vol. 68, No. 3, 2006, pp. 239–266.

[23] Muglikar, M., et al., "How to Calibrate Your Event Camera," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1403–1409.

[24] Bradski, G., "The OpenCV Library," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, Vol. 25, No. 11, 2000, pp. 120–123.

[25] R. Bergelt, O. Khan, and W. Hardt, "Improving the Intrinsic Calibration of a Velodyne LiDAR Sensor," *2017 IEEE SENSORS*, 2017, pp. 1–3.

[26] Lv, J., et al., "Observability-Aware Intrinsic and Extrinsic Calibration of LiDAR-IMU Systems," *IEEE Transactions on Robotics*, Vol. 38, No. 6, 2022, pp. 3734–3753.

[27] Huang, J. K., and J. W. Grizzle, "Improvements to Target-Based 3D LiDAR to Camera Calibration," *IEEE Access*, Vol. 8, 2020, pp. 134101–134110.

[28] Horaud, R., and F. Dornaika, "Hand-Eye Calibration," *The International Journal of Robotics Research*, Vol. 14, No. 3, 1995, pp. 195–210.

[29] Yeong, D. J., et al., "Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review," *Sensors*, Vol. 21, No. 6, 2021, p. 2140.

[30] Geiger, A., et al., "Vision Meets Robotics: The Kitti Dataset," *The International Journal of Robotics Research*, Vol. 32, No. 11, 2013, pp. 1231–1237.

[31] Idrees, Z., et al., "IEEE 1588 for Clock Synchronization in Industrial IoT and Related Applications: A Review on Contributing Technologies, Protocols and Enhancement Methodologies," *IEEE Access*, Vol. 8, 2020, pp. 155660–155678.

[32] 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.

[33] Olson, E., "A Passive Solution to the Sensor Synchronization Problem," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 18–22, 2010, pp. 1059–1064.

[34] Buss, S. R., and J. P. Fillmore, "Spherical Averages and Applications to Spherical Splines and Interpolation," *ACM Transactions on Graphics (TOG)*, Vol. 20, No. 2, 2001, pp. 95–126.

[35] Robeson, S. M., "Spherical Methods for Spatial Interpolation: Review and Evaluation," *Cartography and Geographic Information Systems*, Vol. 24, No. 1, 1997, pp. 3–20.

[36] Peršić, J., et al., "Spatiotemporal Multisensor Calibration Via Gaussian Processes Moving Target Tracking," *IEEE Transactions on Robotics*, Vol. 37, No. 5, 2021, pp. 1401–1415.

[37] Anderson, S., and T. D. Barfoot, "Towards Relative Continuous-Time SLAM," *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 1033–1040.

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